Australian Government
Department of Defence
Defence Science and
Technology Group

RANS Simulations using OpenFOAM Software

D.A. Jones[1], M. Chapuis[3], M. Liefvendahl[3], D. Norrison[1], and R. Widjaja[2]

[1]Maritime Division
[2]Aerospace Division
Defence Science and Technology Group

[3]Defense Security Systems Technology,
Swedish Defence Research Agency - FOI,
SE 147 25 Tumba, Stockholm, Sweden

DST-Group-TR-3204

ABSTRACT

The use of the OpenFOAM software suite for the performance of Reynolds-Averaged Navier-Stokes (RANS) simulations is described and illustrated by applying the simpleFoam solver to two case studies; two dimensional flow along a curved ramp and three dimensional flow along the hull of a generic conventional submarine. A detailed description of the use of the code is provided and aspects of the discretization schemes, solution solvers, turbulence schemes and boundary conditions are discussed.

RELEASE LIMITATION

Approved for public release

Published by

Maritime Platforms Division
DST Group Defence Science and Technology Group
506 Lorimer St
Fishermans Bend, Victoria 3207 Australia

Telephone: 1300 333 362
Fax: (03) 9626 7999

AR-016-502
January 2016

APPROVED FOR PUBLIC RELEASE

RANS Simulations using OpenFOAM Software

Executive Summary

OpenFOAM offers considerable advantages for Computational Fluid Dynamics (CFD) simulations; it is open source, free of charge, and has the ability to run in parallel over large processor arrays. The purpose of this report is to illustrate and test the use of the steady-state Reynolds Averaged Navier-Stokes (RANS) solver in OpenFOAM (simpleFoam), as well as some of the more useful utilities, by applying it to two relatively simple case studies; two dimensional flow down a curved ramp, and three dimensional flow over the bare hull of a generic conventional submarine model.

Because running a CFD simulation using OpenFOAM is quite different to running a simulation using commercial software packages, the first part of the report provides a detailed description of the basic file structure used for all OpenFOAM simulations. The content and purpose of each of the files is discussed, as are some of the useful utilities, such as those for mesh generation and running a simulation in parallel over distributed processors. Next an overview of the OpenFOAM RANS solver, simpleFoam, is provided. This includes a description of the way the equations are discretized and the algorithms used to solve the resulting matrix equations. Useful methods for monitoring the progress of a simulation during run time are also described.

Flow over a contoured ramp is then used to illustrate how simpleFoam is used to perform a two dimensional RANS simulation using both the k-ε and k-ω SST turbulence models both with and without the use of wall functions. Appropriate boundary conditions are described for both models and the simulated results are compared with experimental results. The improvement which can be obtained with the use of more refined meshes and the k-ω SST turbulence model is demonstrated. Flow around the bare hull of a generic conventional submarine model is then used to illustrate the ability of the code to model three dimensional flows and to quantify the sensitivity of the simulated results to the choice of discretization method for one of the important terms in the equations. The results are compared with both model scale data [24] as well as with simulation results from the commercial code Fluent. It is concluded that OpenFOAM provides an ability to perform RANS simulations of submarine bodies and has performance similar to that of commercial software packages.

This page is intentionally blank

Authors

David A. Jones

Maritime Division

David obtained a B.Sc. (Hons) and Ph.D. in Theoretical Physics from Monash University. He joined the Materials Research Laboratories in 1983 after postdoctoral positions at the University of Strathclyde, London University, and the University of New South Wales. In 1987 he was a visiting scientist at the Naval Research Laboratory, Washington, DC. He has authored 90 journal articles and technical reports and given more than 60 presentations at scientific meetings. His research has covered polymer dynamics, chaos theory, atomic and molecular physics, laser-plasma interactions, warhead design, air blast, detonation physics and computational fluid dynamics. Since moving into the Hydrodynamics Group in the Maritime Platforms Division he has been simulating fluid flow around ships and submarines using finite element codes, Lagrangian vortex element methods, Smoothed Particle Hydrodynamics and finite volume codes.

Mattias Liefvendahl

Defence Security Systems Technology,
Swedish Defence Research Agency – FOI

Mattias obtained his PhD in Numerical Analysis at the Royal Institute of Technology, Stockholm, Sweden in 2001. He joined the Swedish Defense Research Agency (FOI) in 2003 and is currently Deputy Research Director for computational and theoretical hydrodynamics. He is also adjoint professor in hydrodynamics at Chalmers University of Technology. Mattias has been active in research in numerical analysis of partial differential equations, electromagnetism, nonlinear stability, optimization and large-deformation computational structural mechanics. Present research activities concern theoretical and computational hydrodynamics, with application to the development and design, as well as operational aspects of, marine platforms and sensors. Flow-induced ship signatures, primarily for submarines, are of particular interest.

Michel Chapuis

Defence Security Systems Technology,
Swedish Defence Research Agency – FOI

Michel holds a MSc degree in vehicle engineering and joined the Swedish Defence Research Agency (FOI) after graduating from the Royal Institute of Technology in 2009. In late 2011 he started working within the private industry as a consultant at the company Xdin. Michel has been working in research projects with computational fluid dynamics focusing on hydrodynamics, aerodynamics and combustion physics with applications such as submarines, surface ships, high speed trains, heavy trucks, gas turbines, ram/scramjet engines and rockets.

Ronny Widjaja

Aerospace Division

Ronny Widjaja graduated from RMIT University in 2001 with a Bachelor degree in Aerospace Engineering. He was awarded a Doctor of Philosophy degree from the University of Melbourne based on his research study in Computational Aero-Acoustics. In 2005, he joined AeroStructures Pty Ltd as an aeronautical consultant and developed a CFD model for the P-3C Orion aircraft. Between 2006 and 2011, he worked for the Maritime Platforms Division of the Defence Science and Technology Organisation. His main role was to lead the submarine manoeuvring research within the Hydrodynamics Group. He worked on submarine propeller modelling during this period and completed a CRC project on advanced composite propellers. In 2012, he moved to the Aerospace Division where he applies CFD tools/capabilities to support ADF clients with their helicopter aerodynamics problems. His areas of research include ship-helicopter operation, concurrent helicopter operation, helicopter brown-out and store release.

Daniel Norrison

Maritime Division

Dr. Daniel Norrison obtained a Bachelor of Engineering (Aerospace Engineering) and Bachelor of Applied Science (Mathematics) from RMIT University in 2004 before joining the Defence Science and Technology Organisation (DSTO) Australia in 2009. He completed a Ph.D. in 2009 at RMIT University in the area of methods to accelerate grid generation and computational fluid dynamics computations for computational aeroelasticity problems. He is currently a Research Scientist in the Maritime Platforms Division where he conducts modelling and prediction of the hydrodynamics of maritime vehicles and their propulsion systems.

Contents

This page is intentionally blank

1. Introduction

OpenFOAM stands for "Open Source Field Operation and Manipulation" and is a library of object oriented software written in the C++ programming language [1]. It contains a large number of solvers for CFD simulations, ranging from solvers for potential flow (potentialFoam), transient laminar flow (icoFoam), time-dependent turbulent flow (turbFoam), incompressible steady-state turbulent flow (simpleFoam), flow on dynamic meshes (icoDyFoam), compressible steady-state flow (rhoSimpleFoam), multiphase flow (interFoam) and Large Eddy Simulation (LES) solvers (oodles), amongst others. OpenFOAM can be run in parallel on distributed processors using the public domain openMPI implementation of the standard message passing interface (MPI). As such, it offers the potential to perform large scale CFD simulations without incurring the significant licence fees concomitant with using commercial software packages.

OpenFOAM offers considerable advantages for CFD simulations; it is open source, therefore the user has access to the source code and can modify the code for individual use. It is free of charge, and the ability to run in parallel over large processor arrays makes it attractive for simulations on large scale meshes, such as those encountered in naval platform simulations. The (initial) disadvantages of the software however are that there is limited documentation (other than access to the code itself), and for a user without prior knowledge of C++ there is a steep learning curve. In addition to the solvers listed above, OpenFOAM also has a large number of utilities for pre-processing and post-processing of results. The purpose of this report is to test and illustrate the use of the steady-state RANS solver in OpenFOAM (simpleFoam), as well as some of the utilities relevant to submarine flows, by applying it to two relatively simple case studies; two dimensional flow down a curved ramp, and three dimensional flow over the bare hull of a generic conventional submarine model. The use of the software for LES studies and a detailed comparison between the RANS results obtained from the commercial software package Fluent and OpenFOAM will be described elsewhere.

2. OpenFOAM software: case files and case structure

Running a CFD simulation using OpenFOAM is quite different to running a simulation using commercial software packages such as Fluent or CFX. The commercial solvers use a Graphical User Interface (GUI) for case setup and data entry whereas for OpenFOAM the case setup is done using text files in a specific folder structure. Figure 1 shows the basic file structure which must be used for all OpenFOAM simulations.

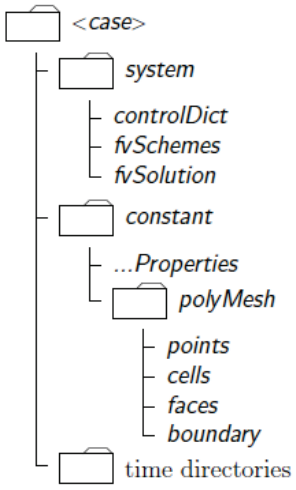

Figure 1: File structure for a typical OpenFOAM simulation

The system directory contains three files which must be present for any OpenFOAM simulation; the controlDict, fvSchemes and fvSolution files. The controlDict file controls the overall running of the simulation. For a transient solver it specifies the solver to be used, the start time, end time, time step, and information which specifies when data is to be written to an output file and the format of that data. For a steady state solver the start time and end time identify the start and end of the iteration loop and the time step is the iteration counter. An example of a controlDict file is shown in Appendix A for a steady state simulation for the two-dimensional ramp case. The purgeWrite entry is an integer value that specifies how many output time directories are stored. When the maximum number of purgeWrite directories has been written the newest time data will overwrite the oldest time directory. Specifying purgeWrite = 0 means that no time data will be overwritten. As mentioned by Shan et al. [2], specifying the name of the solver in the application entry is not essential and has no effect on the run. The specific solver is actually specified at run time by issuing the solver name in the case file directory to initiate the simulation. The controlDict file may also contain statements which print the force and moment data on a body located in the flow field. This will be explained in more detail in section 5 when discussing flow around a generic conventional submarine hull.

The fvSchemes file contains the specification of the numerical discretization schemes used for each of the terms in the governing partial differential equations. Consider, for example, the momentum equation for incompressible laminar flow. In vector form this is:

$$\frac{\partial \mathbf{U}}{\partial t} + (U \bullet \nabla)\mathbf{U} - \nabla \bullet (\nu \nabla \mathbf{U}) = -\frac{1}{\rho}\nabla p$$

and the solution of this equation in the icoFoam solver (transient laminar flow) is represented as:

```
Solve
(
    fvm::ddt (U) + fvm::div (phi, U) – fvm::laplacian (nu, U) = =   - fvc::grad (p)
);
```

The discretization schemes used for the time dependence, divergence, laplacian and gradient terms are all specified in the fvSchemes file. The content of this file obviously depends on the type and number of the equations to be solved. Appendix A shows an example of the fvSchemes file for the simpleFoam solver (incompressible, steady state, turbulent, single phase) used for the two-dimensional ramp case. The example is a RANS simulation using the *k*-ε turbulence model and hence additional terms are required in the fvSchemes file to represent the divergence and laplacian terms for the *k* and ε equations.

Once each of the terms in a given equation has been discretized appropriately the various terms are added together to form a large non-linear matrix equation which is then solved iteratively. The numerical solvers for each equation are specified by the user in the fvSolution file. For a general asymmetric matrix, three classes of iterative solvers are available in OpenFOAM:

- PBiCG – Preconditioned bi-conjugate gradient method
- GAMG - Geometric algebraic multigrid method
- smoothSolver – the Gauss-Seidel method.

The author's experience has shown that the GAMG solver works well on simple geometries and is faster than the PBiCG solver in these cases. For more complicated geometries however, such as fully appended submarines, the GAMG solver has been found to have stability problems in some cases. The PBiCG solver may sometimes be slower than the GAMG solver but it has better stability properties and hence is the recommended solver for three dimensional flows. For matrices which are both symmetric and positive definite, for example when solving the pressure equation, a Preconditioned conjugate gradient (PCG) solver is available with a Diagonal Incomplete Cholesky (DIC) factorization preconditioner.

In the fvSolution file shown in Appendix A the PCG solver with a DIC preconditioner has been used to solve the pressure equation and the PBiCG solver with a Diagonal

Incomplete Lower Upper (DILU) preconditioner has been used for the momentum, k and ε equations. The matrix solvers are iterative and based on reducing the equation residual to within a predetermined value over a successive number of iterations. The residual is evaluated by substituting the current solution into the equation and taking the magnitude of the difference between the left and right hand sides; it is also normalized to make it independent of the scale of the problem being analyzed. Before solving an equation for a particular field, the initial residual is evaluated based on the current values of the field. The user can specify both an absolute tolerance and a relative tolerance. After each solver iteration the residual is re-evaluated. The solver stops if either of the following conditions is satisfied:

1. The residual falls below the absolute tolerance.

2. The ratio of current to initial residuals falls below a specified relative tolerance, relTol.

The absolute tolerance represents the level at which the residual is small enough that the solution is considered to be sufficiently accurate. The solver relative tolerance limits the relative improvement from initial to final solution.

For the pressure equation a typical value for the tolerance is 10^{-8}, while for all other variables a value of 10^{-6} is usually sufficient. The relative tolerance is usually set to 0.01. The fvSolution file also specifies the algorithm used for the pressure-velocity coupling and values for the relaxation factors, if these are required.

Depending on the size of the simulation or the type of post-processing to be performed the system folder may also contain additional files. For a large simulation the user may want to run OpenFOAM in parallel over multiple processors. In this case the system folder will contain a decomposeParDict file, which specifies the way in which the mesh will be divided amongst the number of processors. For post-processing OpenFOAM provides the sample utility for sampling field data, either along a one-dimensional line for graph plotting or on a two-dimensional plane for displaying as isosurface images. The sampling locations are specified through a sampleDict file located in the system directory. Both the decomposePar and sample utilities are executed in the normal manner by issuing the commands decomposePar or sample from the case directory. More information on both of these utilities can be found in the User Guide [3].

The constant directory contains two files which describe the physical properties of the system and the turbulence model used. The file transportProperties describes the physical properties of the fluid. For a steady state incompressible solver only the kinematic viscosity ν is required. It should be noted that the steady state solvers in OpenFOAM work with kinematic pressure, p/ρ, where ρ is the density. The turbulence model is specified in the RASProperties file. This specifies both the turbulence model and the various parameters needed for the model. The constant directory also contains the polyMesh sub-directory, which contains all the files describing the geometry and the mesh. The mesh for an OpenFOAM run can be created in a number of ways. OpenFOAM provides two utilities for mesh generation; blockMesh and snappyHexMesh, and also allows third party meshes to be imported. The simplest OpenFOAM mesh generator is the blockMesh utility, which generates a fully hexahedral mesh from a dictionary file named blockMeshDict which is located in the constant/polyMesh subdirectory of the case directory. The blockMesh utility reads the dictionary file, generates

the mesh and writes the mesh data to the points, owner, neighbour, faces and boundary files in the same directory. The blockMeshDict file which was used to create the mesh for the curved ramp case is shown in Appendix A. The snappyHexMesh utility generates three dimensional hexahedral and split-hexahedral meshes automatically from triangulated surface geometries in stereo lithography (.stl) format. The mesh conforms to the surface by iteratively refining a starting mesh and "snapping" the resulting mesh to the surface geometry. The utility also has the ability to shrink back the resulting mesh and to insert cell layers between the surface and the mesh to better resolve boundary layers. The snappyHexMesh utility is not without problems however and some of these are discussed by Tapia [4] and by Järpner [5].

A number of commercial mesh generation software packages can also be used to create meshes which can then be converted to OpenFOAM format using mesh conversion codes in OpenFOAM. The submarine hull mesh in section 5 was created using the ICEM software package and then exported as a .msh file which was then converted to OpenFOAM format using the fluentMeshToFoam utility. For this approach to work the file must be exported in ASCII format, which is not the default option in Fluent. The Gridgen software package has also been used to create meshes for OpenFOAM. Gridgen will export a Fluent case file in ASCII format which can then be converted to OpenFOAM format using the newer fluent3DMeshToFoam utility. The one problem to note with this approach is that Gridgen inserts commas in the comments section of the case file (ANSYS products do not do this), which causes the converter to fail. The user can either remove the comments from the case file before running the converter, or modify the fluent3DMeshtoFoam utility to ignore the commas. The latest version of the Pointwise meshing software (the successor to Gridgen) has an internal converter which allows the user to output the mesh directly in OpenFOAM format.

When using Fluent, the user is able to choose the mesh boundary conditions from a simple menu, while the initial values of all variables are defined during the solution initialization step. For OpenFOAM, the initial values and boundary conditions are defined in a separate file for each variable. These files are located in the 0 subdirectory of the OpenFOAM case file. For a steady state RANS simulation using the k-ε turbulence model and OpenFOAM version 1.5 the appropriate files are p, U, k and epsilon. For example, the velocity boundary conditions are set in the file U with patches corresponding to the boundary types set in the boundaries file in the polyMesh folder. Appendix A provides examples of these files for the two-dimensional ramp case. In OpenFOAM version 1.5 the specification of a "wall" patch name in the boundaries file automatically assigns wall functions to the wall boundary patches. In OpenFOAM version 1.6 this is no longer the case; as well as specifying a patch name of "wall" for the appropriate patches in the boundaries file the user must also include a nut (turbulent viscosity) file in the 0 directory and specify a nutWallFunction type on the wall boundaries in the nut file, an epsilonWallFunction on corresponding patches in the epsilon file, and a kqRwallFunction on the corresponding patches in the k file.

When generating a mesh using non OpenFOAM software and converting to OpenFOAM format using one of the OpenFOAM supplied converters, it is important to check the patch names in the boundaries file generated by the converter with the patch names used in the variable files in the 0 subdirectory. The file structure, including the contents of the 0 subdirectory, is usually created by copying an existing file structure from a similar case directory. When this procedure is followed there is often a mismatch between the patch names

in the different files which needs to be manually corrected before OpenFOAM will run successfully.

3. Understanding simpleFoam

To gain a better understanding of the required entries in the fvSchemes file we need to look at the simpleFoam coding. The basic code, simpleFoam.C, is shown in Appendix B. The important lines are:

```
{
#       include "UEqn.H"
#       include "pEqn.H"
}
turbulence -> correct();
```

The files *UEqn.H* and *pEqn.H* are also included in Appendix B. The first entry, include "UEqn.H", solves the Reynolds-Averaged momentum equation. The equations describing incompressible Newtonian flow have the form [6]:

$$\rho \frac{\partial u_i}{\partial t} + \rho \frac{\partial}{\partial x_j}\left(u_i u_j\right) = -\frac{\partial p}{\partial x_i} + \mu \nabla^2 u_i , \qquad \frac{\partial u_i}{\partial x_i} = 0 \tag{1}$$

To form the Reynolds-Averaged equations the variables **u** and p are written as the sum of a mean and a fluctuating part:

$$\mathbf{u} = \langle \mathbf{u} \rangle + \mathbf{u}' , \quad p = \langle p \rangle + p' \tag{2}$$

Averaging the above equations over an ensemble of systems then produces the RANS equations in the form:

$$\rho \frac{\partial \langle u_i \rangle}{\partial t} + \rho \frac{\partial}{\partial x_j}\left(\langle u_i \rangle \langle u_j \rangle\right) = -\frac{\partial \langle p \rangle}{\partial x_i} + \mu \nabla^2 \langle u_i \rangle + \frac{\partial \tau_{ij}}{\partial x_j}, \qquad \frac{\partial \langle u_i \rangle}{\partial x_i} = 0 \tag{3}$$

where τ_{ij} is the Reynolds stress tensor, $\tau_{ij} = -\rho\langle u_i' u_j' \rangle$. To close this system of equations τ_{ij} needs to be expressed in term of the mean variables. One way of doing this is to assume that the deviatoric part of the Reynolds stress tensor is proportional to the rate of strain tensor of the mean flow and to write:

$$-\rho\langle u_i' u_j' \rangle + \frac{2}{3}\rho\delta_{ij}k = 2\rho\mu_T \langle S_{ij} \rangle \tag{4}$$

where $$\langle S_{ij} \rangle = \frac{1}{2}\left(\frac{\partial \langle u_i \rangle}{\partial x_j} + \frac{\partial \langle u_j \rangle}{\partial x_i} \right), \; k = \frac{1}{2}\langle u_i' u_i' \rangle \tag{5}$$

and μ_T and k are the eddy viscosity and the kinetic energy of the turbulent fluctuations.

The RANS momentum equation then takes the form:

$$\rho \frac{\partial \langle u_i \rangle}{\partial t} + \rho \frac{\partial}{\partial x_j}\left(\langle u_i \rangle \langle u_j \rangle \right) = \mu_{\text{eff}} \nabla^2 \langle u_i \rangle - \frac{\partial}{\partial x_i}\left(\langle p \rangle + \frac{2}{3}\rho k \right) \tag{6}$$

where $\mu_{eff} = \mu + \mu_T$. This is the same as equation (1) with $\langle u_i \rangle$ and μ_{eff} in place of u_i and μ and with $\langle p \rangle + \frac{2}{3}\rho k$ appearing as a modified mean pressure.

The expression for μ_T depends on the particular type of turbulence model being used. For the standard k-ε model it has the form:

$$\mu_T = C_\mu \rho \frac{k^2}{\varepsilon} \tag{7}$$

where ε is the total rate of dissipation in the flow. The standard equations for k and ε are:

$$\rho \frac{\partial k}{\partial t} + \rho \langle u_j \rangle \frac{\partial k}{\partial x_j} = 2\mu_T \langle S_{ij} \rangle \frac{\partial \langle u_i \rangle}{\partial x_j} - \rho\varepsilon + \frac{\partial}{\partial x_j}\left[\left(\mu + \frac{\mu_T}{\sigma_k} \right) \frac{\partial k}{\partial x_j} \right] \tag{8}$$

$$\rho \frac{\partial \varepsilon}{\partial t} + \rho \langle u_j \rangle \frac{\partial \varepsilon}{\partial x_j} = C_{\varepsilon 1} P_k \frac{\varepsilon}{k} - C_{\varepsilon 2} \rho \frac{\varepsilon^2}{k} + \frac{\partial}{\partial x_j}\left[\left(\mu + \frac{\mu_T}{\sigma_\varepsilon} \right) \frac{\partial \varepsilon}{\partial x_j} \right] \tag{9}$$

and the model constants are given by [6]:

$$C_\mu = 0.09,\ C_{\varepsilon 1} = 1.44,\ C_{\varepsilon 2} = 1.92,\ \sigma_k = 1.0,\ \sigma_\varepsilon = 1.3$$

OpenFOAM uses a cell centred finite volume method to solve the partial differential equations of continuum mechanics and fluid flow. In this approach the equations are integrated over each of the control volumes (cells) on the mesh and volume integrals that contain a divergence term are converted to surface integrals using Gauss's theorem. The surface integrals can then be evaluated by summing the contributions from each of the cell faces. This approach to the solution of the equations requires a method to extrapolate the velocity stored at the centroid of the cell to the value of the velocity at the face of the cell. Many methods to perform this extrapolation are available in OpenFOAM and these are documented in the Users Guide [3]. The incompressible solvers in OpenFOAM work with kinematic viscosity ν rather than dynamic viscosity μ, where $\nu = \mu/\rho$, hence μ, μ_T and μ_{eff} in the above equations are replaced by ν, ν_T and ν_{eff}, which in simpleFoam appear as nu, nut and nuEff. The flux at a cell face f, $\mathbf{U}_f \bullet \mathbf{S}_f$, where U_f is the velocity at the cell face and S_f is the cell face normal vector, is denoted by phi.

To solve equation (6) the fvSchemes file required by simpleFoam contains entries for div(phi,U), laplacian(nuEff,U) and div(nuEff*dev(grad(U).T()))). The first of these represents the convective flux term on the left hand side of equation (6), while the remaining two terms, when added together, represent the Laplacian term on the right hand side of the equation. This is explained quite clearly in the report by Furbo [7].

As well as solving the momentum equation for the velocity the continuity equation must also be satisfied. This is done by taking the divergence of the momentum equation and then substituting this into the continuity equation to produce a Poisson equation for the pressure. This is the equation solved in pEqn.H and explains the presence of the laplacian((1 | A(U)),p) entry in the fvSchemes file. The form of this equation is quite complicated and various approximation schemes have been devised to solve the equation. simpleFoam uses the SIMPLE (Semi-Implicit Method for Pressure Linked Equations) algorithm first proposed by Patankar and Spalding [8]. When implemented on a collocated mesh (as in OpenFOAM) the SIMPLE algorithm can produce pressure oscillations in the solution. This problem is avoided in simpleFoam by implementing a version of the Rhie-Chow interpolation scheme [9]. The particular version used and its implementation in OpenFOAM is described in detail by Kärrholm [10].

The momentum and pressure equations are then solved in an iterative manner. First the momentum equation is solved to compute an intermediate velocity field. The pressure equation is then solved to provide a correction to the pressure field and then the velocities are updated on the basis of the new pressure field. The fluxes at the cell faces and the boundary conditions are then updated and the process is repeated until convergence is obtained. In practice it has been found that the above procedure can become unstable. Stability can be improved with a suitable selection of relaxation factors limited to a range between 0 and 1. For the pressure equation the recommended value for the relaxation factor is 0.3 while for the remaining equations the value 0.7 is recommended. These are the default values found in the fvSolutions file, however the user is able to change these if smaller values are required to prevent the residuals from undergoing extreme oscillations. The above description provides a rather brief account of the SIMPLE algorithm and a more detailed discussion can be found in many references [11, 12]. The Ph.D. thesis of Jasak [13] also provides a detailed description of the algorithm as used in OpenFOAM.

If the standard k-ε turbulence model is being used then the equations for k and ε also need to be solved to enable the turbulent viscosity to be updated. In simpleFoam this is done with the line "turbulence->correct();". In this expression "turbulence" is a pointer to the appropriate turbulence model object and the arrow symbol dereferences the pointer to get back to the object. "correct" is a member function of the turbulence model which solves the k and ε equations. The terminology indicates that the function corrects the k and ε fields by updating them from the values they held at the previous iteration step. From equations (8) and (9) we can see that simpleFoam requires expressions for the convective flux term for both k and ε as well as Laplacian terms involving $\nu + \nu_T/\sigma_k$ for the k equation and $\nu + \nu_T/\sigma_\varepsilon$ for the ε equation. The k-ε turbulence model in OpenFOAM is coded in the file kEpsilon.C, from which one can see that DkEff = $\nu + \nu_T/\sigma_k$ and DepsilonEff = $\nu + \nu_T/\sigma_\varepsilon$. This explains the entries div(phi,k), div(phi,epsilon), laplacian(DkEff,k) and laplacian(DepsilonEff,epsilon) in the fvSchemes file.

A more detailed discussion of the implementation of the *k*-ε model in OpenFOAM is provided in the report by Furbo [7].

From the above description it is clear that use of OpenFOAM requires the user to have a greater knowledge of the physics of the turbulence models being employed than is the case when using some of the commercial CFD codes. The experienced user will be aware of the equations being solved for a particular turbulence model and hence will be able to provide the appropriate entries in the fvSchemes file from the outset. Less experienced users can arrive at the same result by simply changing the specification of the turbulence model in the RASProperties file and then attempting to run the code. If the fvSchemes file has not been properly configured then the code will print error messages which will list the terms which need to be added to enable a solution to be found.

The progress of an OpenFOAM run can be monitored visually from run time plots of the residuals by using the open source plotting program Gnuplot. A portion of the output from the log file for a two-dimensional *k*-ε turbulence model run is shown in Figure 2. A simple Gnuplot script which reads the log file is shown in Figure 3 and the residual plot produced from this script is shown in Figure 4.

```
Time = 10
DILUPBiCG:  Solving for Ux, Initial residual = 0.204726, Final residual = 9.21631e-08, No Iterations 6
DILUPBiCG:  Solving for Uy, Initial residual = 0.194801, Final residual = 4.73701e-07, No Iterations 6
DICPCG:  Solving for p, Initial residual = 0.190084, Final residual = 8.88088e-09, No Iterations 543
time step continuity errors : sum local = 7.55915e-07, global = 4.53851e-09, cumulative = -4.61396e-08
DILUPBiCG:  Solving for epsilon, Initial residual = 0.0398468, Final residual = 1.9845e-07, No Iterations 5
DILUPBiCG:  Solving for k, Initial residual = 0.0305759, Final residual = 4.24999e-07, No Iterations 5
ExecutionTime = 34.73 s  ClockTime = 35 s
Time = 11
DILUPBiCG:  Solving for Ux, Initial residual = 0.58514, Final residual = 2.79129e-07, No Iterations 6
DILUPBiCG:  Solving for Uy, Initial residual = 0.11231, Final residual = 1.77404e-07, No Iterations 6
DICPCG:  Solving for p, Initial residual = 0.0421649, Final residual = 9.14012e-09, No Iterations 533
time step continuity errors : sum local = 3.10804e-06, global = -5.91926e-09, cumulative = -5.20588e-08
DILUPBiCG:  Solving for epsilon, Initial residual = 0.0377455, Final residual = 2.28548e-07, No Iterations 5
DILUPBiCG:  Solving for k, Initial residual = 0.028526, Final residual = 5.57131e-07, No Iterations 5
ExecutionTime = 38.69 s  ClockTime = 39 s
```

Figure 2: Sample output from a two-dimensional simpleFoam run using the k-ε turbulence model

The sample script shown in Figure 3 uses the standard linux utilities cat, grep, cut and tr to extract the relevant numbers from the log file. The cat utility reads the log file and the pipe command (shown as a vertical line) then inputs the file to the grep utility which searches through the file to find the line containing the specified string of characters, such as "Solving

for Ux". The cut utility then selects the appropriate number from this line and the output is piped to the gnuplot plot command. The –f option in the cut utility selects the ninth field from the input line and the –d option tells cut to use spaces, not tabs, as delimiters. The tr utility uses the –d option to delete the comma after the residual value.

Note that the script shown in Figure 3 will need to be modified if a highly nonorthogonal mesh is used for the simulation. In this case the pressure equation will be solved a number of times within each iteration of the SIMPLE loop to increase the accuracy of the final result. The number of times this occurs is determined by the nNonOrthogonalCorrectors keyword in the fvSolutions file. If this is set to 5, for example, then the line containing the grep command "Solving for p" needs to be changed to:

"<cat log | grep 'Solving for p' | cut -d ' ' -f9 | sed -n 'p;N;N;N;N' | tr -d ',"' title 'p' with lines,\

In this case the sed editor with the –n option allows only the last occurrence of the pressure residual to be plotted.

An alternative procedure for monitoring the progress of an OpenFOAM simulation is to use the foamLog script. This extracts data from a log file and presents it as a set of files stored in a subdirectory of the case file directory called logs. The files are presented in the two-column format of iteration number and extracted variable. More details on the use of this script can be found in the OpenFOAM User Guide [3].

```
set term png
set output "residuals.png"
set logscale y
set title "Residuals"
set ylabel 'Residual'
set xlabel 'Iteration'
set xrange [1 : 5000 ]
plot "<cat log | grep 'Solving for Ux' | cut -d ' ' -f9 | tr -d ',"' title 'Ux' with lines,\
"<cat log | grep 'Solving for Uy' | cut -d ' ' -f9 | tr -d ',"' title 'Uy' with lines,\
"<cat log | grep 'Solving for Uz' | cut -d ' ' -f9 | tr -d ',"' title 'Uz' with lines,\
"<cat log | grep 'Solving for p' | cut -d ' ' -f9 | tr -d ',"' title 'p' with lines,\
"<cat log | grep 'Solving for k' | cut -d ' ' -f9 | tr -d ',"' title 'k' with lines,\
```

Figure 3. Sample gnuplot script for plotting residuals data

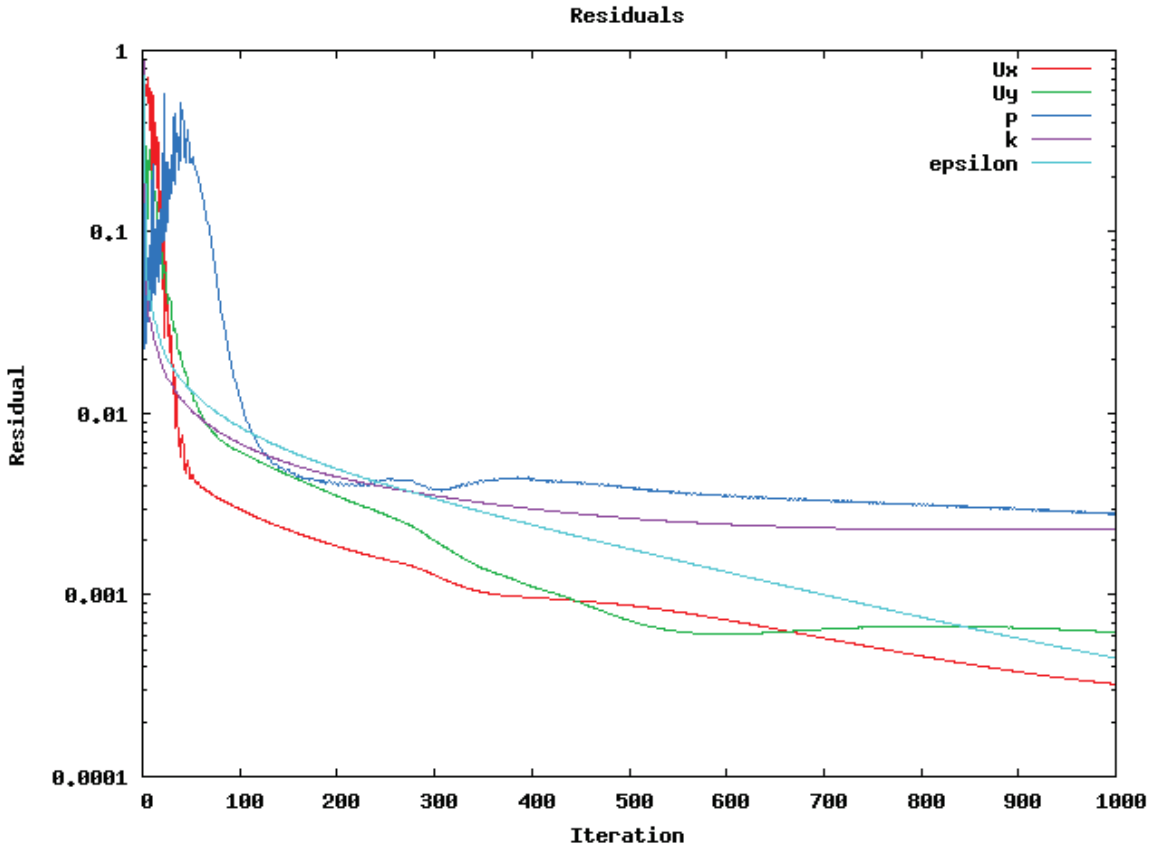

Figure 4: Residuals plot obtained by running the script shown in Figure 3

4. Flow along a Curved Ramp

In order to illustrate the use of simpleFoam we have used it to simulate the contoured ramp experiment of Song et al. [14]. In this experiment a two-dimensional boundary layer develops along a flat plate and then flows down a curved ramp, where it separates and then reattaches and redevelops on a downstream flat plate.

4.1 Curved Ramp Experimental Description

The Song et al. experiments were performed in a close-loop wind tunnel having a test section 152 mm by 711 mm by 3.0 m long. A special insert was placed into the rectangular test section to form the ramp geometry. A two-dimensional schematic of the downstream end of this insert is shown in Figure 5. The two-dimensionality of the flow was checked by measuring the streamwise mean and fluctuating velocity at several spanwise locations. The measurements showed that the flow in the central region of the test section was spanwise uniform and was not affected by the side wall boundary layers. The ramp itself is a portion of a circular arc having a radius of 127 mm and its horizontal extent, denoted to be the ramp length L, is 70 mm. Experimental data are presented in a coordinate system in which x is the freestream direction and y is the wall normal direction. The y axis is maintained vertical and does not follow the curvature of the ramp. A normalized streamwise coordinate, $x' = (x\text{-}x_0)/L$, where x_0 corresponds to the beginning of the ramp, is also used. The flow has a free stream velocity of 20.4 m/s and the working medium is air. The Reynolds number based on the step height of 21 mm is 28,600.

The boundary layer on the upstream plate has a thickness of approximately 25.3 mm as it approaches the ramp. The curvature of the ramp initially produces a favourable pressure gradient up to $x' = 0.16$, after which the effect of the expansion dominates and causes a strong adverse pressure gradient which leads to separation of the boundary layer at $x' = 0.77$. Reattachment occurs at $x' = 1.36$ and the horizontal extent of the separation bubble is 41 mm. The height of the separation bubble at the trailing edge is 4.7 mm. Experimental velocity measurements are available at several streamwise locations between $x' = -2$ and $x' = 7$.

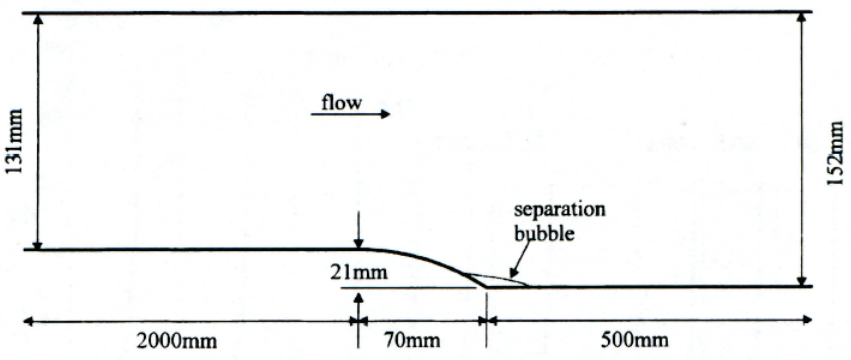

Figure 5: Two-dimensional schematic of contoured ramp experiment of Song et al.

4.2 Mesh Generation

The computational mesh for the simulation was created using the OpenFOAM blockMesh utility and the required blockMeshDict file is shown in Appendix A. Four meshes were used for the simulations and the details are summarized in Table 1 below. Mesh refinement is done by changing the number of node points in the x and y directions for the edges of the blocks on the cell and then issuing the blockMesh command again to create the new mesh. The quality of the resulting mesh can then be checked by using the checkMesh utility. This calculates the maximum aspect ratio, defined as the ratio of the longest to shortest edge length of a cell, the maximum nonorthogonality, which is defined as the angle between the face normal to the cell and the line joining the cell centre to the neighbouring cell centre, and the face skewness, which is defined as the angle between the face centroid and the face integration point. Large values for the maximum aspect ratio are generally to be avoided, although in practice OpenFOAM seems to be remarkably tolerant of values as high as several thousand. The maximum nonorthogonality value should be kept below 70 degrees. The solution accuracy is reduced if the value is between 70 and 90 degrees, although this can be compensated for by increasing the number of nonOrthogonalCorrectors in the fvSolution file. If the maximum nonorthogonality is over 90 degrees the code will not run. Highly skewed face cells reduce the order of face integration but do not cause instability problems. For meshes with highly skewed cells special gradient calculation schemes should be used, such as the least squares gradient with limiters.

Table 1: Mesh details for the four meshes used for the ramp case

	Coarse	**Fine**	**Finer**	**Finest**
Total number of cells	**33,600**	**168,000**	**432,000**	**756,000**
Height of first cell on ramp	**2.3 mm**	**0.365 mm**	**0.183 mm**	**0.102 mm**
Maximum aspect ratio	**12.08**	**27.39**	**54.70**	**97.66**
Maximum non-orthogonality	**32.62**	**33.04**	**33.15**	**33.26**
Average non-orthogonality	**3.34**	**3.50**	**3.55**	**4.79**
Maximum skewness	**1.176**	**0.7724**	**0.759**	**0.713**

4.3 Boundary Conditions

Two-dimensional simulations were run for the standard k-ε, k-ε RNG, k-ε Realizable and k-ω SST turbulence models on the coarse mesh and the k-ω SST model was also run on the more resolved meshes. For the standard k-ε model simulations were run using both OpenFOAM versions 1.5 and 1.6 and the results were found to be identical. The appropriate wall boundary condition for k, ε and p in version 1.5 is the zero gradient condition, while for U the velocity is specified to be zero on the wall. The exact form of the boundary conditions for k, ε, U and p are shown in Appendix A. The remaining simulations were run using OpenFOAM version 1.6 and this necessitated slight changes to some of the boundary conditions. As mentioned in Section 2, in OpenFOAM version 1.5 the specification of a wall patch name in the boundaries

file automatically assigns wall functions to the wall boundary patches whereas in version 1.6 this is no longer the case; the 0 directory requires the addition of a nut file with the specification of a nutWallFunction on all boundaries. The epsilon file then requires an epsilonWallFunction on wall boundaries, the k file requires a kqRwallFunction and the omega file requires an omegaWallFunction. These boundary conditions are appropriate when a high Reynolds number turbulence model is being run on a coarse mesh and the y^+ value is in the range $30 < y^+ < 200$. The k-ω SST turbulence model can also be used on highly resolved meshes without the use of wall functions however, in which the case the above boundary conditions need to be changed so that nut and k have a fixed value of zero on the wall. The boundary condition for omega remains the same, and the specification of an omegaWallFunction on a fine mesh will set the boundary condition for omega to that recommended by Menter [15]. A summary of these boundary conditions for OpenFOAM version 1.6 is given in Table 2.

Table 2: Wall function boundary conditions for OpenFOAM-1.6

Field	Coarse near wall mesh.	Resolved near wall mesh.
nut	**nutWallFunction**	**fixedValue 0**
k,q,R	**kqRWallFunction**	**fixedValue 0**
epsilon	**epsilonWallFunction**	**zeroGradient**
omega	**omegaWallFunction**	**omegaWallFunction**
nuTilda	**zeroGradient**	**fixedValue 0**

4.4 Solver Settings

The discretization schemes and the solvers used for each of the resulting matrix equations for the standard k-ε model simulation for OpenFOAM version 1.5 are shown in the fvSchemes and fvSolution files in Appendix A. Experimentation and previous user experience [2] have shown that the simulated results are insensitive to the discretization scheme used for the convective divergence term in the turbulence equations (for example k, ε or ω). For these equations, use of the upwind discretization scheme ensures stability and does not degrade solution accuracy. This is not the case for the convective divergence term in the momentum equation however, where different discretization schemes can have a significant effect on the results. In particular, the use of upwind discretization in the momentum equation produces significant errors. This is illustrated in detail in Section 5.

The linearUpwind scheme for the convective divergence term in the momentum equation provides better accuracy and corresponds to the second order upwind scheme in Fluent. It uses the gradient term to extend the flux between neighbouring cells to second order accuracy, hence when using this scheme the user needs to specify the discretization scheme used for the gradient term. For example div(phi,U) can be set to: Gauss linearUpwind Gauss linear. It is also possible to limit the gradient used in the linearUpwind scheme, in which case div(phi,U) is specified as: Gauss linearUpwind cellLimited Gauss linear 1. The results also appear to be relatively insensitive to the discretization scheme used for the Laplacian term in

each of the equations and so a central differencing scheme is usually chosen for these terms [16].

The initial values for k, ε and ω were calculated from standard expressions. The turbulence intensity in the wind tunnel was assumed to be 3% and the initial value for k was calculated from $k = 1.5 \bullet (I\langle u \rangle)^2$, where I is the turbulence intensity. The initial value for ε was based on the assumption that the ratio of turbulent to laminar viscosity was five, hence ε was calculated from the expression $\varepsilon = c_\mu k^2 / 5\nu$, and ω was calculated from $\omega = \varepsilon / k$. This resulted in the following numerical values: k = 0.562 m^2/s^2, ε = 378 m^2/s^3 and ω = 674 s^{-1}.

4.5 Running simulations using MPI

The simulations on the more resolved meshes were run in parallel mode over multiple processors (where one processor is defined as one computing core). This was done by first running the decomposePar utility which specifies the way in which the mesh will be divided amongst the number of processors. The user has a choice between four different methods of domain decomposition, as follows:

Simple: A geometric decomposition in which the domain is automatically split by direction.

Hierarchical: This is the same as Simple, except that the user specifies the order in which the directional split is done.

Metis: This requires no geometric input from the user and attempts to minimize the number of processor boundaries.

Manual: In this decomposition scheme the user directly specifies the allocation of each cell to a particular processor.

In the author's experience simple and metis are the preferred options. Which method works best for a given mesh is impossible to decide a priori and the user has to try both options. While simple may work faster on some meshes, metis is usually the safer option. The method attempts to minimise communication between processors and the user has the option of specifying a weighting for each processor if the cluster is composed of processors providing unequal performance. If the cluster consists of a number of identical processors then the list of weighting coefficients can be neglected and the user only has to specify the total number of processors in the cluster. The decomposition method and required parameter values are specified in the decomposeParDict file which is located in the system folder. The decomposePar utility is then executed by typing the command decomposePar in the case directory. This will create a set of subdirectories in the case directory with names processorN, where N = 0,1,2, ... n-1, where n is the number of cores. Each processorN subdirectory has a time directory containing the decomposed field description and a constant/polyMesh directory containing the decomposed mesh description. The simulation is then run using the command

```
mpirun –np 6 simpleFoam –parallel > log &
```

which runs the job over six processors and outputs residuals information to a log file. Upon completion of the run, and depending on the post processing software used, the data may then need to be reconstructed for post-processing. This is done by issuing the command reconstructPar from the case directory. This merges the sets of time directories from each of the processor directories into a single set of time directories in the case directory.

The standard k-ε, k-ε RNG and k-ω SST simulations all converged well on the coarse mesh, with the pressure residual decreasing to ~ 10^{-4} and the k, ε, ω and velocity variables decreasing to ~ 10^{-6}. The k-ε Realizable simulation converged very poorly, the pressure residual decreased to ~ $4x10^{-3}$ while the k, ε and velocity variables only decreased to ~ 10^{-4}. The k-ω SST simulation on the finer meshes converged best of all, with the pressure residual decreasing to ~ 10^{-5} and the k, ω and velocity variables decreasing to ~ 10^{-8}. Figure 6 shows the residuals plot from the fine mesh simulation. The sudden jump in the pressure residual at around iteration 7,000 is due to a change in the solver setting.

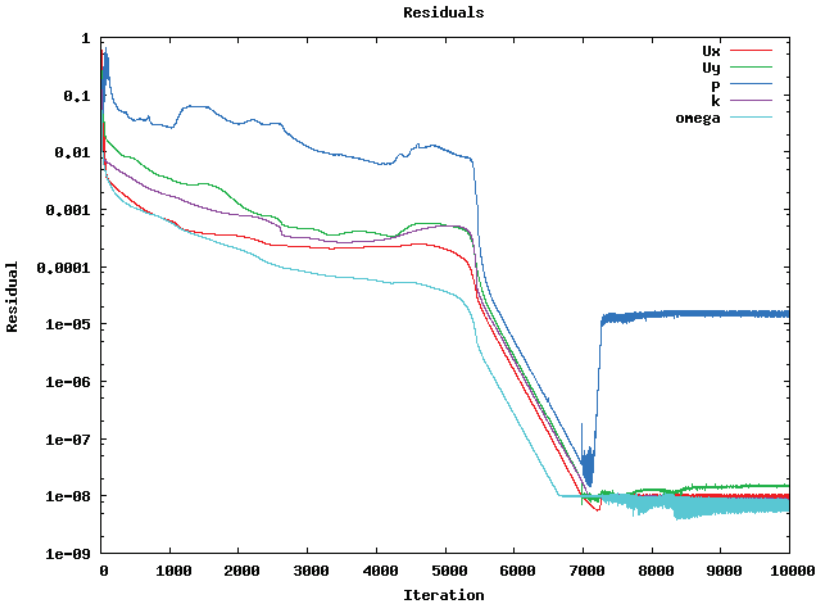

Figure 6: Residuals plot for k-ω SST simulation on the fine mesh

4.6 Simulation Results

Velocity profiles in the wall normal direction at streamwise locations x' = -2, 0, 0.77, 1 and 4 are shown in Figures 7 through 12. The simulation data for these plots was obtained by using the OpenFOAM sample utility. The line over which the data is to be sampled is specified in the sampleDict file in the system directory. The sampleDict file used for the present plots is shown in Appendix A. The results shown in Figures 7 through 12 show the expected trend. Upstream from the start of the ramp a typical two-dimensional flat plate turbulent boundary layer develops and this is reasonably well simulated by all of the turbulence models, other than the k-ε Realizable model, which failed to converge adequately. The k-ω SST model on the

fine mesh shows excellent agreement with the experimental data just at the start of the ramp at x/L = 0.

Experimentally the flow separates at $x' = 0.77$ and reattaches at $x' = 1.36$. The standard k-ε model is not expected to be able to capture this separation point, as shown in Figure 9, although neither do the k-ε RNG nor the k-ω SST models. However at $x' = 1$, shown in Figure 10, which is approximately in the middle of the separation bubble, the k-ω SST model on the fine mesh shows a region of negative velocity, indicating that it has at least qualitatively simulated this separation region. Figure 11 shows the profiles at $x' = 4$, where the flow has essentially recovered to the standard two-dimensional flat plate boundary layer again and all the models (except for the poorly converged k-ε Realizable model) show reasonable agreement with the experimental points.

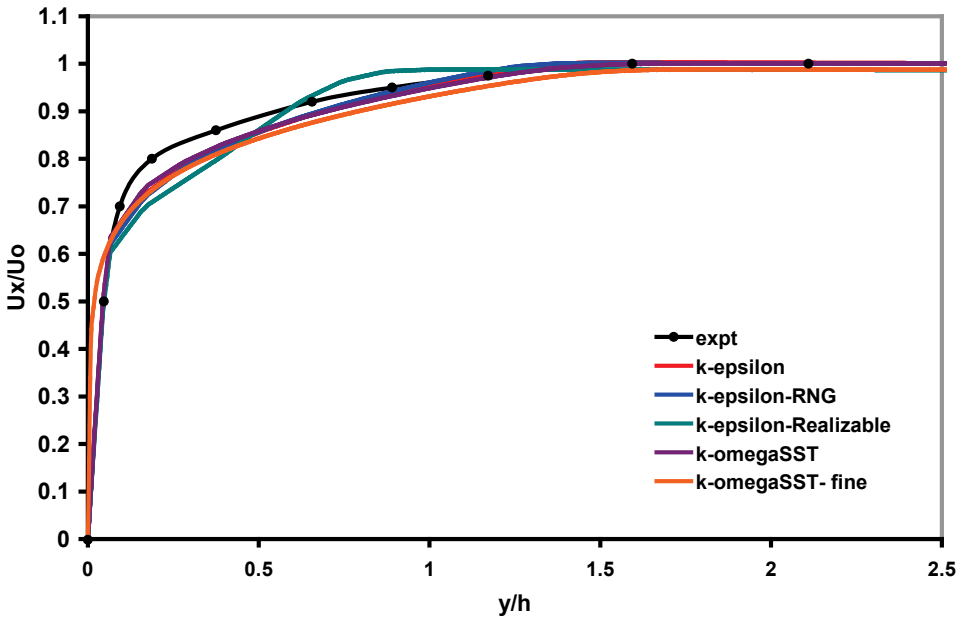

Figure 7: U_x/U_0 versus y/h at $x' = -2$

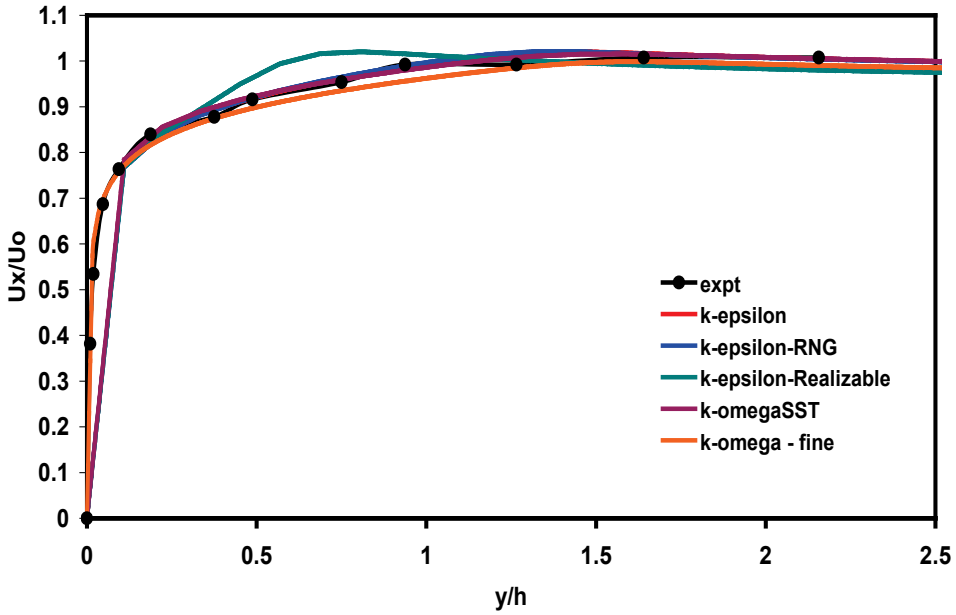

Figure 8: U_x/U_0 versus y/h at x′ = 0

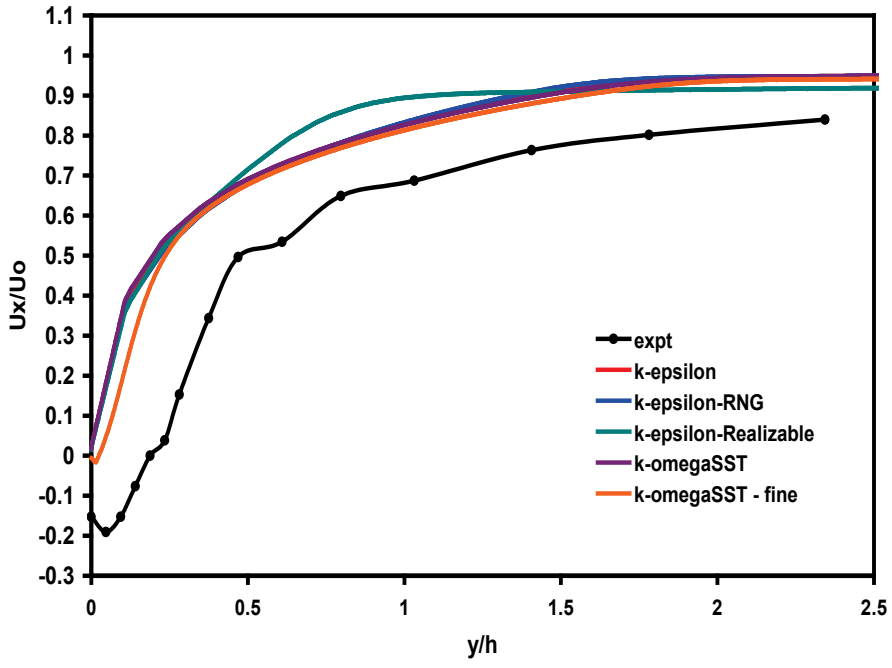

Figure 9: U_x/U_0 versus y/h at x′ = 0.77

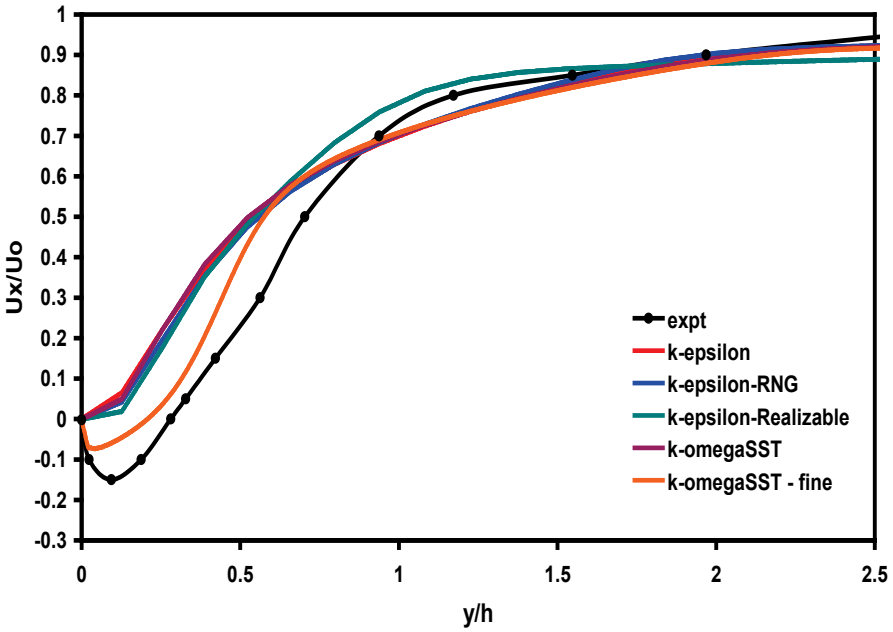

Figure 10: U_x/U_0 versus y/h at x′ = 1.0

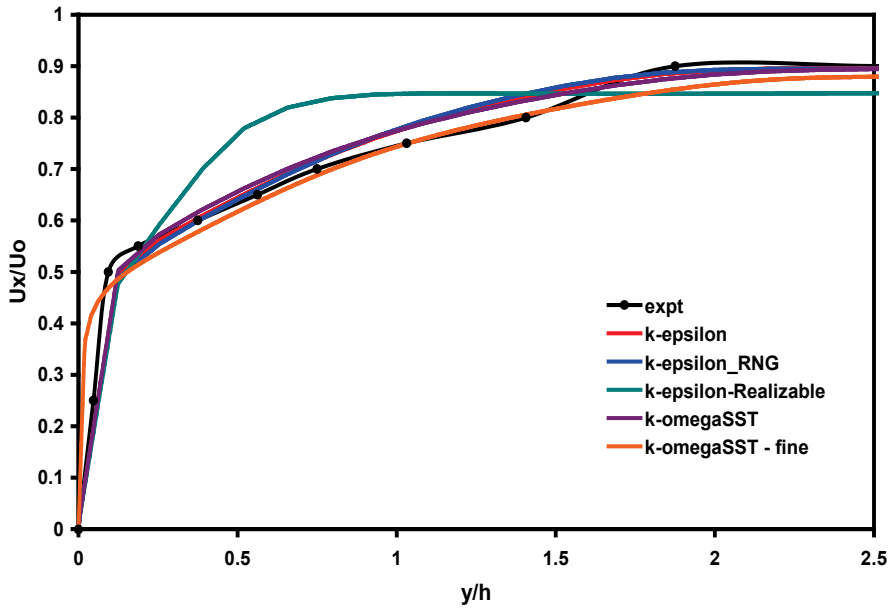

Figure 11: U_x/U_0 versus y/h at x′ = 4

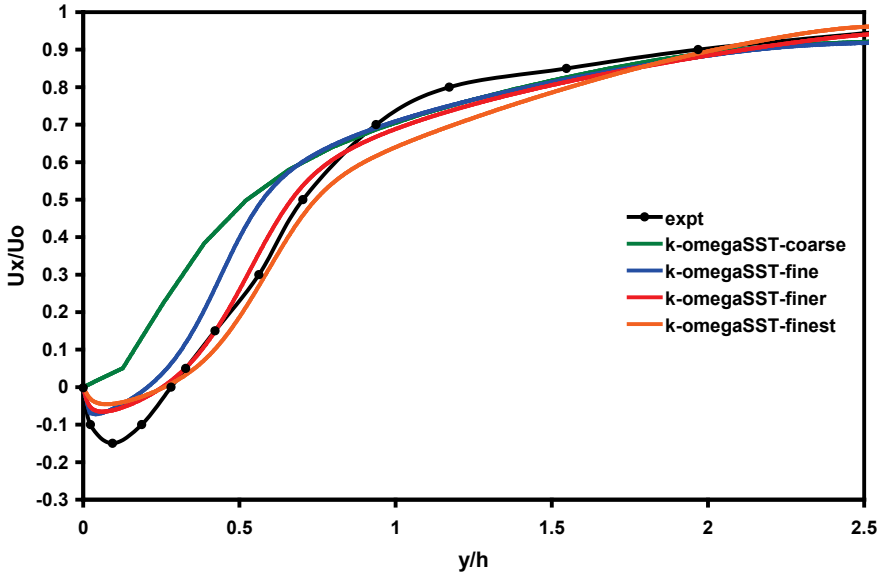

Figure 12: Ux/U0 versus y/h at x′ = 1, k-ω SST model

The effect of increased resolution using the *k-ω* SST model without a wall function is most clearly seen in Figure 12, which shows the velocity profile in the wall normal direction at x' = 0.77, which is in the middle of the separation bubble. The *k-ω* SST model on the coarse mesh uses a wall function and is unable to capture the region of flow reversal, whereas when used without a wall function on the three finer meshes the model clearly shows a region of negative velocity. The results using the more resolved meshes show considerably better agreement with the experimental results in the region around $y/h = 0.5$.

5. Flow around a Conventional Submarine Hull

The second case used to demonstrate the simpleFoam solver is the simulation of the flow around the bare hull of a generic conventional scale model submarine. The design is based on the reports by Joubert [17, 18] and the geometry of the fully appended model without propeller is shown in Figure 13. A 1.35 m long scale model of this design has been constructed and tested in the Air Vehicles Division low speed wind tunnel. Experimental data is available for force and moment measurements over a range of pitch and yaw angles as well as pressure and skin friction coefficients along the hull [19, 20].

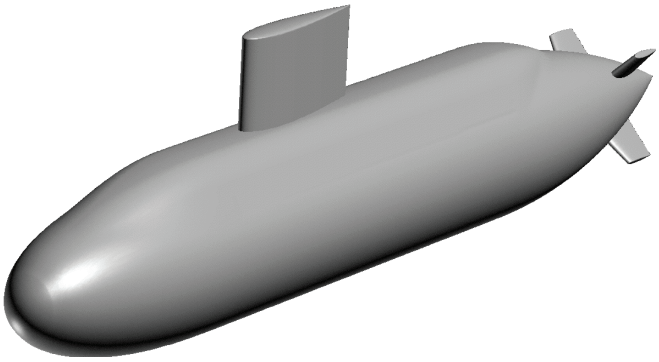

Figure 13. The generic conventional DSTO submarine hull form, bow view

5.1 Mesh Generation

A structured hexahedral mesh of the generic conventional submarine hull was constructed using ICEM meshing software and a one quarter geometry model of the hull. Cylindrical geometry was adopted for the far field boundary along the axis of the hull and a hemi-spherical cap was used for the region forward of the bow. The mesh contains 944,984 hexahedral cells and was designed to be used with turbulence models requiring the use of a wall function. The average y^+ value along the cylindrical mid-section of the hull is 100. This increases to 140 over the nose and part of the tail and drops to around 30 at the very end of the tail. Figure 14 shows the domain used for the simulation and Figure 15 provides a detailed view of the mesh in the vicinity of the hull. Each of these meshes was read in to the Fluent code and then exported in ASCII format as a .msh file and then placed in an appropriate OpenFOAM case directory. The meshes were then converted to OpenFOAM format by running the fluentMeshToFoam utility as described in Section 2.

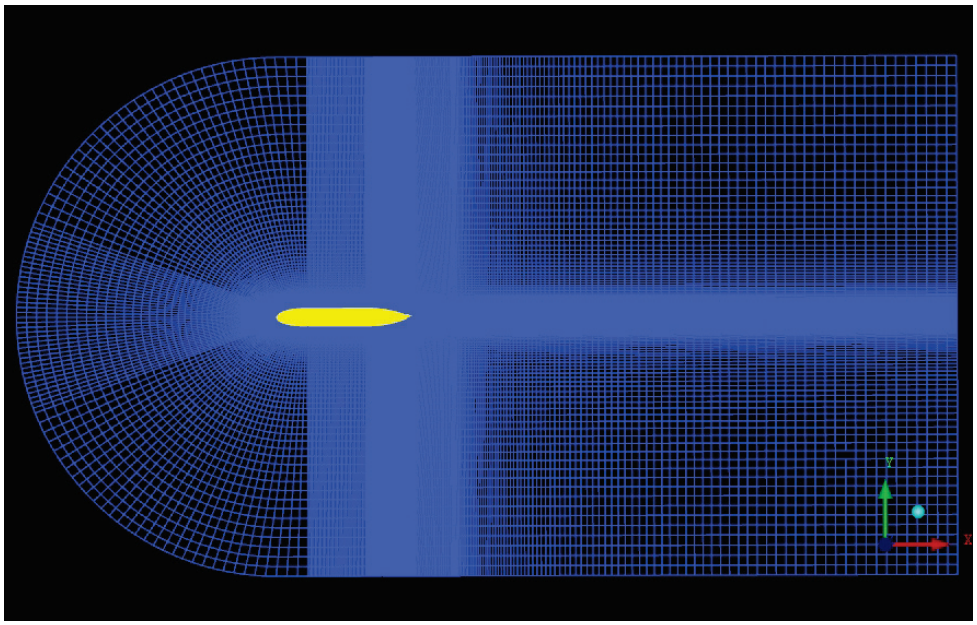

Figure 14. Domain used for the simulation

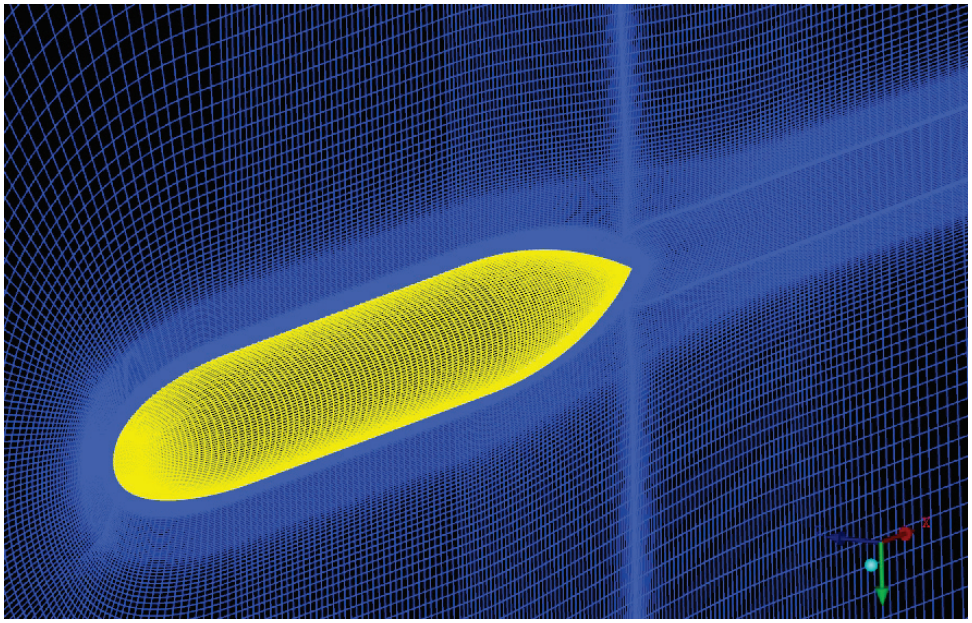

Figure 15. Detailed view of mesh in the vicinity of the hull.

5.2 Boundary Conditions

The standard combination of Dirichlet and Neumann boundary conditions appropriate to incompressible flow around a bluff body were used for all simulations. A velocity inlet boundary condition was applied at the inflow and a pressure outlet boundary condition was applied at the outflow. A no slip boundary condition was applied to the hull surface and the remaining surfaces, consisting of the side wall, symmetry wall and far field, were set to symmetry. Simulations were performed at a flow speed of 60 m/s and the assumed inflow intensity level was 3%. Initial values for k and ε were calculated following the same procedure as described for the curved ramp case, resulting in the values $k = 4.86\ m^2/s^2$ and $\varepsilon = 2.9 \times 10^4\ m^2/s^3$. These settings are summarized in Table 3, where **n** denotes a unit vector normal to the geometry surface.

Table 3: Boundary conditions for OpenFOAM-1.5 simulations on bare hull

Geometry	Boundary condition
Inlet	$\mathbf{v}_0 = 60\ m/s,\ \nabla p \cdot \mathbf{n} = 0,\ k = 4.86\ m^2/s^2,\ \varepsilon = 29.1 \times 10^3\ m^2/s^3$
Outlet	$\nabla \mathbf{v} \cdot \mathbf{n} = 0,\ p = 0,\ \nabla k \cdot \mathbf{n} = 0,\ \nabla \varepsilon \cdot \mathbf{n} = 0$
Hull	$\mathbf{v} = 0,\ \nabla p \cdot \mathbf{n} = 0,\ \nabla k \cdot \mathbf{n} = 0,\ \nabla \varepsilon \cdot \mathbf{n} = 0$
Side wall	Symmetry Plane
Symmetry wall	Symmetry Plane
Far field	Symmetry Plane

5.3 Calculation of Forces and Moments

Calls to function objects were used in order to calculate the forces and moments on the hull as the simulation proceeded. These function objects are independent pieces of coding which have been compiled into libraries and which can be called from the controlDict file to monitor the progress of the run as the simulation proceeds. They are called at the end of each iteration and do not change the structure of the main code. An example of the statements required in the controlDict file to output the drag, lift and moment coefficients by calling the "forceCoeffs" function object is shown in Figure 16. Similar statements can be used to call the "forces" function object to calculate the x, y and z components of both the pressure force and viscous force on the hull.

```
functions
(
forceCoeffs
{
type forceCoeffs;
functionObjectLibs ("libforces.so");
outputControl timeStep;
outputInterval 1;
log true;
patches ( HULL ) ;
rhoNamerhoInf;
rhoInf 1.225;
CofR (0 0 0);
liftDir (0 1 0);
dragDir (1 0 0);
pitchAxis (0.25 0 0);
magUInf 60.0;
lRef 1.35;
Aref 0.455625;
}
);
```

Figure 16. Part of the controlDict file in the system subdirectory showing the statements required for force and moment reporting after each iteration.

5.4 Solver Settings

In the fvSchemes file the gradSchemes were set to Gauss linear, the laplacianSchemes to Gauss linear corrected and the div(phi,k) and div(phi, epsilon) terms set to Gauss upwind. In the fvSolution file the pressure solver was PCG using the DIC preconditioner, a tolerance of 1.0e-08 and a relTolerance of 0.0. For U, k and epsilon the PBiCG solver was used with the DILU preconditioner. The tolerance was typically set at 1.0e-06 with relTolerance = 0.0. nNonOrthogonalCorrectors was set to zero. All runs showed excellent convergence with the residuals decreasing by 5 or 6 orders of magnitude, except for the two limitedLinear runs, where the pressure residual only decreased by one to two orders of magnitude.

As mentioned in the previous section, experience has shown that the simulated results are insensitive to the discretization scheme used for the convective divergence term in the turbulence equations, but this is not the case for the convective divergence term in the momentum equation. In order to illustrate the sensitivity of the solutions to the choice of discretization scheme used in OpenFOAM a number of such schemes were tried and these are listed in Table 4, along with the results obtained for the viscous and pressure forces obtained using these schemes.

The upwind scheme is very effective at suppressing spurious oscillations at the leading and trailing edges of sharp wave forms by taking the flow direction into account when

determining the value of a flow variable at a cell face. In this scheme, the value at the cell face is taken to be equal to the value of the variable at the upstream node. Whilst this provides good stability properties, the scheme has only first order accuracy and is very diffusive. The linearUpwind scheme was described in the previous section and corresponds to Fluent's second order upwind scheme. The blended 0.9 scheme is a blend of 90% central differencing (a linear scheme) and 10% upwind.

A number of non-linear flux limited schemes are also available in OpenFOAM. The limitedLinear scheme has a flux limiter of the form max(min(2r/k,1),0) where k is the parameter specified after the name limitedLinear and r is the ratio of the consecutive gradients. The Gamma scheme is a smooth and bounded blend between a second order central differencing scheme and a first order upwind scheme. Central differencing is used wherever it satisfies the boundedness requirements and upwind is used wherever central differencing is unbounded [13]. For the convection divergence terms in the momentum equations some of these schemes have improved formulations which take into account the direction of the flow field. These are designated as "V" schemes, for example GammaV, linearLimitedV.

5.5 Simulation Results

5.5.1 Effect of discretization scheme used for convective term

The results shown here have been calculated using the standard *k*-ε turbulence model on the coarse mesh using standard wall functions. The simulated drag coefficient and the total pressure and viscous force exerted on the hull are shown in Table 4. The linearUpwind scheme in OpenFOAM is equivalent to the 2nd order upwind scheme in Fluent. For the three slightly different implementations of this scheme tested here C_d varies between 0.001316 and 0.001322. The average value is 0.001319. For the two slightly different Gamma schemes C_d varies between 0.001317 and 0.001323, with an average vale of 0.001320. The upwind scheme is expected to be the least accurate of these discretization schemes and the calculated value of 0.001652 is 25.2%higher than the average value obtained from the linearUpwind and Gamma schemes. The accuracy of the blended scheme will depend on the amount of blending between the central differencing and upwind schemes. Blended 0.9 contains only 10% of the upwind scheme, but the calculated drag coefficient is 17.8% higher than the average value from the linearUpwind and Gamma schemes. The limitedLinear simulations showed very poor convergence of the pressure residual and should be disregarded.

It should be noted that there are a large number of other discretization schemes available in OpenFOAM and that there are few guidelines available in the literature to indicate which of these are the most accurate. The schemes most commonly used by OpenFOAM users appear to be linearUpwind and linearUpwind cellLimited [21, 22]. Our simulations have shown that these schemes provide good convergence and give results which are less than 0.1% different to the results obtained from the OpenFOAM Gamma scheme.

Table 4. Effect of discretization scheme used for the convective divergence term in the momentum equation. OpenFOAM simulation on bare hull

Discretization scheme - OpenFOAM	C_d	$F_{pressure}$ (N)	F_{visc} (N)
upwind	**0.001652**	**0.4339**	**1.2259**
linearUpwind Gauss linear	**0.001316**	**0.1020**	**1.2206**
linearUpwindV cellLimited Gauss linear 1.0	**0.001322**	**0.1105**	**1.2172**
linearUpwindV cellLimited Gauss linear 0.75	**0.001320**	**0.1050**	**1.2204**
blended 0.9	**0.001343**	**0.1313**	**1.2183**
limitedLinear 0.9	**0.001392**	**0.1425**	**1.2562**
limitedLinear 0.5	**0.001339**	**0.1267**	**1.2187**
Gamma 0.5	**0.001317**	**0.1005**	**1.2229**
Gamma 0.9	**0.001323**	**0.1010**	**1.2285**

5.5.2 Pressure Coefficient

A plot of the pressure coefficient C_p versus distance along the hull for two OpenFOAM runs is shown in Figure 17. Fieldview post processing software was used to obtain the data by first running the foamToFieldview9 utility to create the appropriate unstructured .uns file. Also shown are the experimental data obtained from the work of Quick et al. [19]. The simulated results using the linear upwind scheme provide a reasonable fit to the experimental data except at the very aft region of the hull. The results using the upwind scheme are less accurate, as expected, and show a maximum deviation from those using the linear upwind scheme of approximately 20%, which is similar to the difference in the simulated values of the drag coefficient using the two different schemes.

The experimental C_p data at the aft region of the hull suggests that flow separation may be occurring in this region. The simulated pressure field does not show the same behaviour however. The contour plot of the mean axial velocity shown in Figure 18 also does not show any evidence of flow separation. A possible explanation for the discrepancy between the experimental and simulated C_p data in this region is the perturbation to the flow caused by the experimental support strut, which is not included in the current simulations. The perturbation to the flow caused by the strut may only become significant on the upper surface of the model in the region very close to the tail, where the model is very thin. Further investigations of the flow field in this region are planned using a less intrusive support structure and flow visualization techniques.

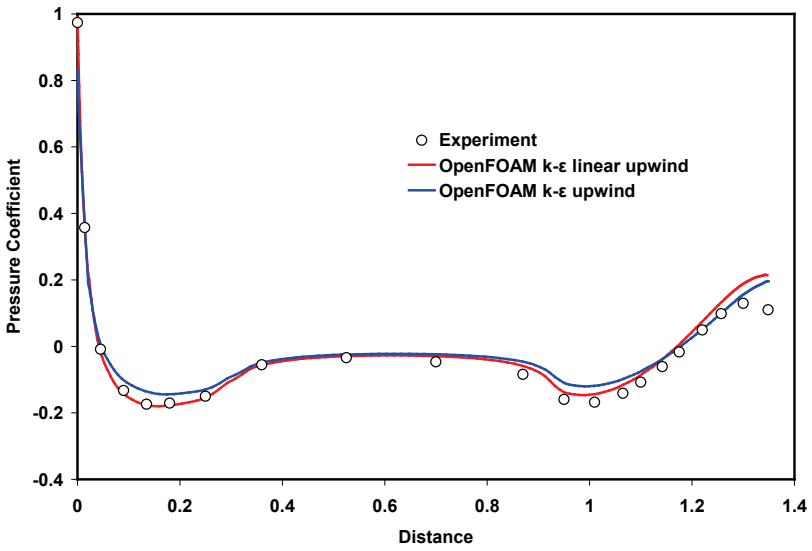

Figure 17. Pressure coefficient Cp versus distance along the hull

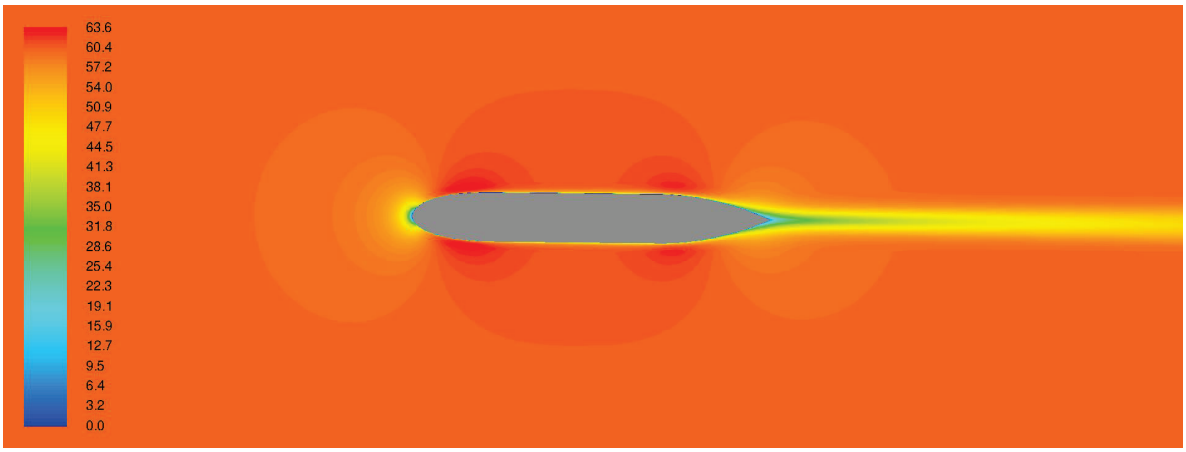

Figure 18.Contour plot of mean axial velocity.

5.5.3 Skin Friction Coefficient

A plot of the skin friction coefficient C_f versus distance along the hull for the same two OpenFOAM runs is shown in Figure 19. The experimental data has again been taken from the results of Jones et al. [20]. The C_f values for the OpenFOAM simulations were obtained by first running the OpenFOAM wallShearStress utility to calculate the components of the wall shear stress. Note that because the steady state solvers in OpenFOAM work with the kinematic viscosity rather than the dynamic viscosity the magnitude of the wall shear stress calculated by OpenFOAM is non-dimensionalised by dividing by $0.5U_o^2$ rather than by $0.5\rho U_o^2$, where U_o is the free stream velocity. The difference between the OpenFOAM results from the

upwind and linearUpwind simulations is very minor in this case. The agreement between the simulated and experimental results for C_f is less impressive than that for C_p. Although the shapes of the simulated and experimental curves show very similar trends, there is an average difference between the two of approximately 16%. The absence of simulation results near the nose of the model is due to complications arising from the use of a ¼ model geometry and symmetry boundary conditions which lead to incorrect wall shear stress values in the cells close to the symmetry planes.

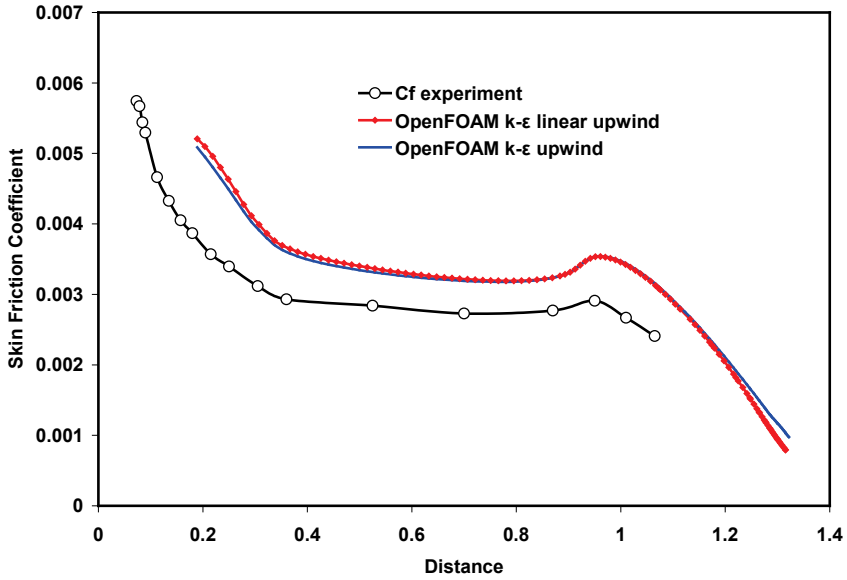

Figure 19. Skin Friction Coefficient C_f versus distance along the hull

5.6 Comparison with Fluent Code

Since the majority of the flow simulations performed in the past by the MPD Hydrodynamics Group have used the commercial code Fluent it is of interest to compare the simulated results produced by the two codes. In this section we report results obtained from a number of simulations which were run using the Fluent code using the same mesh and identical input and boundary conditions. All simulations used the standard k-ε turbulence model. The results obtained for C_d and the pressure and viscous forces are shown in Table 5.

5.6.1 Effect of discretization scheme used for convective term

Our experience with Fluent over the last few years has shown that the 2nd order upwind discretization scheme usually provides best agreement with experiment. The simulated value for the drag coefficient using this scheme is 0.001385. Both the QUICK and MUSCL schemes give the same value, Cd = 0.001376. The difference between 2nd order upwind result and that calculated by the QUICK and MUSCL schemes is less than 1%. The upwind result is 23% higher than the 2nd order upwind, QUICK and MUSCL schemes and is only 2% different from the OpenFOAM upwind result. The Power Law scheme is expected to be more accurate than

the upwind scheme and yet the value for C_d calculated using this scheme, C_d = 0.001701, is higher than the value calculated using the upwind scheme. Versteeg and Malalasekera [23] have noted however that this scheme can revert to the upwind scheme under certain flow conditions. The average value for C_d from the 2nd order upwind, QUICK and MUSCL schemes is 0.001381. This should be compared with the average OpenFOAM result of C_d = 0.001320. The difference is approximately 4.9% and could easily be accounted for by slight differences in the implementation of the turbulence models between Fluent and OpenFOAM or different solver tolerances.

Table 5. Effect of discretization scheme used for the convective divergence term in the momentum equation. Fluent simulation on bare hull.

Discretization scheme – Fluent	**C_d**	**$F_{pressure}$ (N)**	**$F_{viscous}$ (N)**
Fluent 2nd order upwind	**0.001385**	**0.1369**	**1.2549**
Fluent QUICK	**0.001376**	**0.1311**	**1.2517**
Fluent MUSCL	**0.001376**	**0.1321**	**1.2502**
Fluent Power Law	**0.001701**	**0.4588**	**1.2499**
Fluent upwind	**0.001687**	**0.4459**	**1.2490**

The simulated value for the drag coefficient calculated from both OpenFOAM and Fluent can also be compared with the simulations of Quick et al. [19]. They performed simulations on the bare hull at zero degree angle of incidence using Fluent and an unstructured tetrahedral mesh using prism layers to capture the boundary layer. Several different turbulence models were used. For the k-ε Realizable model they found C_d = 0.00142, for the k-ω model C_d = 0.00147, while for the k-ω SST model C_d = 0.00144. These values are on average 7% higher than the values calculated using Fluent on the current mesh and 9% higher than the value calculated using OpenFOAM. The experimental value is 0.00179 [19]. In the experimental runs however the model is supported on a pylon and there is also an angle arm attached to the model. Both of these features disturb the flow around the model to some extent. The presence of the pylon and the angle arm has not been taken into account in the CFD results quoted above. Snowden [19] however has investigated the effect of these perturbations to the flow field using CFD simulations. Using an unstructured mesh which modelled the bare hull, the pylon and the angle arm he has performed a k-ω simulation and found C_d = 0.00156, which is approximately 6% higher than the drag coefficient calculated using the k-ω model on the bare hull mesh and 13% lower than the experimental value. If we increase the value of the drag coefficient calculated on the mesh used here by 6% to account for the presence of the pylon and the angle arm then the OpenFOAM result becomes Cd = 0.001400 and the Fluent result is Cd = 0.00147, which are respectively 22% and 18% lower than the experimental value. These values will change however as the mesh used here is refined. The current mesh is a relatively coarse one close to the hull and no grid refinement study has been performed as the aim of this work was to provide an understanding of the simpleFOAM code and to illustrate its application to some generic cases. A detailed grid independence study on both the bare hull and the fully appended Joubert model using OpenFOAM will depend on future work priorities.

5.6.2 Pressure Coefficient

A plot of the pressure coefficient C_p versus distance along the hull for the two codes is shown in Figure 20. The OpenFOAM simulation used the linear upwind scheme for the discretization of the convective term in the momentum equation while the Fluent simulation used the 2nd order upwind scheme. The C_p plots from the two codes are effectively identical.

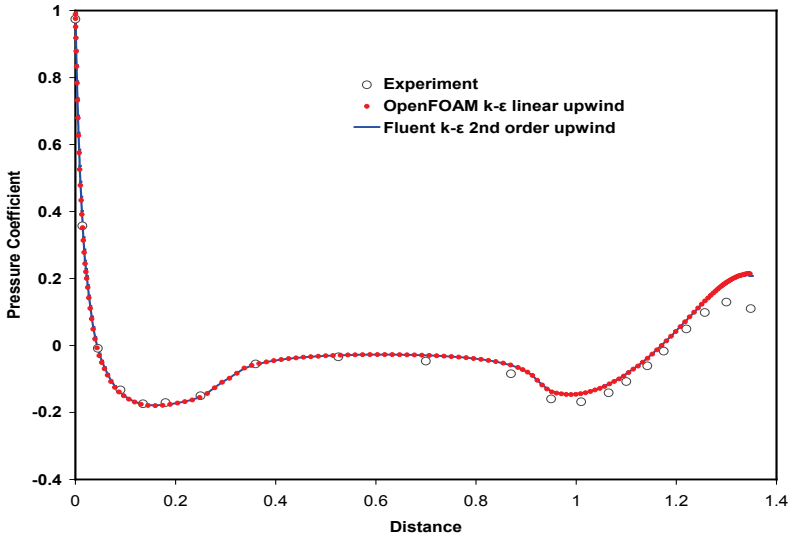

Figure 20. Pressure coefficient along the hull from OpenFOAM and Fluent simulations

5.6.3 Skin Friction Coefficient

A plot of the skin friction coefficient C_f versus distance along the hull for the two codes from the same simulations as above is shown in Figure 21. The two curves are very similar but not identical, as was the case for the pressure coefficient. Agreement is almost perfect along the cylindrical midsection of the hull and the differences over the nose and tail are no more than 10%. One possible reason for these discrepancies could be slight differences in the methods used to implement the wall function in the two codes.

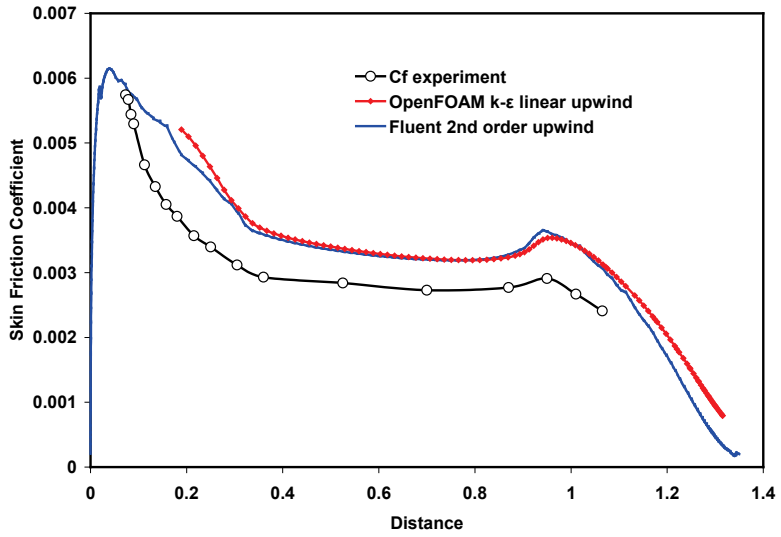

Figure 21. Skin Friction Coefficient along the hull from OpenFOAM and Fluent simulations

5.7 Comparison with other OpenFOAM Simulations

Johansson [24] has also used simpleFoam to simulate flow around the bare hull of the Joubert model submarine. Advantage was taken of the axisymmetric nature of the hull to create a 2D wedge type mesh with a 5º angle between the front and back of the mesh. The mesh was created using the OpenFOAM blockMesh utility and was built in sets of three dimensional hexahedral blocks. It contained 80,000 cells and the cells closest to the body were designed to give a y^+ value close to 40. A number of different RANS turbulence models were used, including the standard *k*-ε, *k*-ε RNG, *k*-ε Realizable, Lien cubic *k*-ε, Non-linear Shih *k*-ε, *k*-ω, and *k*-ω SST. Simulation results were compared with experimental values for C_p, C_f and axial velocity plots along the cross flow direction for various axial stations along the stern. The conclusion drawn was that the standard k-ε, k-ε RNG, k-ω and k-ω SST turbulence models gave best agreement with experimental results. For the standard *k*-ε model the calculated pressure coefficient values were virtually identical to those shown in Figure 17 calculated using the linear upwind scheme. The skin friction coefficient plot also showed exactly the same trend as seen in Figure 18, with the simulated values consistently following the shape of the experimental plot but remaining about 10% too high, rather than the 16% difference shown in Figure 18. Although Johansson's mesh has fewer cells, it has approximately twice the resolution close to the hull surface and this may explain the difference in the simulated results. Values for the drag coefficient were not tabulated and so cannot be compared here.

Anderson et al. [25] used an OpenFOAM Large Eddy Simulation (LES) code with several different subgrid scale turbulence models and a much finer mesh (17 million cells) to simulate flow around the Joubert bare hull model. Their simulated C_p and C_f values showed similar levels of agreement with the experimental values as those found here but the calculated drag coefficient for each model was approximately 10% higher than the values calculated using

either simpleFoam or Fluent. The improved agreement with the experimental value is most likely due to their enhanced mesh resolution.

6. Conclusion

The OpenFOAM software suite and the simpleFoam solver have been shown to be comparable to commercial CFD software packages in their ability to perform RANS simulations relevant to submarine flows. Some of the advantages of the software are its open source nature and ability to run in parallel over large processor arrays. One disadvantage however is the lack of documentation describing effective use of the various codes. This report has addressed some of the problems encountered by new OpenFOAM users by providing an introduction to the basic file structure used by all OpenFOAM solvers as well as a fairly detailed description of the OpenFOAM RANS solver simpleFoam. Several methods used to generate meshes in OpenFOAM format have been described and illustrated. Useful utilities for monitoring the progress of a simulation during run time and for post processing simulation output have also been described and illustrated. The simpleFoam code has been applied to two generic flows; two-dimensional flow along a smoothly contoured ramp and three-dimensional flow around a generic conventional submarine hull. The two-dimensional simulations were performed on four successively finer meshes to illustrate the use of the k-ω SST turbulence model both with and without the use of standard wall functions. The three-dimensional simulations over the submarine hull were performed on a single mesh as the objectives were to investigate the effect on the simulated results of the different convective discretization schemes used in the momentum equation, as well as performing a comparison with the commercial code Fluent. Despite the relatively coarse mesh used, the simulated results for the drag coefficient and the pressure and skin friction plots along the meridian line showed reasonable agreement with the experimental results. The results from the simpleFoam code were generally in very good agreement with those from the Fluent code, with the maximum discrepancy occurring in the value of the drag coefficient, where a difference of 4.9% was observed.

7. Acknowledgements

We thank Dr. Christer Fureby and Mr. Carl Troëng from FOI for explaining many features of the OpenFOAM software during a three day course given at DSTO Fishermans Bend in April 2011. We are grateful to Mr. Brendon Anderson, Head of the Hydrodynamics Group, for his continued support of this work. We thank Dr. Darrin Stephens from Applied CCM for assistance in understanding some aspects of OpenFOAM coding.

8. References

1. Weller, H.G., Tabor, G., Jasak, H. and Fureby, C., "A tensorial approach to computational continuum mechanics using object-oriented techniques", Computers in Physics, Vol 12, No. 6, 620 – 631, Nov/Dec 1998.
2. Shan, H., Delaney, K., Kim, S-E., Rhee, B, Gorski, J and Ebert, M., "Guide to NavyFOAM V1.0", NSWCCD-50-TR-2011/025, April 2011.
3. OpenFOAM, The Open Source CFD Toolbox, User Guide, Version 1.5, 9th July, 2008.
4. Tapia, X. P., "Modelling of wind flow over complex terrain using OpenFOAM", Master's Thesis in Energy Systems, University of Gävle, Vattenfall, June 2009.
5. Järpner, C., "CFD with Open Source Software: projection of a mesh on a .stl surface", Chalmers University of Technology, October 2011.
6. Pope, S.B., "Turbulent Flows", Cambridge University Press, 2000.
7. Furbo, E., "Evaluation of RANS turbulence models for flow problems with significant impact of boundary layers", Masters Thesis, Uppsala University, December 2010.
8. Patankar, S.V. and Spalding, D.B., "A Calculation Procedure for Heat, Mass and Momentum Transfer in Three-Dimensional Parabolic Flows", *Int. J. Heat Mass Transfer*, **15**, pp. 1787 – 1806, 1972.
9. Rhie, C.M. and Chow, W.L., "A Numerical Study of the Turbulent Flow Past an Isolated Airfoil with Trailing Edge Separation", *AIAA J.*, **21**, pp. 1525-1532, 1983.
10. Kärrholm, F.P., "Rhie-Chow interpolation in OpenFOAM", Department of Applied Mechanics, Chalmers University of Technology, Sweden, 2006.
11. Zikanov, O., "Essential Computational Fluid Dynamics", John Wiley and Sons, Inc, 2010.
12. Tu, J., Yeo, G.H. and Liu, C., "Computational Fluid Dynamics: A Practical Approach", Elsevier Inc. 2008.
13. Jasak, H., "Error Analysis and Estimation for the Finite Volume Method with Applications to Fluid Flows", Ph.D. thesis, Department of Mechanical Engineering, Imperial College of Science, Technology and Medicine, University of London, June 1996.
14. Song, S., DeGraff, D.B. and Eaton, J.B., "Experimental Study of a Separating, Reattaching, and Redeveloping Flow over a Smoothly Contoured Ramp", Int. J. of Heat and Fluid Flow, **21**, p 512 – 519, 2000.
15. Menter, F.R., "Improved Two-Equation k-ω Turbulence Models for Aerodynamic Flows, NASA Technical Memorandum 1203975, October 1992.
16. Nilsson, H., "Evaluation of OpenFOAM for CFD of turbulent flow in water turbines". *23rd IAHR Symposium*, Yokohama, October 2006.
17. Joubert P.N., "Some Aspects of Submarine Design Part 1 - Hydrodynamics", *DSTO Technical Report*, DSTO-TR-1622, 2004.
18. Joubert P.N., "Some Aspects of Submarine Design Part 2. Shape of a Submarine 2026", *DSTO Technical Report*, DSTO-TR-1920, 2006.

19. Quick, H., Widjaja, R., Anderson, B., Woodyatt, B., Snowden, A.D. and Lam, S., "Phase 1 Experimental Testing of a Generic Submarine Model in the DSTO Low Speed Wind Tunnel", DSTO-TN-1101, July 2012.

20. Jones, M.B, Erm, L.P., Valiyff, A. and Henbest, S.M., "Skin friction measurements on a model submarine", *Draft DSTO Report*, 2011 (draft currently [24/04/2013] with Research Leader)

21. "OpenFOAM Advanced Training", Course Notes, OpenCFD Ltd., April 26, 2010.

22. de Villiers, E., "Precompiled Applications and Utilities, Running Tutorials, Running in Parallel, and General Post-Processing Utilities", presented at the 6th OpenFOAM Workshop, Pennsylvania State University, PA, USA, June 2011.

23. Versteeg, H.K. and Malalasekera, W., "An Introduction to Computational Fluid Dynamics – The Finite Volume Method", Longman Group Ltd, 1995.

24. Johansson, M., "Hydrodynamic investigation of an axisymmetric streamlined body with respect to the velocity distribution in the stern region", Masters Thesis, FOI, July 2012.

25. Anderson, B., Chapuis, M., Erm, L., Fureby, C., Giacobello, M., Henbest, S., Jones, D., Jones, M., Kumar, C., Liefvandahl, M., Manovski, P., Norrison, D., Quick, H., Snowden A., Valiyff, A., Widjaja, R. and Woodyatt, B., "Experimental and Computational Investigation of a Generic Conventional Submarine Hull Form", presented at the 29th Symposium on Naval Hydrodynamics, Gothenburg, Swede, 26 – 31 August, 2012.

Appendix A: OpenFOAM case files for the ramp case

File: *controlDict*

```
/*--------------------------------------------------------------------------------*\
|  =======                 |                                                     |
|  \\      /    F ield     | OpenFOAM: The Open Source CFD Toolbox               |
|   \\    /     O peration  | Version:  1.5                                       |
|    \\  /      A nd        | Web:      http://www.OpenFOAM.org                   |
|     \\/       M anipulation |                                                   |
\*--------------------------------------------------------------------------------*/

FoamFile

{
   version     2.0;
   format      ascii;
   class       dictionary;
   object      controlDict;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
application simpleFoam;
startFrom       startTime;
startTime       0;
stopAt          endTime;
endTime         1000;
deltaT          1;
writeControl    timeStep;
writeInterval   200;
purgeWrite      0;
writeFormat     ascii;
writePrecision  6;
writeCompression uncompressed;
timeFormat      general;
timePrecision   6;
runTimeModifiable yes;
// ************************************************************************* //
```

File: *fvSchemes*

```
/*--------------------------------------------------------------------------------*\
|  =======                    |                                                    |
|  \\      /    F ield         | OpenFOAM: The Open Source CFD Toolbox  |
|   \\    /     O peration     | Version:  1.5                                      |
|    \\  /      A nd           | Web:      http://www.OpenFOAM.org                  |
|     \\/       M anipulation  |                                                    |
\*--------------------------------------------------------------------------------*/
FoamFile
{
   version     2.0;
   format      ascii;
   class       dictionary;
   object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
ddtSchemes
{
   default steadyState;
}
gradSchemes
{
   default          Gauss linear;
   grad(p)          Gauss linear;
   grad(U)          Gauss linear;
}
divSchemes
{
   default                          none;
   div(phi,U)                       Gauss linearUpwindV Gauss linear;
   div(phi,k)                       Gauss upwind;
   div(phi,epsilon)                 Gauss upwind;
   div((nuEff*dev(grad(U).T()))) Gauss linear;
}
laplacianSchemes
{
   default                    none;
   laplacian(nuEff,U)              Gauss linear corrected;
   laplacian((1|A(U)),p)           Gauss linear corrected;
   laplacian(DkEff,k)              Gauss linear corrected;
   laplacian(DepsilonEff,epsilon)  Gauss linear corrected;
}
interpolationSchemes
{
   default            linear;
   interpolate(U)  linear;
}
snGradSchemes
{
   default         corrected;
}
fluxRequired
{
   default         no;
   p;
}
// ************************************************************************* //
```

File: ***fvSolution***

```
/*--------------------------------------------------------------------------------------*\
|  ========                     |                                                      |
|  \\        /     F ield       | OpenFOAM: The Open Source CFD Toolbox  |
|   \\      /      O peration   | Version:  1.5                                        |
|    \\    /       A nd         | Web:      http://www.OpenFOAM.org                    |
|     \\/          M anipulation  |                                                    |
\*--------------------------------------------------------------------------------------*/
FoamFile
{
   version     2.0;
   format      ascii;
   class       dictionary;
   object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
solvers
{
   p PCG
   {
      preconditioner   DIC;
      tolerance        1e-08;
      relTol           0.01;
   };
   U PBiCG
   {
      preconditioner   DILU;
      tolerance        1e-06;
      relTol           0.01;
   };
   k PBiCG
   {
      preconditioner   DILU;
      tolerance        1e-06;
      relTol           0.01;
   };
   epsilon PBiCG
   {
      preconditioner   DILU;
      tolerance        1e-06;
      relTol           0.01;
   };
}
SIMPLE
{
   nNonOrthogonalCorrectors 0;
}
relaxationFactors
{
   p             0.3;
   U             0.7;
   k             0.7;
   epsilon       0.7;
 }
// ************************************************************************* //
```

File: *blockMeshDict*

```
/*--------------------------------------------------------------------------------*\
|  =======                 |                                                     |
|  \\      /    F ield     | OpenFOAM: The Open Source CFD Toolbox               |
|   \\    /     O peration  | Version:  1.5                                       |
|    \\  /      A nd        | Web:      http://www.OpenFOAM.org                   |
|     \\/       M anipulation |                                                   |
\*--------------------------------------------------------------------------------*/
FoamFile
{
   version     2.0;
   format      ascii;
   class       dictionary;
   object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.001;

vertices

(
   (-2000 127 0)
   (0 127 0)
   (70 106 0)
   (570 106 0)
   (570 260 0)
   (70 260 0)
   (0 260 0)
   (-2000 260 0)
   (-2000 127 10)
   (0 127 10)
   (70 106 10)
   (570 106 10)
   (570 260 10)
   (70 260 10)
   (0 260 10)
   (-2000 260 10)
);

blocks
(
   hex (0 1 6 7 8 9 14 15) (500 30 1) simpleGrading (1 2 1)
   hex (1 2 5 6 9 10 13 14) (20 30 1) simpleGrading (1 2 1)
   hex (2 3 4 5 10 11 12 13) (50 30 1) simpleGrading (4 2 1)

);

edges

(
   arc 1 2 (35 122.0819396 0)
   arc 10 9 (35 122.0819396 10)

);
```

```
patches
(
      patch inlet
   (
      (0 8 15 7)
   )
   patch outlet
   (
      (3 11 12 4)
   )
   wall top
   (
      (7 6 14 15)
      (6 5 13 14)
      (5 4 12 13)
   )
   wall bottom
   (
      (0 1 9 8)
      (1 2 10 9)
      (2 3 11 10)
   )
   empty frontAndBack
      (0 1 6 7)
      (1 2 5 6)
      (2 3 4 5)
      (8 9 14 15)
      (9 10 13 14)
      (10 11 12 13)
    )
);
mergePatchPairs
(
);
// ************************************************************************* //
```

File: *U*

```
/*--------------------------------------------------------------------------------*\
|  ========                     |                                                  |
|  \\        /     F ield       | OpenFOAM: The Open Source CFD Toolbox  |
|   \\      /      O peration   | Version:  1.5                                    |
|    \\    /       A nd         | Web:      http://www.OpenFOAM.org                |
|     \\  /        M anipulation  |                                                |
\*--------------------------------------------------------------------------------*/
FoamFile
{
   version     2.0;
   format      ascii;
   class       volVectorField;
   object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform (0 0 0);

boundaryField
{
   inlet
   {
      type           fixedValue;
      value          uniform (20.4 0 0);
   }

   outlet
   {
      type           zeroGradient;
   }

   top
   {
      type           fixedValue;
      value          uniform (0 0 0);
   }

   bottom
   {
      type           fixedValue;
      value          uniform (0 0 0);
   }

   frontAndBack
   {
      type           empty;
   }
}

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

File : *p*

```
/*--------------------------------------------------------------------------------*\
|  ========                 |                                                    |
|  \\        /    F ield     | OpenFOAM: The Open Source CFD Toolbox              |
|   \\      /     O peration | Version:  1.5                                      |
|    \\    /      A nd       | Web:      http://www.OpenFOAM.org                  |
|     \\  /       M anipulation |                                                  |
\*--------------------------------------------------------------------------------*/
FoamFile
{
   version     2.0;
   format      ascii;
   class       volScalarField;
   object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform 0;

boundaryField
{
   inlet
   {
      type          zeroGradient;
   }

   outlet
   {
      type          fixedValue;
      value          uniform 0;
   }

   top
   {
      type          zeroGradient;
   }

   bottom
   {
      type          zeroGradient;
   }

   frontAndBack
   {
      type          empty;
   }
}

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

File: *k*

```
/*--------------------------------------------------------------------------------*\
|  =======                 |                                                     |
|  \\      /    F ield     | OpenFOAM: The Open Source CFD Toolbox               |
|   \\    /     O peration | Version:  1.5                                       |
|    \\  /      A nd       | Web:      http://www.OpenFOAM.org                   |
|     \\/       M anipulation |                                                   |
\*--------------------------------------------------------------------------------*/

FoamFile
{
   version     2.0;
   format      ascii;
   class       volScalarField;
   object      k;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform 0.375;

boundaryField
{
   inlet
   {
      type           fixedValue;
      value          uniform 0.375;
   }

   outlet
   {
      type           zeroGradient;
   }

   top
   {
      type           zeroGradient;
   }

   bottom
   {
      type           zeroGradient;
   }

   frontAndBack
   {
      type           empty;
   }
}

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

File : ***epsilon***

```
/*--------------------------------------------------------------------------------------*\
|  ========                     |                                                      |
|  \\        /     F ield       | OpenFOAM: The Open Source CFD Toolbox  |
|   \\      /      O peration   | Version:  1.5                                        |
|    \\  /         A nd         | Web:      http://www.OpenFOAM.org                    |
|     \\/          M anipulation  |                                                    |
\*--------------------------------------------------------------------------------------*/
FoamFile
{
   version     2.0;
   format      ascii;
   class       volScalarField;
   object      epsilon;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform 14.855;

boundaryField
{
   inlet
   {
      type           fixedValue;
      value          uniform 14.855;
   }

   outlet
   {
      type           zeroGradient;
   }

   top
   {
      type           zeroGradient;
   }

   bottom
   {
      type           zeroGradient;
   }

   frontAndBack
   {
      type           empty;
   }
}

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

File: *sampleDict*

```
/*--------------------------------------------------------------------------------------*\
|  =======                    |                                                         |
|  \\         /    F ield     | OpenFOAM: The Open Source CFD Toolbox  |
|   \\       /     O peration  | Version:  1.5                                           |
|    \\     /      A nd        | Web:      http://www.OpenFOAM.org                       |
|     \\   /       M anipulation  |                                                      |
\*--------------------------------------------------------------------------------------*/
FoamFile
{
   version     2.0;
   format      ascii;
   class       dictionary;
   object      sampleDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
interpolationScheme cellPoint;
setFormat     raw;
sets
(
   line_1_k_eps_coarse_v6
   {
      type          uniform;
      axis          y;
      start         (-0.14 0.127 0.005);
      end           (-0.14 0.197 0.005);
      nPoints       1000;
   }
   line_2_k_eps_coarse_v6
   {
      type          uniform;
      axis          y;
      start         (0.0 0.127 0.005);
      end           (0.0 0.197 0.005);
      nPoints       1000;
   }
   line_3_k_eps_coarse_v6
   {
      type          uniform;
      axis          y;
      start         (0.0539 0.115 0.005);
      end           (0.0539 0.197 0.005);
      nPoints       1000;
   }
   );
surfaces
();
fields
(
   U
);
// ************************************************************************* //
```

Appendix B: The simpleFoam code

File: *simpleFoam*

```
/*---------------------------------------------------------------------------*\
|  =======                 |                                                 |
|  \\      /    F ield     | OpenFOAM: The Open Source CFD Toolbox           |
|   \\    /     O peration | Version:  1.5                                   |
|    \\  /      A nd       | Web:      http://www.OpenFOAM.org               |
|     \\/       M anipulation |                                               |
\*---------------------------------------------------------------------------*/
#include "fvCFD.H"
#include "incompressible/singlePhaseTransportModel/singlePhaseTransportModel.H"
#include "incompressible/RASModel/RASModel.H"
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
int main(int argc, char *argv[])
{
#   include "setRootCase.H"
#   include "createTime.H"
#   include "createMesh.H"
#   include "createFields.H"
#   include "initContinuityErrs.H"

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

    Info<< "\nStarting time loop\n" << endl;

    for (runTime++; !runTime.end(); runTime++)
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;

#       include "readSIMPLEControls.H"
#       include "initConvergenceCheck.H"

        p.storePrevIter();

        // Pressure-velocity SIMPLE corrector
        {
#           include "UEqn.H"
#           include "pEqn.H"
        }

        turbulence->correct();

        runTime.write();

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;

#       include "convergenceCheck.H"
    }
    Info<< "End\n" << endl;
    return(0);
}

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

File : *UEqn.H*

```
// Solve the Momentum equation
    tmp<fvVectorMatrix> UEqn
    (
        fvm::div(phi, U)
      + turbulence->divDevReff(U)
    );
    UEqn().relax();
    eqnResidual = solve
    (
        UEqn() == -fvc::grad(p)
    ).initialResidual();
    maxResidual = max(eqnResidual, maxResidual);
```

File: *pEqn.H*

```
    p.boundaryField().updateCoeffs();

    volScalarField AU = UEqn().A();
    U = UEqn().H()/AU;
    UEqn.clear();
    phi = fvc::interpolate(U) & mesh.Sf();
    adjustPhi(phi, U, p);
    // Non-orthogonal pressure corrector loop
    for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
    {
        fvScalarMatrix pEqn
        (
            fvm::laplacian(1.0/AU, p) == fvc::div(phi)
        );
        pEqn.setReference(pRefCell, pRefValue);
        // retain the residual from the first iteration
        if (nonOrth == 0)
        {
            eqnResidual = pEqn.solve().initialResidual();
            maxResidual = max(eqnResidual, maxResidual);
        }
        else
        {
            pEqn.solve();
        }
        if (nonOrth == nNonOrthCorr)
        {
            phi -= pEqn.flux();
        }
    }
#   include "continuityErrs.H"
    // Explicitly relax pressure for momentum corrector
    p.relax();
    // Momentum corrector
    U -= fvc::grad(p)/AU;
    U.correctBoundaryConditions();
```

File : *kEpsilon.C*

……

```
tmp<fvVectorMatrix> kEpsilon::divDevReff(volVectorField& U) const
{
    return
    (
      - fvm::laplacian(nuEff(), U)
      - fvc::div(nuEff()*dev(fvc::grad(U)().T()))
    );
}
```

……

```
    volScalarField G = nut_*2*magSqr(symm(fvc::grad(U_)));

#   include "wallFunctionsI.H"

    // Dissipation equation
    tmp<fvScalarMatrix> epsEqn
    (
        fvm::ddt(epsilon_)
      + fvm::div(phi_, epsilon_)
      - fvm::Sp(fvc::div(phi_), epsilon_)
      - fvm::laplacian(DepsilonEff(), epsilon_)
     ==
        C1_*G*epsilon_/k_
      - fvm::Sp(C2_*epsilon_/k_, epsilon_)
    );

    epsEqn().relax();

#   include "wallDissipationI.H"

    solve(epsEqn);
    bound(epsilon_, epsilon0_);

    // Turbulent kinetic energy equation
    tmp<fvScalarMatrix> kEqn
    (
        fvm::ddt(k_)
      + fvm::div(phi_, k_)
      - fvm::Sp(fvc::div(phi_), k_)
      - fvm::laplacian(DkEff(), k_)
     ==
        G
      - fvm::Sp(epsilon_/k_, k_)
    );

    kEqn().relax();
    solve(kEqn);
    bound(k_, k0_);

    // Re-calculate viscosity
    nut_ = Cmu_*sqr(k_)/epsilon_;

#   include "wallViscosityI.H"

}
```

DEFENCE SCIENCE AND TECHNOLOGY GROUP
DOCUMENT CONTROL DATA

1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)

2. TITLE

RANS Simulations using OpenFOAM Software

3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION)

Document	(U)
Title	(U)
Abstract	(U)

4. AUTHOR(S)

D.A. Jones, M. Chapuis, M. Liefvendahl, D. Norrison, and R. Widjaja

5. CORPORATE AUTHOR

Defence Science and Technology Group
506 Lorimer St
Fishermans Bend Victoria 3207 Australia

6a. DST Group NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DST-Group-TR-3204	AR-016-502	Technical Report	January 2016

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. NO. OF PAGES	12. NO. OF REFERENCES
2012/1175723	07386	FSP	49	25

DST Group Publications Repository

http://dspace.dsto.defence.gov.au/dspace/

14. RELEASE AUTHORITY

Chief, Maritime Platforms Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, EDINBURGH, SA 5111

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CITATION IN OTHER DOCUMENTS Yes

18. DST Group RESEARCH LIBRARY THESAURUS

Computational Fluid Dynamics, OpenFOAM, Reynolds Averaged Navier Stokes

19. ABSTRACT

The use of the OpenFOAM software suite for the performance of Reynolds-Averaged Navier-Stokes (RANS) simulations is described and illustrated by applying the simpleFoam solver to two case studies; two dimensional flow along a curved ramp and three dimensional flow along the hull of a generic conventional submarine. A detailed description of the use of the code is provided and aspects of the discretization schemes, solution solvers, turbulence schemes and boundary conditions are discussed.