This document was prepared for the ETI by third parties under contract to the ETI. The ETI is making these documents and data available to the public to inform the debate on low carbon energy innovation and deployment.

Programme Area: Marine

Project: PerAWAT

Title: Tidal Array Scale Numerical Modelling: Interactions within a Farm (Unsteady Flow)

Abstract:

This deliverable demonstrates the functionality of an actuator disc model of a tidal turbine, which is performing under Basin Scale conditions with mesoscale tidal flows to create three-dimensional unsteady Reynolds Averaged Navier-stokes (uRANS) simulations.

Context:

The Performance Assessment of Wave and Tidal Array Systems (PerAWaT) project, launched in October 2009 with £8m of ETI investment. The project delivered validated, commercial software tools capable of significantly reducing the levels of uncertainty associated with predicting the energy yield of major wave and tidal stream energy arrays. It also produced information that will help reduce commercial risk of future large scale wave and tidal array developments.

PerAWaT (MA 1003) report for WG3 WP2 D5b

Tidal array scale numerical modelling. Interactions within a farm (unsteady Flow).

Participant lead on the deliverable: DA Olivieri
Other participant: DM Ingram

Institute for Energy Systems
School of Engineering
The University of Edinburgh
King's Buildings, Edinburgh EH9 3JL
30 October 2012

Authored	DA Olivieri
Checked	DM Ingram
Signed off	IG Bryden

Contents

1. Executive summary

The aim of this deliverable is to demonstrate the functionality of an actuator disc model of a tidal turbine, which is performing under Basin Scale conditions with mesoscale tidal flows. The work therefore falls within the category of "*turbine modelled/farm resolved*" three-dimensional unsteady Reynolds Averaged Navier-stokes (uRANS) simulations. By averaging the results from these tests, the output can be used in the creation of shallow water equations, representing models of turbines, which are incorporated into basin scale models in WG2 WP3. By employing a uRANS approach, the calculation time can be much faster than if a blade-resolved approach was taken. Ultimately, in environments with significant anisotropic turbulence, a Large Eddy Simulation (LES) would be employed instead [1]. For the purpose of this deliverable, avoiding otherwise extremely large computational costs associated with LES, a more isotropic tidal flow environment has been considered justifying the application of the cheaper uRANS approach involving a k-ε turbulence model. As a result, the Sound of Islay tidal environment has been chosen as a suitable test bed for a tidal array for this investigation [2].

In this report, the development, testing and validation of a mesoscale, three-dimensional tidal channel model, with the addition of an Actuator Disc model, is detailed. The modelling work uses the computational fluid dynamics software *Code Saturne*, with unsteady Reynolds-averaged Navier-Stokes equations resolving the turbulent tidal flow, including the interactions with the tidal farm, which is represented as a set of Actuator Discs on the local flow.

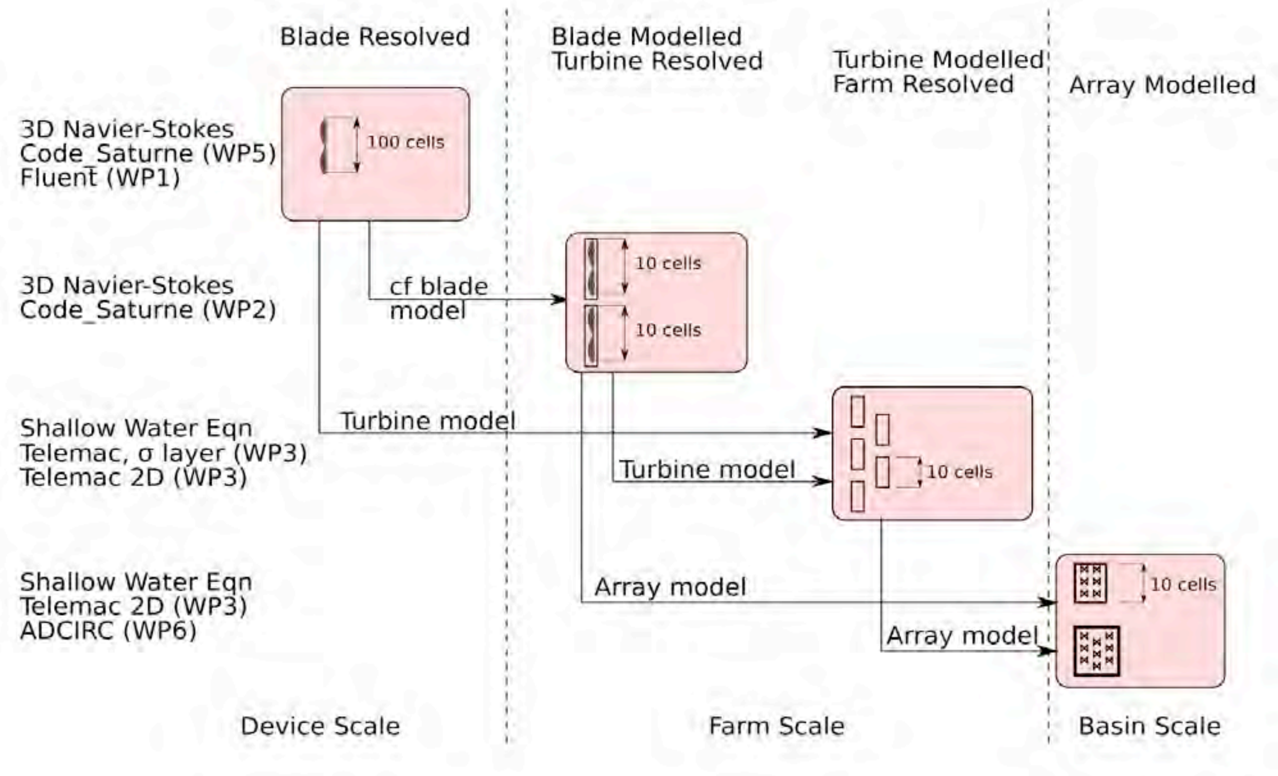

Figure 1: Different scales of turbine interaction associated with PerAWaT WG3 work group.

Earlier findings within deliverable WG3 WP2 D4 have been utilized to set up appropriate time-averaged downstream flow conditions to ensure characteristics of velocity, turbulent kinetic energy and turbulent dissipation profiles, which persist correctly at the point of flow coming into the turbines within the flow field.

The acceptance criteria for D5b are:

1) ***Code Saturne* input and output files for the simulations described in this report**.

   These are on the accompanying CD; a README file in the root directory of the CD describes the contents in full.

2) **That the report describes the following, for unsteady turbulent flow conditions:**

   a) The required modifications to previous Steady Tidal Array modelling, described in deliverable WG3-WP2-D5a at farm scale, to incorporate the tidal site characteristics of the suggested tidal site given with deliverable WG3-WP2-D4.
   b) Statistical methods employed to provide a modification to the earlier steady state parameterization of the wake. This is implicit in the reporting of section 4.

2. Previous Steady Array model studies at farm scale

The previous work, which was covered in deliverable WG3 WP2 D5a, gave an extensive assessment of tidal turbine modelling using an Actuator Disc model involving steady RANS. In D5a tests on a small array of turbines were performed, but to limit the computational costs of these simulations. symmetry conditions were used to minimise the size of the domain. These simulations were based on cases considered by the University of Manchester (in particular tests 19 and 20), both of which use two rows of turbines. It was agreed at the tidal sub-project meeting in September (document reference CR-P74/2012/) and at the 12th PerAWAT project steering committee meeting on the 13th September 2012 (Document No. 104325/BT/19), that Manchester test 13, which comprises a single row of three turbines, would be studied. To provide an additional validation case for the Blade Element Momentum Theory Actuator Disk (BEMT-AD) model, Manchester test 13 would be simulated under steady flume conditions, similar to those considered in D5a for Tests 19 and 20. This allows comparisons to be made with the available experimental data from the University of Manchester.

The meshing strategy employed is similar to that used in the staggered case in D5a. Each individual turbine is meshed using cylindrical O-grid to discretise its rotor plane. Following the approach used in D5a, only the nacelles have been included in the computation and there are no support structures. In meshing the nacelle an H-grid region has been used at the centre of the O-grid. In the present case, the full width of the array has been simulated and there is no central symmetry plane.

2.1 Computational set up

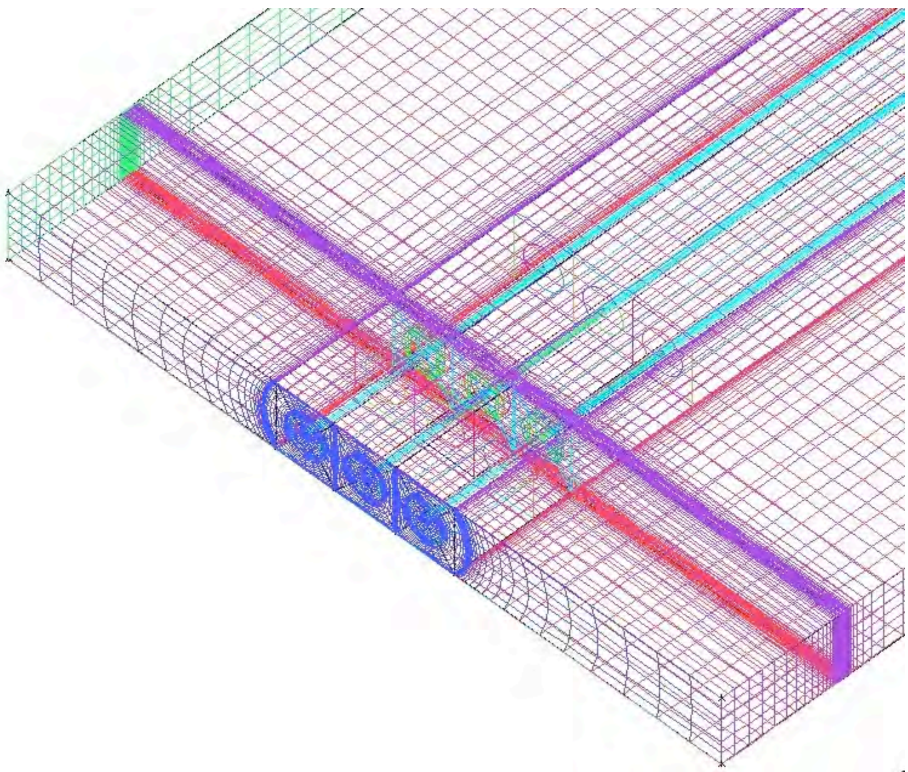

Figure 2 Detail from the ICEM mesh generator showing the cross-sectional grid (blue), the solid volumes for the nacelle (green) and the refined mesh region in the neighbourhood of the upstream actuator disc

As shown in Figure 2, the meshing strategy involves the **ICEM** package for a single array of three turbines. As can be seen, three cylindrical O-grids extend the length of the computational domain, with the rotor planes discretised appropriately. The O-grids in each case have 15 cells in the radial direction and 24 cells in the circumferential direction. Mesh stretching has been used to refine the grid at the edge of the rotor, so that the shear between the free stream flow and the fluid, which has passed through the actuator disk, is sufficiently resolved. Following the same strategy as was employed in the simulations of Manchester tests 19 and 20 (see WG3 WP2 D5a), a 6×6 H grid is used to model the nacelle. It should be noted that the nacelle does not form part of the BEMT-AD.

The blade geometry used in these tests is shown in Table 1. The required tables for coefficients of lift, C_L, and drag, C_D, against angle of attack, α, for the Goe804 airfoil were obtained from the University of Manchester and can be found in the file Goe804.txt. The lift and drag coefficients for the airfoil are plotted in Figure 3. The code given in `usinv.f90`, `ustsns.f90` was modified to create three actuator discs, located on the faces of the upstream rotors. The source code for the user routines, together with the run files blade.txt, circle.txt and Geo804.txt are included on the accompanying CD in the Double directory.

Table 1: Blade geometry for the three bladed horizontal axis turbines tested in the University of Manchester test13 array test.

Radial ordinate	Twist angle	Chord length	Thickness t/c	Section
0.001	38.4	0.015	100%	Circle
0.015	38.4	0.015	100%	Circle
0.033	22.9	0.0200	4%	Goe804
0.045	16.0	0.0300	4%	Goe804
0.059	11.3	0.0275	4%	Goe804
0.073	9.0	0.0250	4%	Goe804
0.085	7.5	0.0223	4%	Goe804
0.099	5.8	0.0195	4%	Goe804
0.113	4.1	0.0175	4%	Goe804
0.125	3.1	0.0155	4%	Goe804
0.135	2.6	0.0130	4%	Goe804

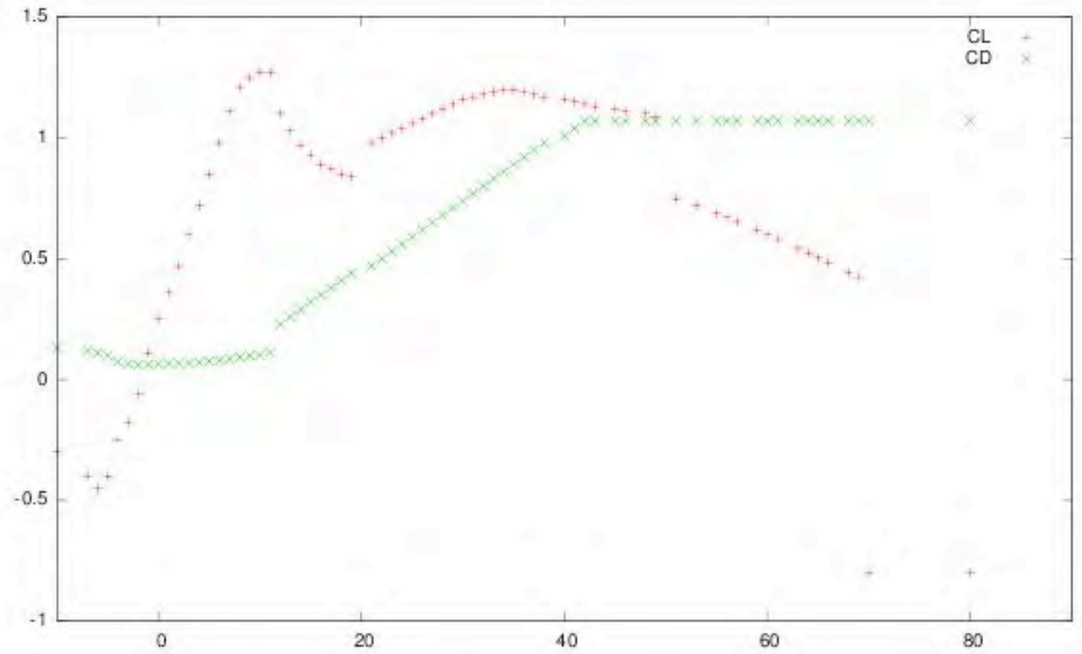

Figure 3 Coefficients of aerodynamic lift, C_L, and drag, C_D, for Goe804 aerofoil

The results from these simulations are presented in section 4.1.

3. Unsteady flow Array modelling at farm scale

To demonstrate the use of the BEMT-AD model in farm scale CFD models, a simulation of a two row, staggered array in the a tidal channel has been performed. The tidal channel used is that presented in WG3 WP2 D4 and is based on the Sound of Islay; A tidal straight between the islands of Islay and Jura in Scotland. The Sound of Islay has been selected by Scottish Power Renewables (UK) Ltd for a 10MW array demonstration site [2]. Following the approach followed in WG3 WP2 D4, the sound has been modelled as a rectangular domain 5km long and 1km wide.

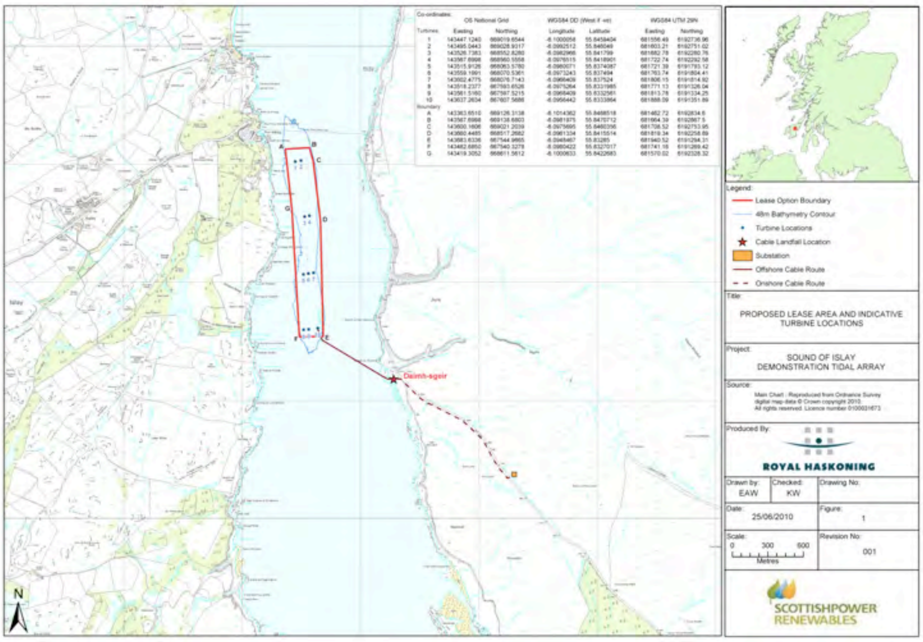

Figure 4 Sound of Islay 10MW tidal demonstration project, reproduced from [2].

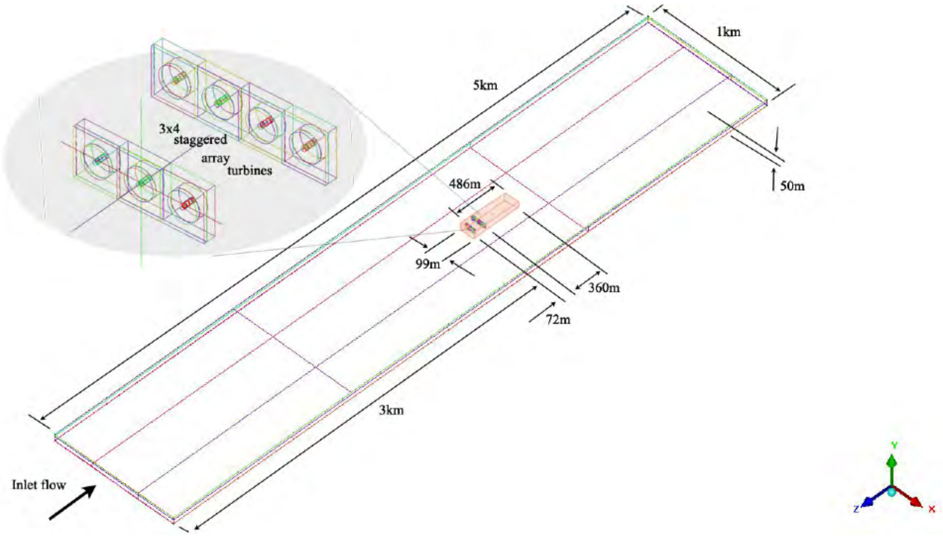

Figure 5 Isle of Islay flow field design for unsteady staggered array modelling at farm scale

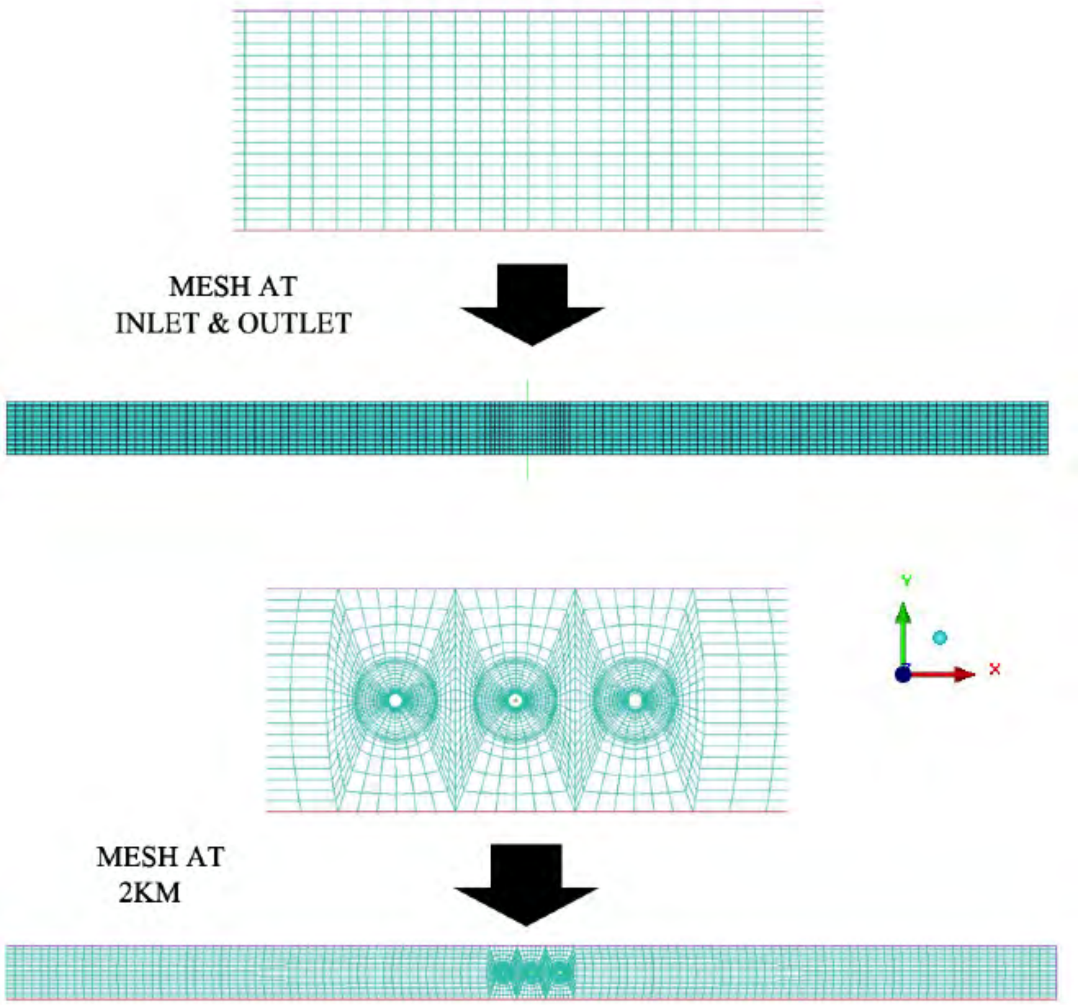

Figure 6 Mesh design variation normal to the flow direction at different cut points in the flow field

3.1 Computational set up

Figure 5 and Figure 6 indicate the necessary flow field design for unsteady modelling at farm scale. In accordance with the design described in WG3 WP2 D4 [3], The aspect ratio (length:height) of each cell was kept as near as possible to 3.2. High aspect ratio cells are critical for accurate and stable boundary layer development. Modelling the rotor planes (Figure 6) requires the aspect ratio of the cells in the neighbourhood of the turbines to be less than 3.2.An initial meshing strategy based on extruding the turbine mesh downstream proved unstable. Further computational experiments have shown that the CFD model will not break down, as long as the following criteria are met:

- The recommendations for boundary and flow conditions and general *Code Saturne* settings, outlined in Creech[3] from WG3 WP2 D4, are kept, with hub height at 25m.
- The placement of the first array of turbines should be 2km downstream of the inlet boundary, to allow the flow conditions to settle to a steady state and to provide the correct shear flow profile, representing a working environment for the staggered tidal array farm. (The y plane cutting the first array of turbines gives an upper middle region where $0.4 < y/y_{max} < 0.8$, which has flow speeds with a maximum deviation of 4% as stipulated by Creech[3]).

This approach allows a sufficiently accurate representation of the rotor plane for the BEMT-AD model and provides a stable CFD simulation of the tidal channel. It is also important to note that, in these simulations, the nacelle and support structure of the turbine are not represented. As such, these simulations are categorised as turbine modelled/farm resolved simulations (Figure 1).

Details of the implementation of the BEMT-AD model are presented in WG3 WP2 D5a chapter 2, together with a description of the input files required to set-up the BEMT-AD model. The *Code_Saturne* user Fortran routines used for this case can be found in Appendices B, C and D. For the present case, the blade design for the rotors follows the generic TGL design used as the basis for the EDF single-rotor flume tests (WG4 WP1) and the blade-resolved CFD simulations performed for WG3 WP5 D1. The operating conditions are for a design tip speed ratio of 4.5, when steady flow conditions have been met after several time steps. Gretton has covered the TGL design in considerable detail in deliverable WG3 WP5 D1 [4]. The blade.txt file and cl/cd graphs files given in the CD are based on the work reported in WG3 WP5 D1. The sections are used in TGL blade designs are based on the NACA six series airfoils 63_3-418 to 63_3-455. The 6-series airfoils are designed to maximise laminar flow compared with the equivalent NACA 1-series airfoil. The 6-series family of airfoils is described using six digits in the following sequence:

1. The number "6" indicating the series.
2. One digit describing the distance of the minimum pressure area in tens of percent of chord.
3. The subscript digit gives the range of lift coefficient in tenths above and below the design lift coefficient in which favourable pressure gradients exist on both surfaces
4. A hyphen.
5. One digit describing the design lift coefficient in tenths.
6. Two digits describing the maximum thickness in tens of percent of chord.

The TGL blades are thus designed to have at a minimum pressure at 30% of the chord, a design lift coefficient of 0.4 and a thickness of between 18% and 55%.

4. Results

4.1. Single row of turbines at farm scale under steady state conditions

In this *Code Saturne* simulation, as with all three of the Manchester comparison experiments, a rough-wall model has been used together with the *k-ε* turbulence model to model the Manchester experimental basin (see WG3 WP2 D5a). It should be noted, however, that the Manchester experimental tests were not designed for direct comparison with CFD simulations and, as a result, the boundary layer at the bottom of the basin is less well characterised than would normally be expected for CFD/Experimental comparisons. In figure 7, the basic flow field approximations are shown.

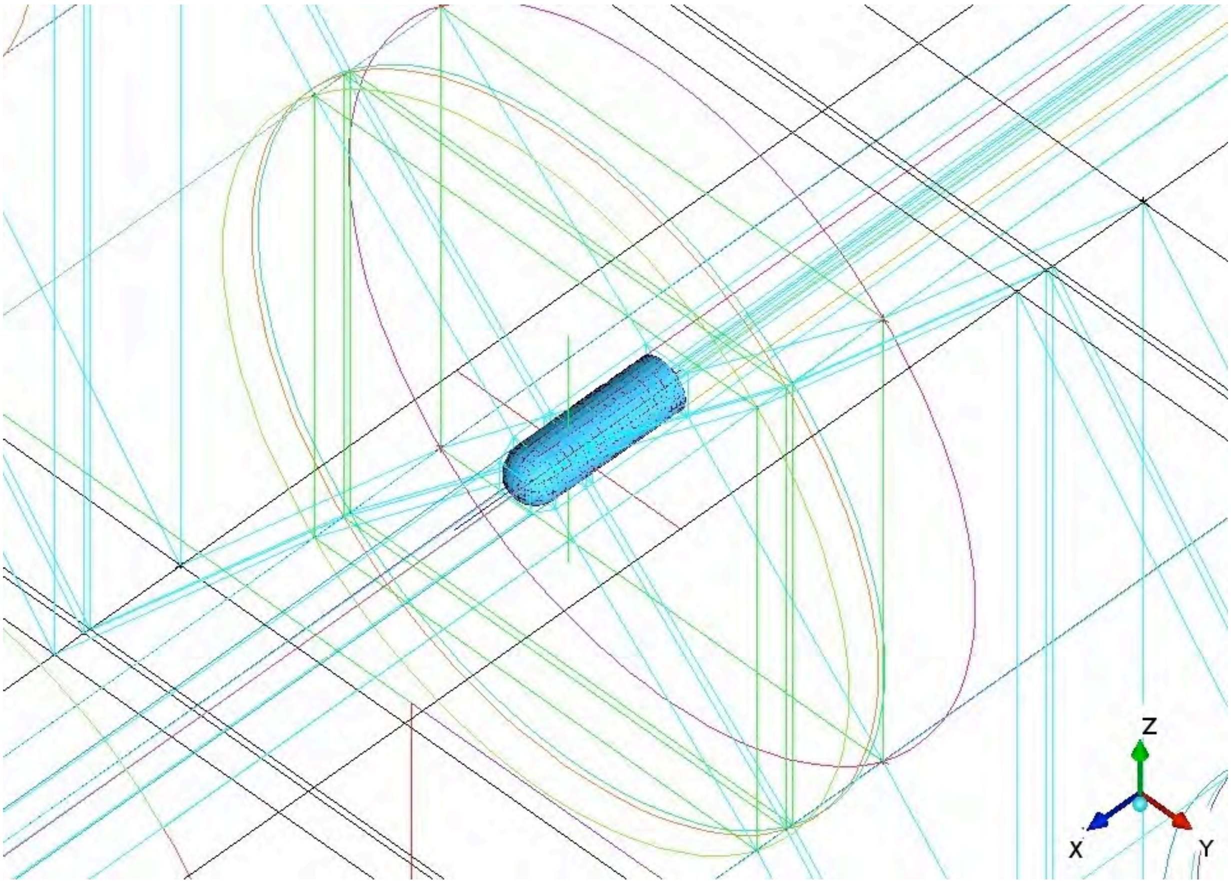

Figure 7 of a working turbine Flow field approximation

The nacelle can be seen as a blue object against a system of curves representing the flow field, with two closely positioned circles representing the swept volumes generated by a multi-bladed rotating turbine. The BEMT-AD model will introduce source/sink terms to represent the velocity changes and energy losses associated with the rotor's blades averaged over one complete rotation of the turbine. The test conditions for each turbine rig were as described by Stallard and Collins [5], on pages 10, 13 and 14. The rotor plane used in the computational simulation is slightly thicker than the width of the blades. This approximation is needed to comply with the meshing criteria advised for *Code Saturne* users [6] to ensure numerical stability. The presence of the nacelle in these tests makes the task of mesh creation for the flow field with ICEM more complex, as a smooth, well discretised, mesh is required on the nacelle surface and in the base region. Failure to mesh this with sufficient fidelity leads to poor predictions of the base flow behind the nacelle and poor resolution of the boundary layer around nacelle. Conversely, high resolution meshing of the nacelle leads to an unacceptably large mesh and very high computational costs.

The results for axial velocity Ux and Turbulence Intensity distributions along the geometric axis of symmetry of each turbine are shown in Figures 8 and 9, which show good agreement with experiment by six turbine diameters downstream. However, the near wake has a poorer agreement, due to the departure of the modelled nacelle geometry from that previously used in flume tests conducted at the University of Manchester.

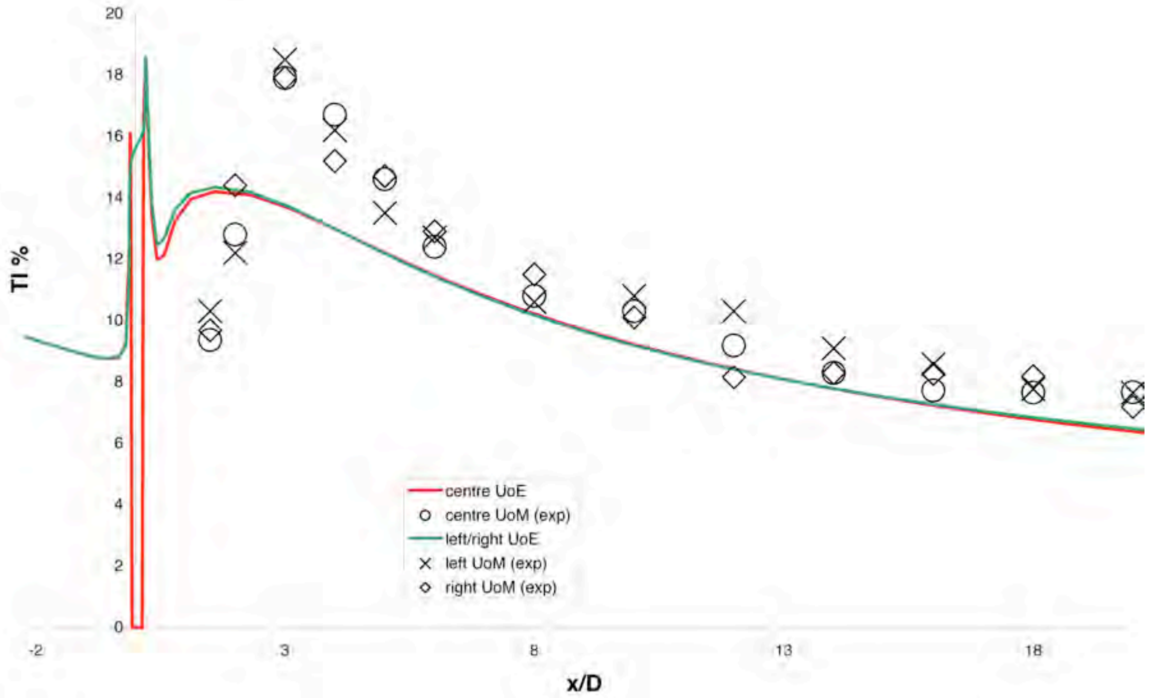

Figure 8 Distribution of Turbulence Intensity along the axis of system of each turbine for test 13

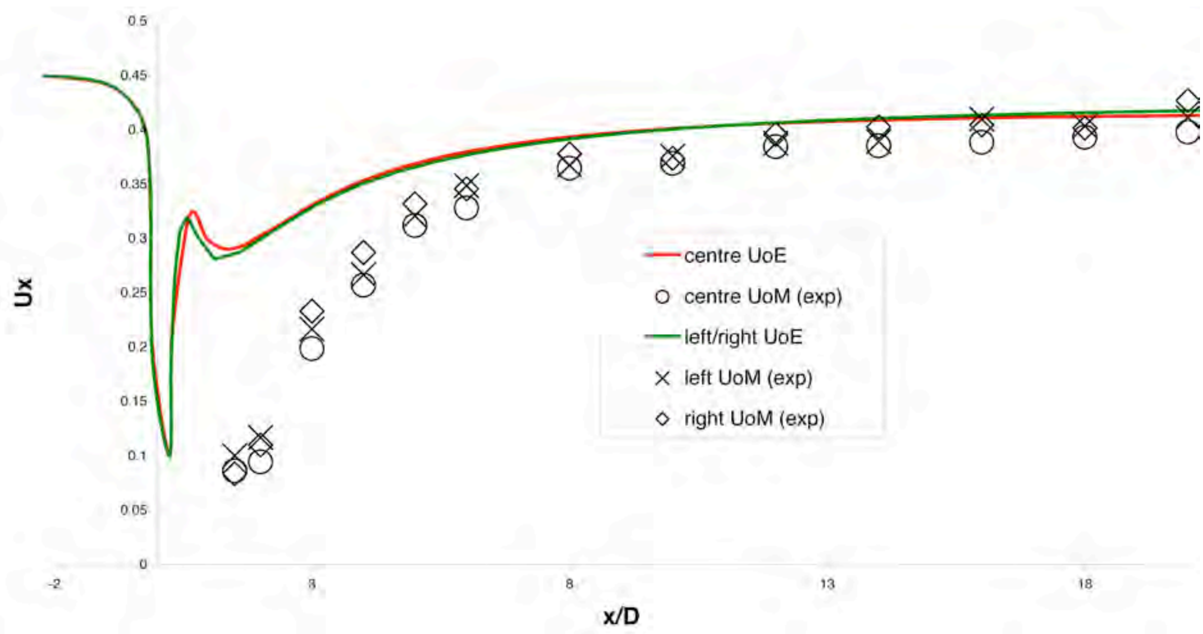

Figure 9 Distribution of axial velocity along the axis of symmetry of each turbine for test 13

The far wake agrees with the BEMT-AD model because the turbine rotor is the principal source of momentum extraction. Flow over bluff bodies (ie the nacelle and support structure) in the near wake create large eddies, leading to a significant, local, momentum deficit. Most of this loss of momentum is recovered, though mixing, by the time the far wake is reached. Accurate modelling of flow in the near wake region requires both high resolution meshing of the nacelle and support structures, and a blade resolved CFD model. The agreement between the BEMT-AD model and the experimental measurements in the fare wake zone is, therefore, as expected. Should higher fidelity modelling be needed of the near wake region either a BEMT Actuator Line model or a fully blade resolved computation is required (See WG3 WP2 D5a). This hypothesis is supported by Figures 10 to 27, which show how momentum is removed and, to some extent returned, with the exception of that taken by the turbine.

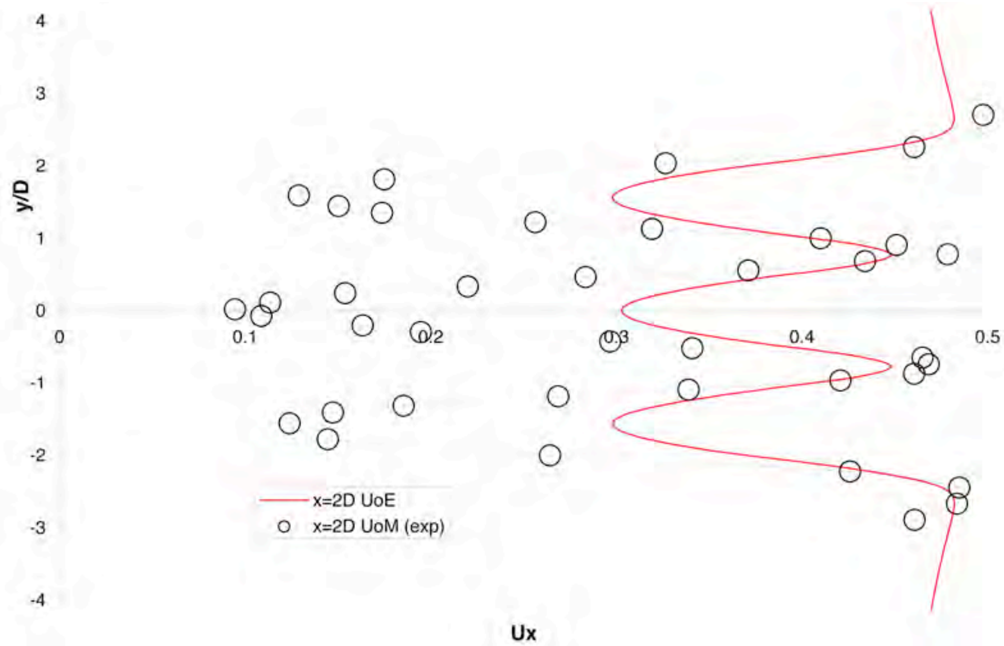

Figure 10 Ux distribution across the y plane at four diameters downstream of the turbine farm

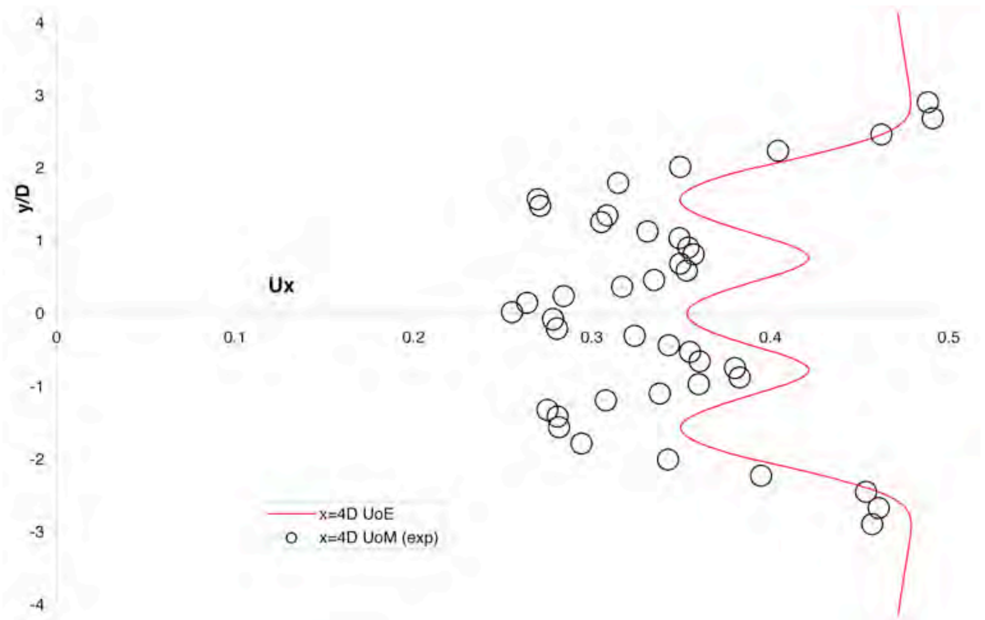

Figure 11 Ux distribution across the y plane at four diameters downstream of the turbine farm

The increasing thickness of the boundary layer in the region -1.0 < z/D < -0.8 (Figures 22 to 27) causes the CFD curve of the Ux distribution to “tail back. It can be noticed from these figures that the growth of the CFD boundary layer is significantly more pronounced than that seen in the experimental data. This difference is a direct result of not being able to correctly specify the boundary layer profile in region -1.0 < z/D < -0.8, a difficulty that arises because of the limited number of experimental data points in this region[1]. This difference causes the flow between the free surface at z/D=1 and flume floor at z/D=-1 to be more blocked, explaining why the CFD velocity curves are displaced to the right, as seen in Figures 12 to 15 and 24 to 27.

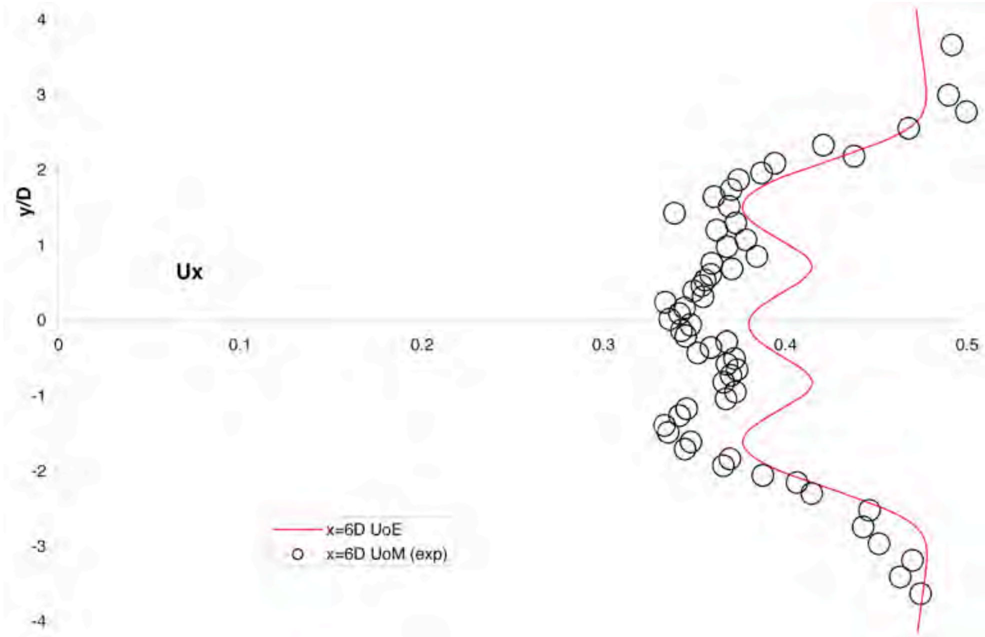

Figure 12 Ux distribution across the y plane at six diameters downstream of the turbine farm

[1] Recent communication with Stallard [7] over the nature of the original contract indicates that it was originally specified that no use would be made of UoM experimental data for CFD comparison. As such, the near bed profile was not specified for these experiments and the need for this data in PerAWaT has only arisen due to contract changes, which failed to take this issue into account!

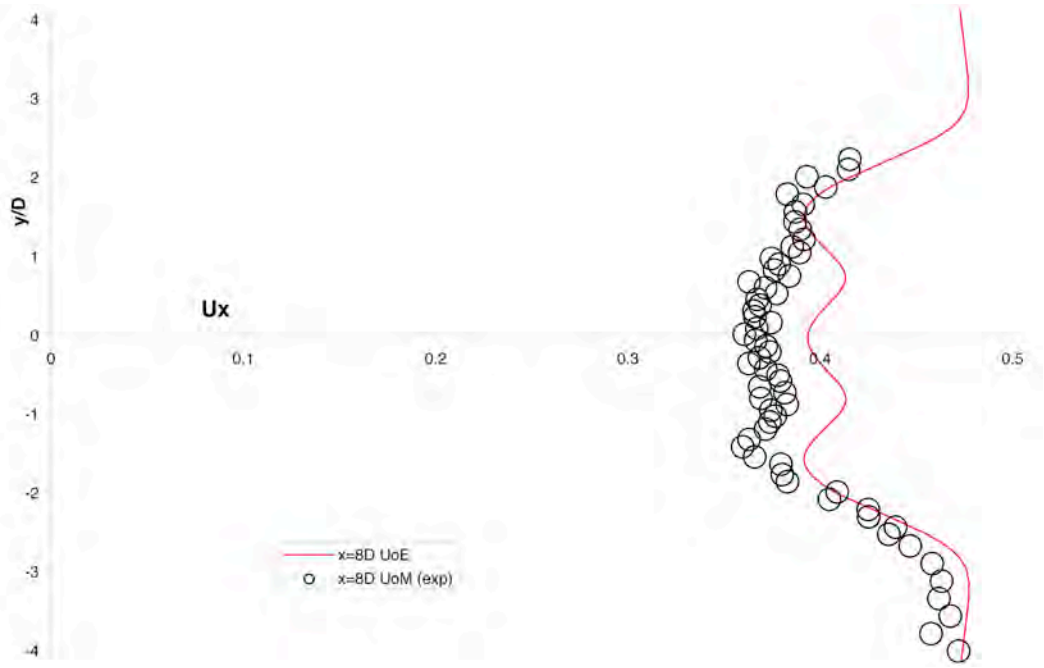

Figure 13 Ux distribution across the y plane at eight diameters downstream of the turbine farm

Figure 8 and figures 16 to 21 indicate significant differences in turbulence intensity distributions for the experimental data and the CFD predictions in the near wake. The simplified arrangement of the actuator disc, with a disc of thickness 0.01 times the turbine diameter, makes no provision for the actual rotor blade generated turbulence during one rotational sweep of the rotor. The RANS BEMT-AD model should have turbulence added artificially to compensate for this effect (This issue will be addressed in D6). Also, as previously described, the approximations to the nacelle geometry and lack of support structure have an influence on the local wake area flow features.

To complete the analysis of Manchester test 13 results, the loss of momentum from the wake can be accounted for by considering a more global verification of the effect. The use of parameters, namely the thrust and power coefficients, *CT* and *CP* respectively, for each turbine, has been investigated at different tip speed ratios, based on the upstream conditions of undisturbed velocity $U_\infty = 0.45$m/s, at a hub height of $H = 0.225$m, for their inflow weir conditions. Figure 28 indicates the extremely close agreement between theory and experiment for test 13. The influence of each of the three turbines on each other, causing the CT to be elevated beyond that normally experienced by a single turbine alone, is clear.

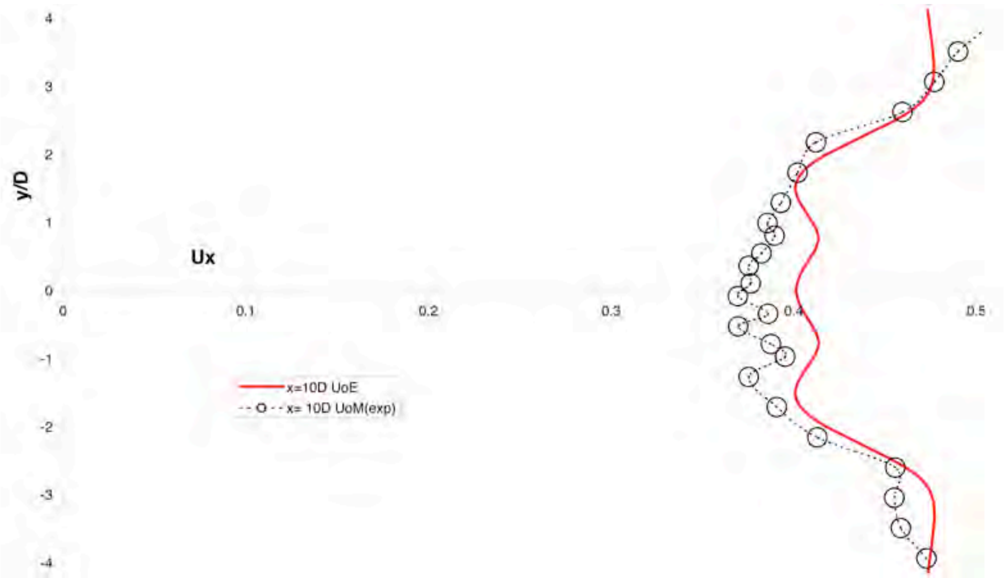

Figure 14 Ux distribution across the y plane at ten diameters downstream of the turbine farm

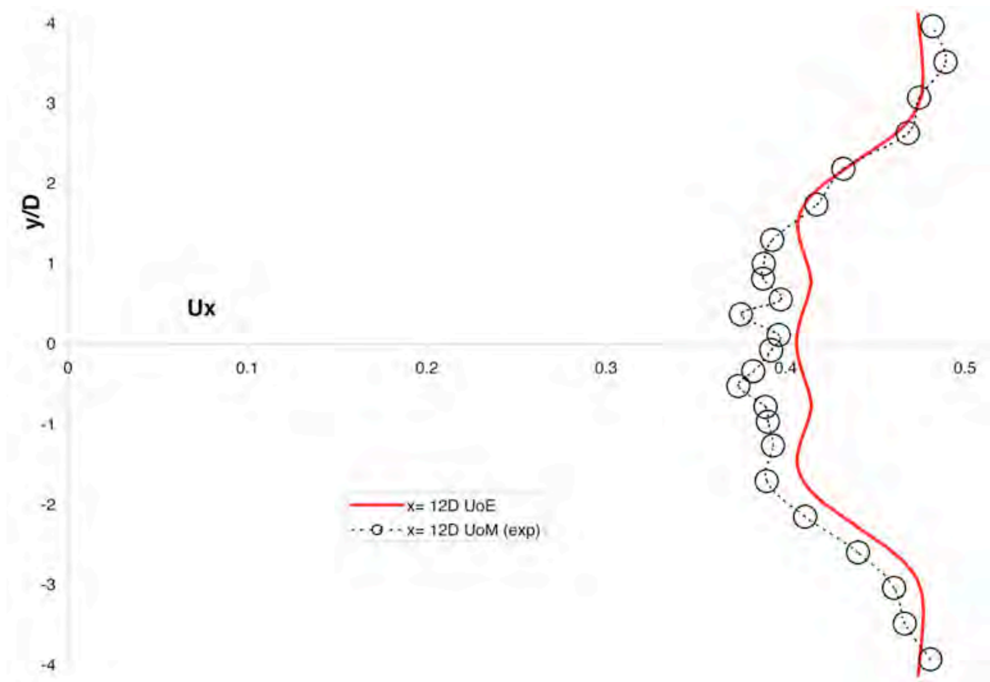

Figure 15 Ux distribution across the y plane at 12 turbine diameters downstream of the turbine farm

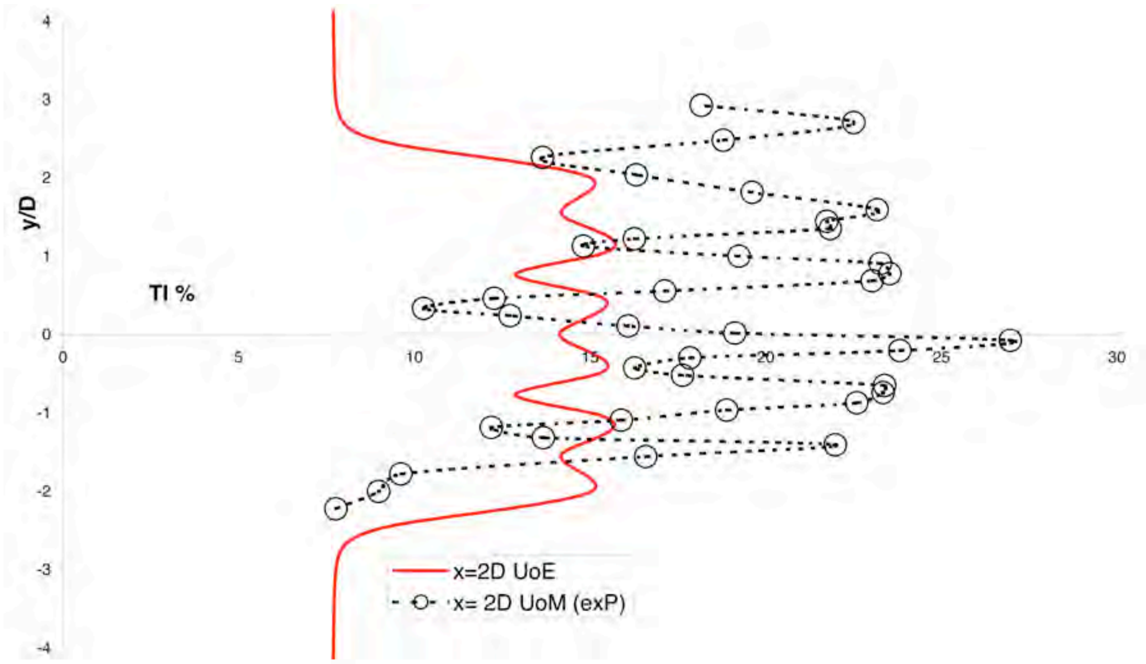

Figure 16 Turbulence intensity distribution across the y plane at two diameters downstream of the turbine farm

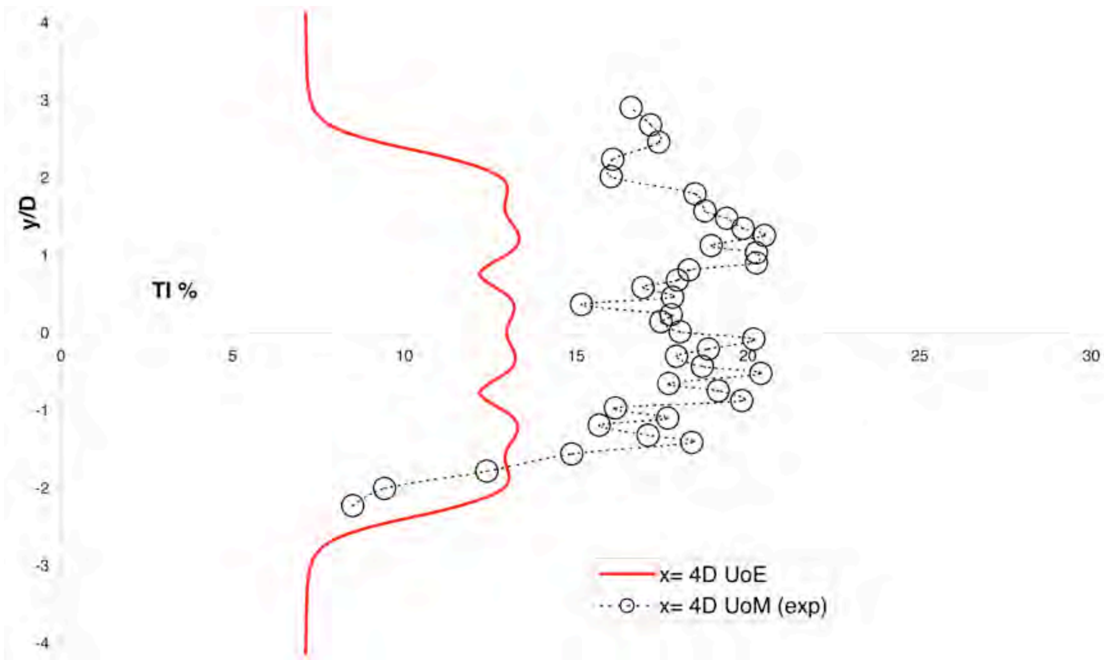

Figure 17 Turbulence intensity distribution across the y plane at four diameters downstream of the turbine farm

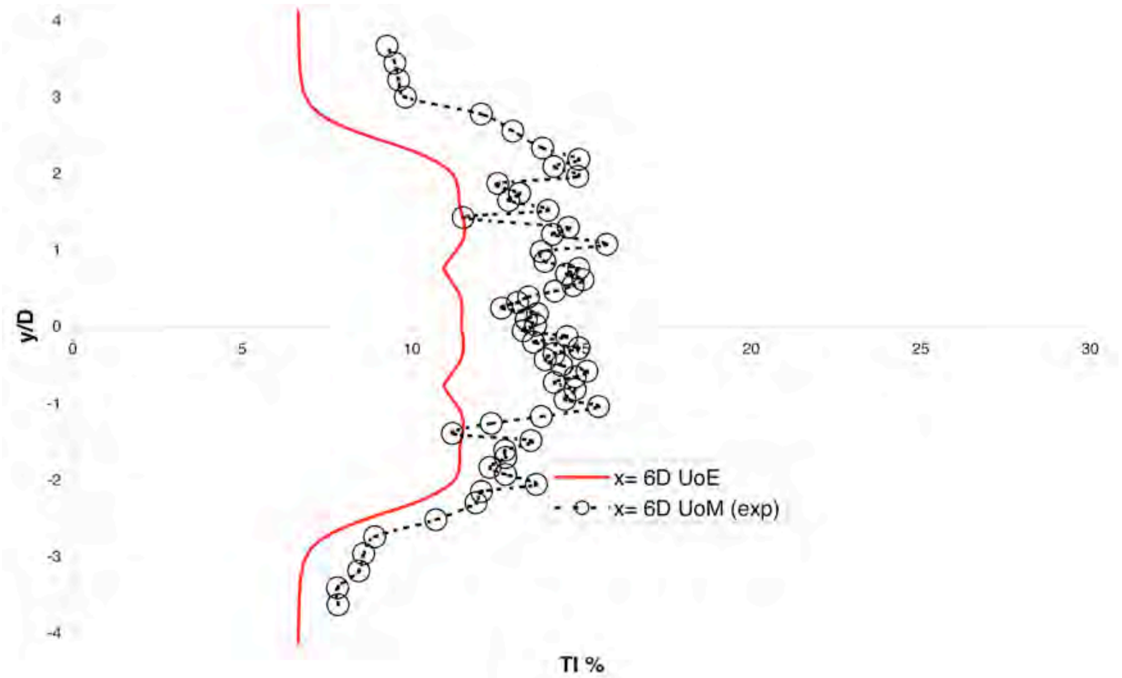

Figure 18 Turbulence intensity distribution across the y plane at six diameters downstream of the turbine farm

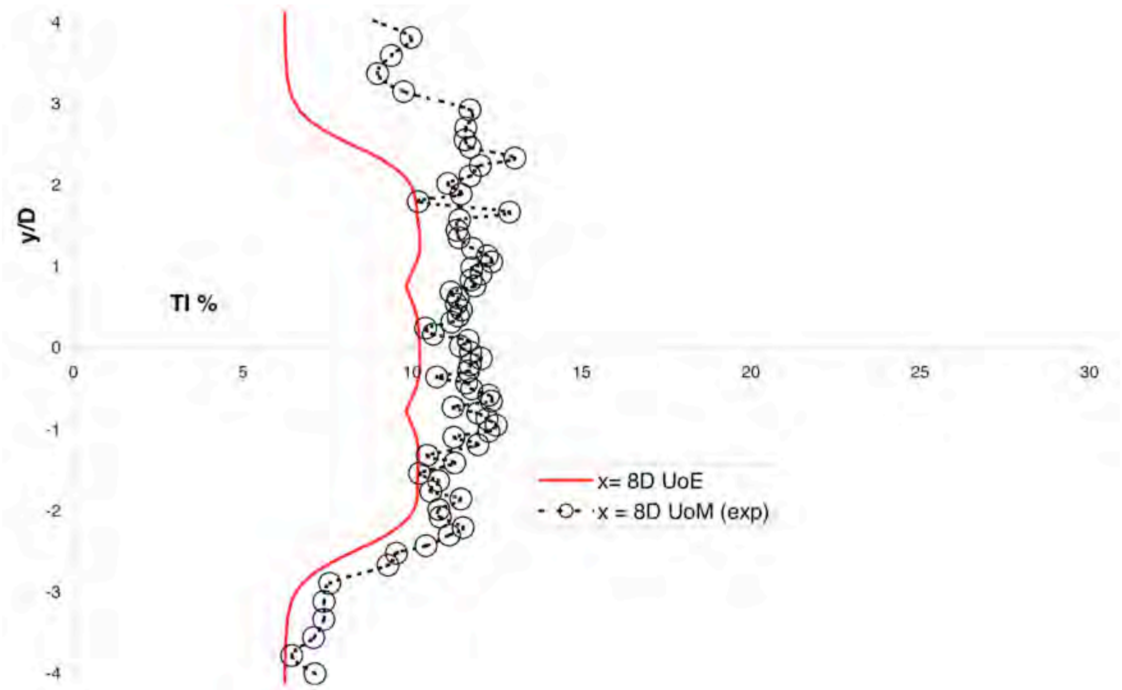

Figure 19 Turbulence intensity distribution across the y plane at eight diameters downstream of the turbine farm

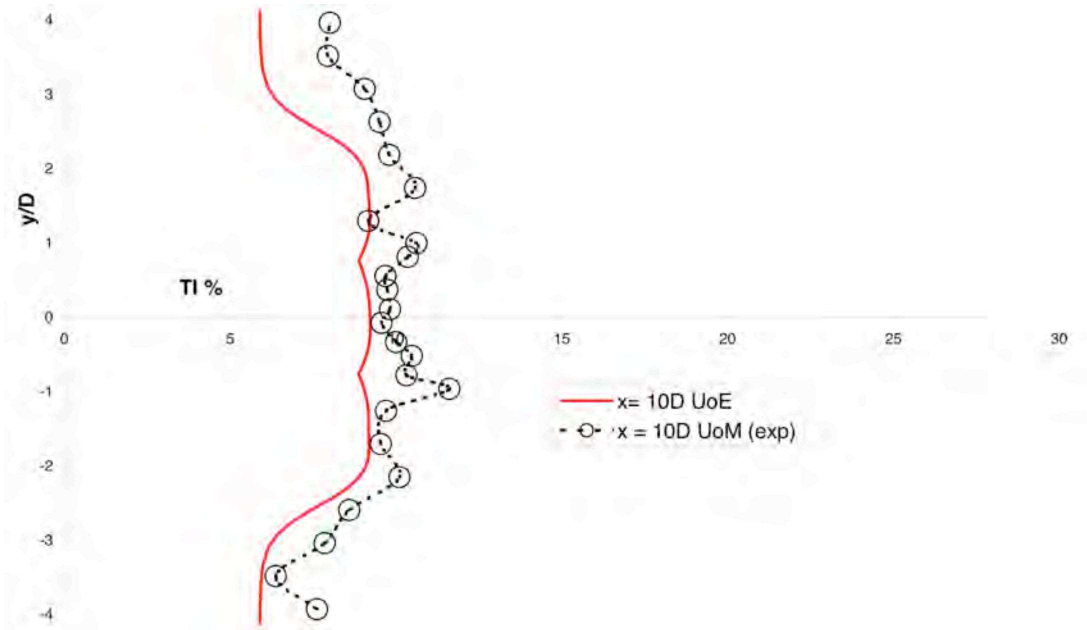

Figure 20 Turbulence intensity distribution across the y plane at ten diameters downstream of the turbine farm

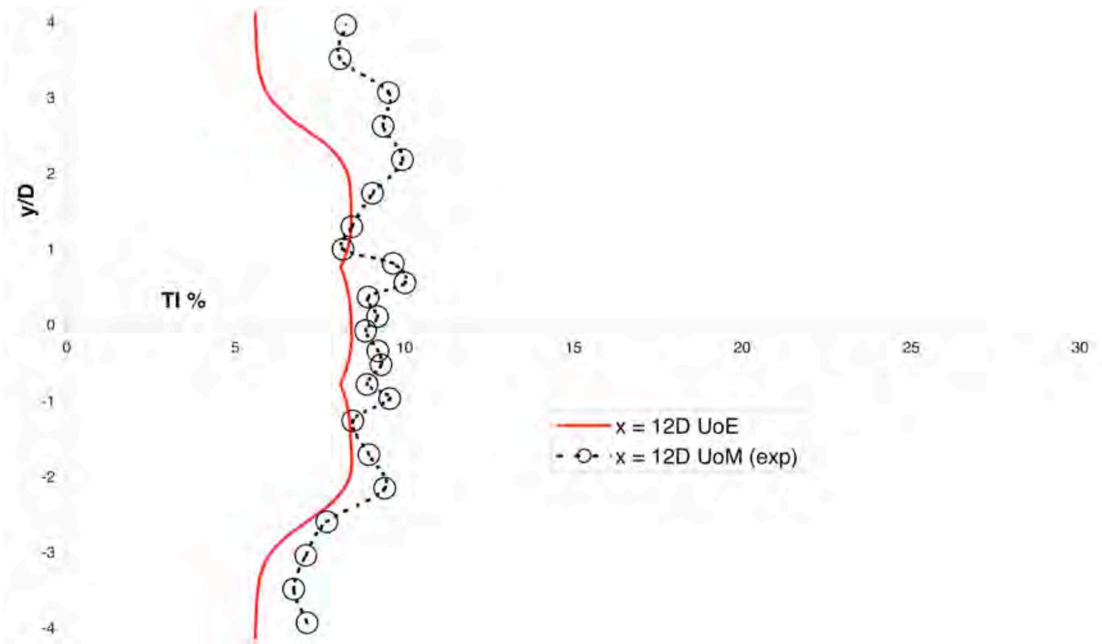

Figure 21 Turbulence intensity distribution across the y plane at twelve diameters downstream of the turbine farm

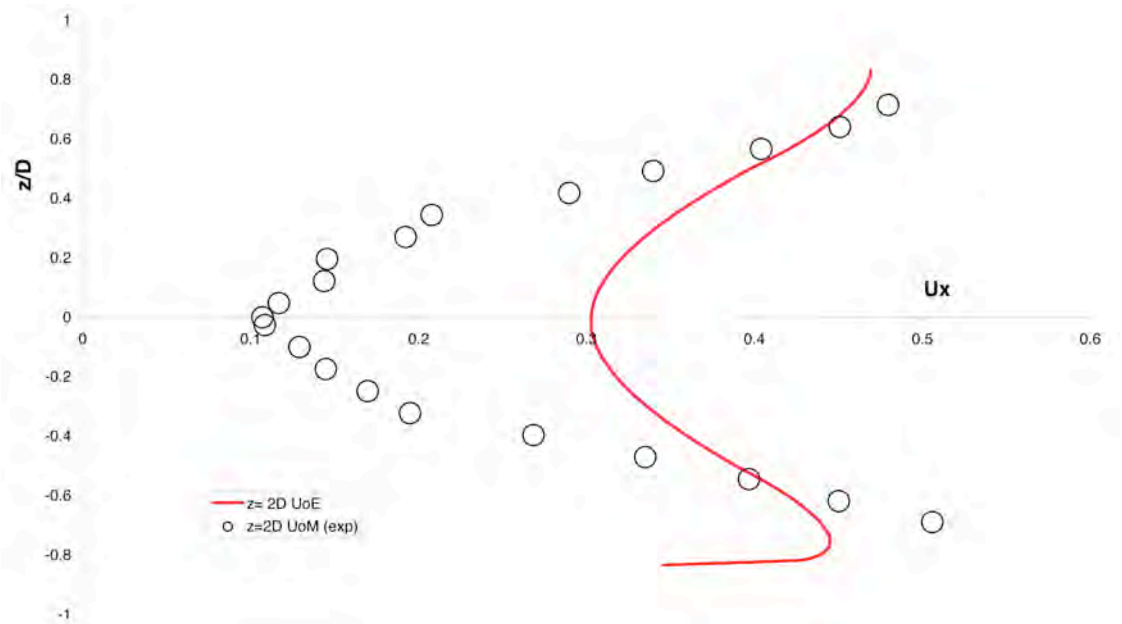

Figure 22 Averaged Ux distribution across the z cut plane at two diameters downstream of the turbine farm

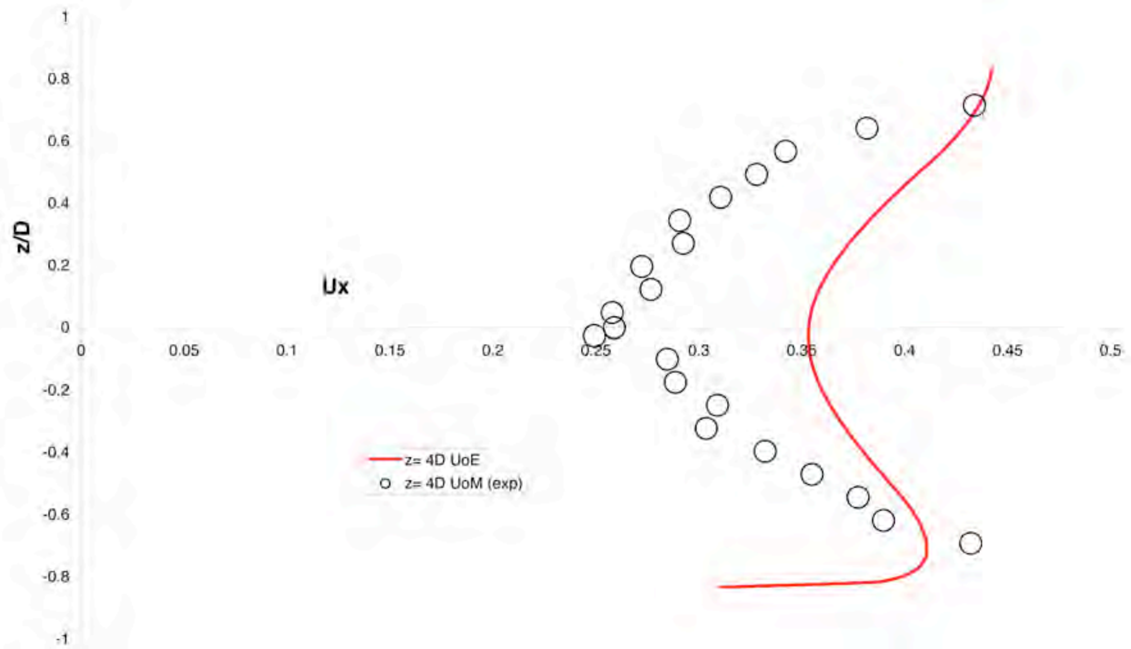

Figure 23 Averaged Ux distribution across the z cut plane at four diameters downstream of the turbine farm

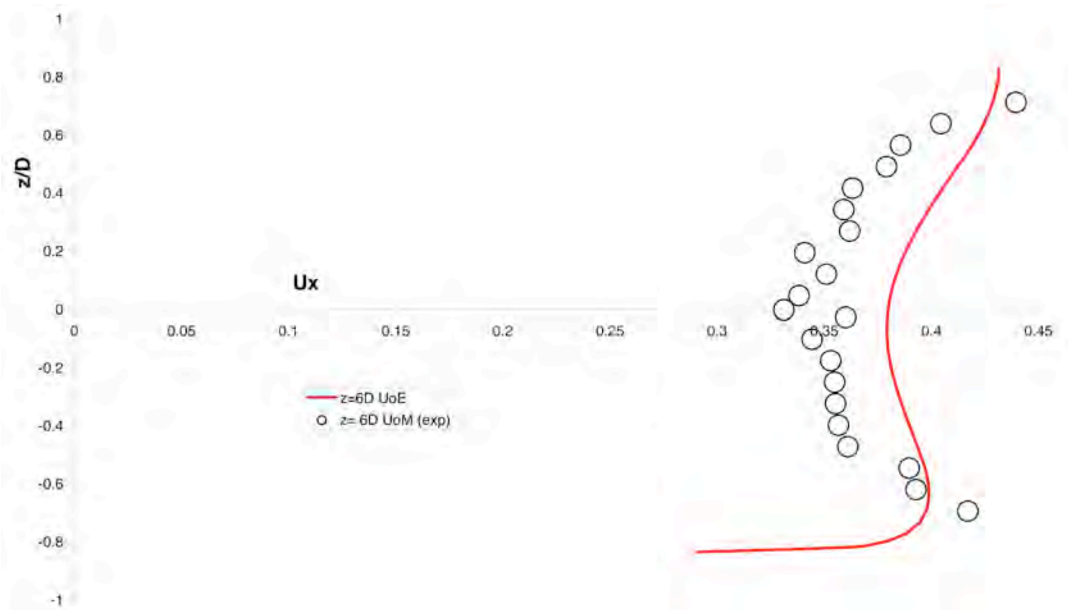

Figure 24 Averaged Ux distribution across the z cut plane at six diameters downstream of the turbine farm

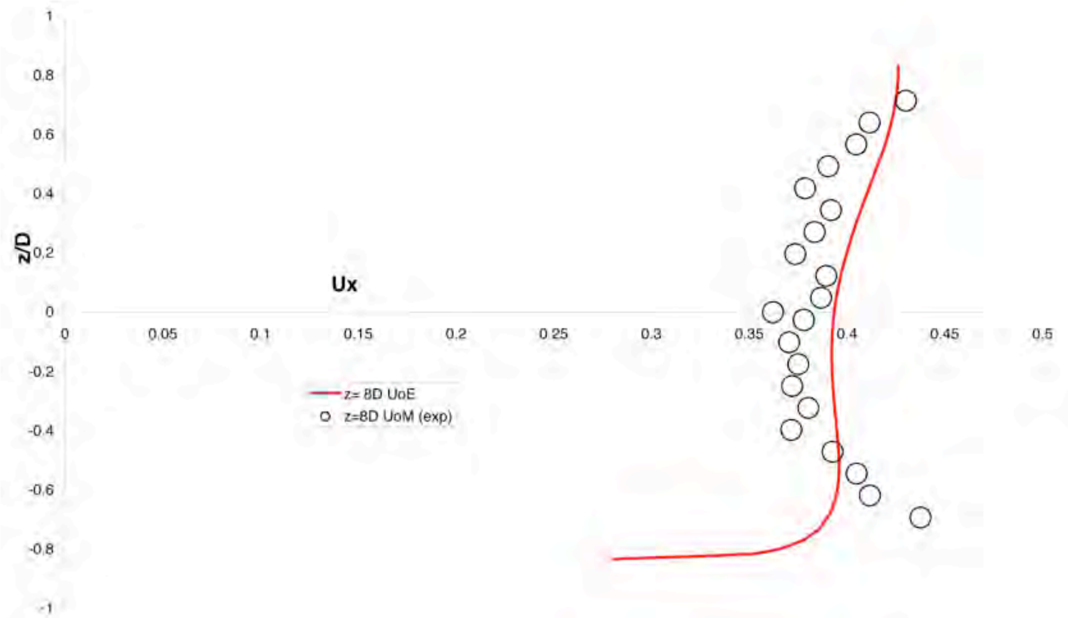

Figure 25 Averaged Ux distribution across the z cut plane at eight diameters downstream of the turbine farm

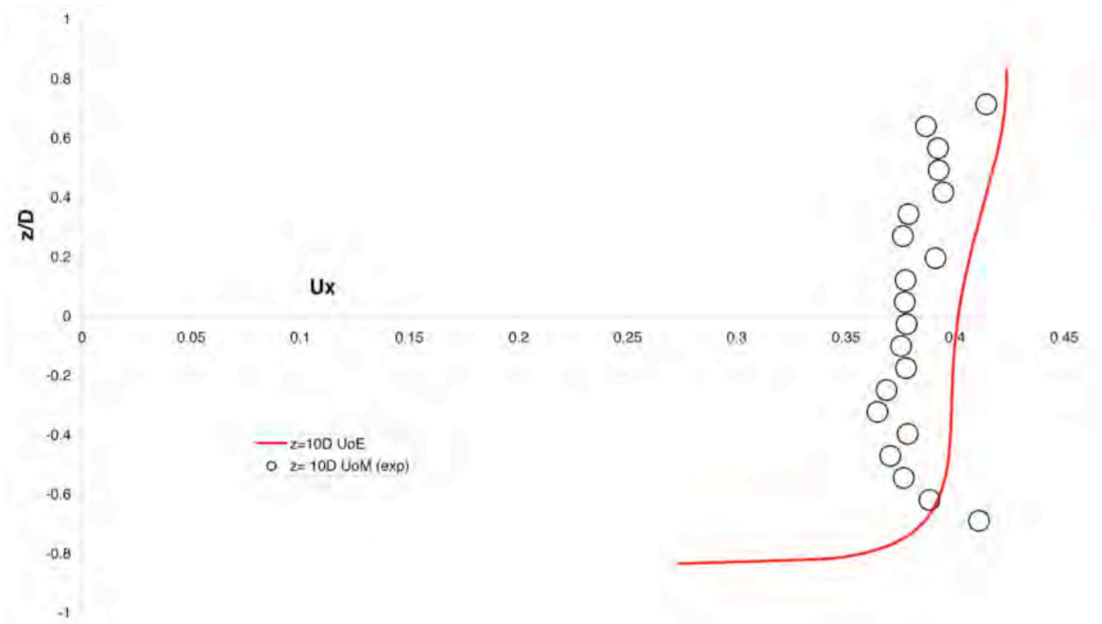

Figure 26 Averaged Ux distribution across the z cut plane at ten diameters downstream of the turbine farm

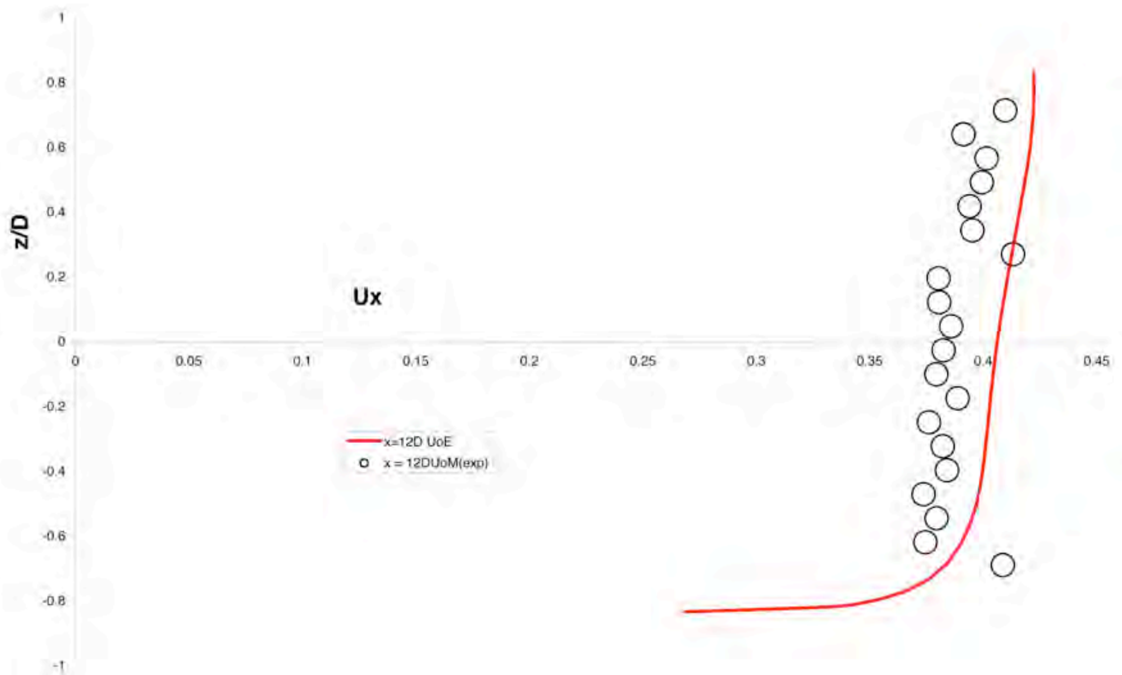

Figure 27 Averaged Ux distribution across the z cut plane at twelve diameters downstream of the turbine farm

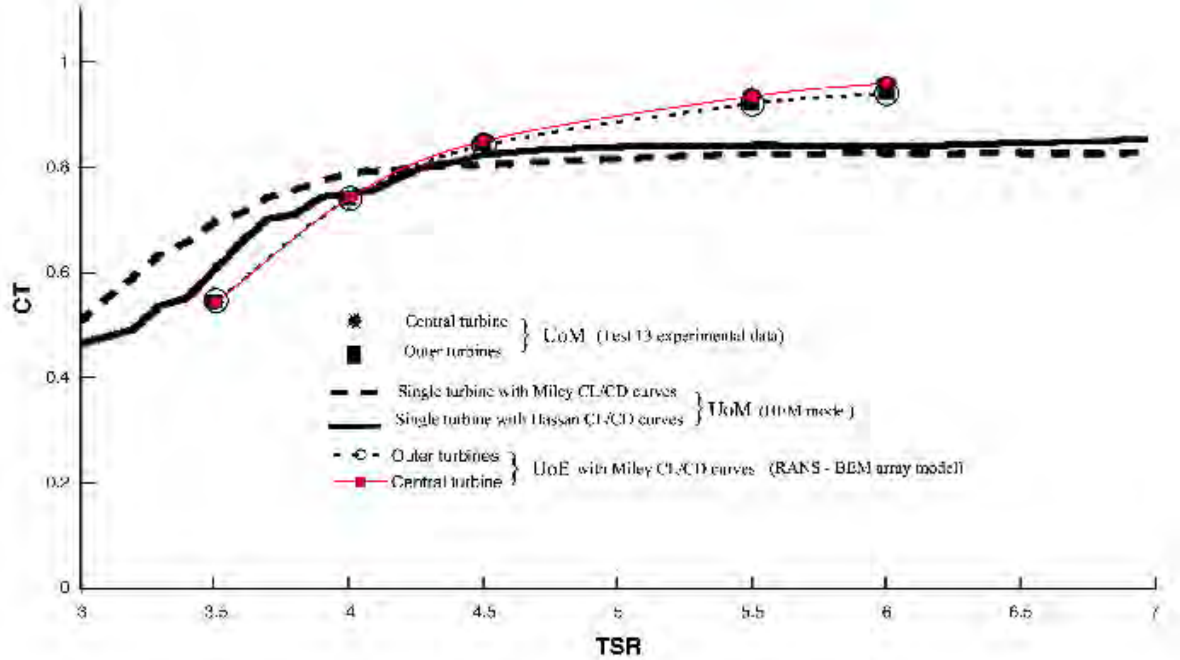

Figure 28 Comparison of experimental results and predicted thrust coefficient CT from each of the three turbines involved in test 13

4.2. Staggered array of turbines working under unsteady conditions representing the Sound of Islay

The results given in figures 29 to 33 give a time average results from the staggered farm simulation for the Sound of Islay (Section 3). In the present simulations, the turbine rotors have a constant rotational speed, equivalent to a TSR of 4.5. In a more realistic simulation (the subject of WG3 WP2 D7/D8) the speed of the rotor would be governed by the power extracted from the flow and the shaft torque, thus providing a more complete model of the power train.

General *Code Saturne* settings

This section details the general settings used within all the *Code Saturne* simulations, and explains the reasoning behind them. This should facilitate easy recreation of tidal channel simulations, even without the original XML files. Unless otherwise indicated, these settings are what the *Code Saturne* User Guide calls L1 (level 1) options: i.e. options that can be changed through the *Code Saturne* GUI. The *Code Saturne* option key is in brackets, which can be found in section 9 of the user guide if more details are required.

Table 1. List of physical parameters.

Name	Keyword	Value	Explanation
Density	*IROVAR*	1020 Kg/m^3	Density of seawater at 20ºC near surface.
Dynamic viscosity	*IVIVAR*	0.001 Na.s	Dynamic viscosity at 20ºC.

Table 2. Numerical parameters used within *Code Saturne* in every simulation.

Name	Keyword	Value	Explanation
Flow algorithm	*IDTVAR*	Unsteady	RANS unsteady state numerically stable; also D5b will use unsteady RANS, so more appropriate choice.
Turbulence model	*ITURB*	k-epsilon	k-ω turbulence modelling buggy in *Code Saturne* with rough walls: see WG3 WP2 D5a and WG3 WP2 D4 for further explanation.
Initial velocity	-	0 m/s	No initial profile assumed; allow channel velocity profile to develop through bottom drag.
Initial turbulence	-	$k=6.0\times10^{-4}\ m^2/s^2$ $\varepsilon=1.6\times10^{-4}\ m^2/s^2$	A small degree of initial turbulence was found to increase numerical stability: no effect on eventual levels of turbulence.
Unsteady flow algorithm management	-	Variable in time and uniform in space	Default values used for Max. CFL no., Max. Fourier no, etc. except NTMABS (see below).
Number of iterations	*NTMABS*	10000	Sufficient number of iterations for flow to fully develop and become statistically stable.
Equation parameters/ scheme (VelocityX, VelocityY, … Dissip)	*ISCHCV*	SOLU	Second-order upwind scheme found experimentally to give greater numerical stability.
Gradient calculation method	*IMRGRA*	Least sq. method over extended cell neighbourhood	Improves numerical stability with strong vertical velocity gradients for slight increase in CPU usage.
Output Control/Post-processing	*NTCHR*	Post-processing every 10 time steps	Gives sufficient time-stepped field data towards end of simulations to allow averages to be calculated. Total disc usage found to be ~30Gb (For every 1 time steps, this increases to 300Gb)
Number of parallel cores	*(see SCRIPTS/ runcase qsub script)*	120	Number of cores to run simulations under MPI. Simulations take approximately 24 hours.
Memory per core	*(see SCRIPTS/ runcase qsub script)*	4Gb	Minimum amount of memory available per core, setting the 4Gb limit ensures none of the allocated cores will have 2Gb of ram so the *Code Saturne* pre-processor can run without a memory fault.

Bed roughness

We assume that our channel model has a bottom roughness, which represents the effect of friction caused by an uneven, rocky layer on the seabed. This can be set within *Code Saturne* as part of a rough wall boundary condition, with the roughness prescribed as a roughness height, Z_0. In the present simulations, a bed roughness length of Z_0=0.2m has been used (WG3 WP2 D4 presents a justification for the selection of this roughness length).

Velocity profile and inlet turbulence

The velocity profile is set at the inlet as a Dirichlet condition. This takes the form of a standard logarithmic profile for turbulent flow, ie.

(1) $$u(z) = \frac{u_\tau}{\kappa}\ln\left(\frac{z}{z_0}\right)$$

Where $u(z)$ is the x-component of the water velocity at height z above the seabed, κ is the Von Karman constant ($= 0.41$), and u_τ is the friction velocity. The y and z components of velocity are zero.

It should be noted that we are neglecting the viscous sub-layer here, as we expect the flow to properly develop downstream in the CFD simulation, and so the boundary velocity profile must only qualitatively represent the flow overall. As a result, where $z < z_0$ we set $u = 0$.

To calculate u_τ, as we already know z_0, we must specify u at a known height at the boundary. A sensible choice would be at the presumed hub-height, z_H, of the tidal turbines to be modelled. If we say $u_H = u(z = z_H)$, then we can write the frictional velocity as

(2) $$u_\tau = u_H \kappa \left[\ln\left(\frac{z_H}{z_0} \right) \right]^{-1}$$

In the present simulations, $z_H = 40\,\mathrm{m}$. This has been selected for consistency with D4. The velocity log profile is set via the **`usclim.f90`** routine in *Code Saturne*.

Within the *Code Saturne* GUI, the turbulence at the inlet can be specified by two parameters: the turbulent intensity (TI), and the hydraulic diameter (D_H). There are several definitions of the hydraulic diameter, D_H; we shall use the most common definition, ie.

(3) $$D_H = \frac{A}{P},$$

where A is the cross-sectional area of the channel, and P is the wetted perimeter. As $P = width \times depth$, this gives us a hydraulic parameter of $D_H \approx 24\,\mathrm{m}$. The turbulence intensity (at the inlet) for these simulations has been selected as 15%, again for consistency with D4 (which provides a justification).

Initial conditions

The initial velocity in the channel is set to 0 ms^{-1}. The use of a quiescent initial condition allows the flow around the turbines to develop slowly and prevents the stability issues, which would be encountered by using a "big splash" initial condition where the initial flow field would have a constant uniform velocity. The disadvantage of this approach is that the incoming tide must be allowed to wash through the domain, and the initial transients allowed to propagate out of the domain before post processing can be done. Figures 29 to 31 show the developing velocity profile through the channel after 1, 200 and 400 time steps. The propagation of the velocity front can be clearly be seen.

Post processing

Once sufficient time steps have been performed for a quasi-steady solution to be obtained (circa 1400), the results can be post processed. In order to remove small scale fluctuations in the plots, the presented quantities are averaged over the last 200 iterations of the simulation. This provides plots of mean flow quantities in the same way as the results presented in D4. Additional post processing can also be performed to present other statistical quantities associated with the flow such as variance.

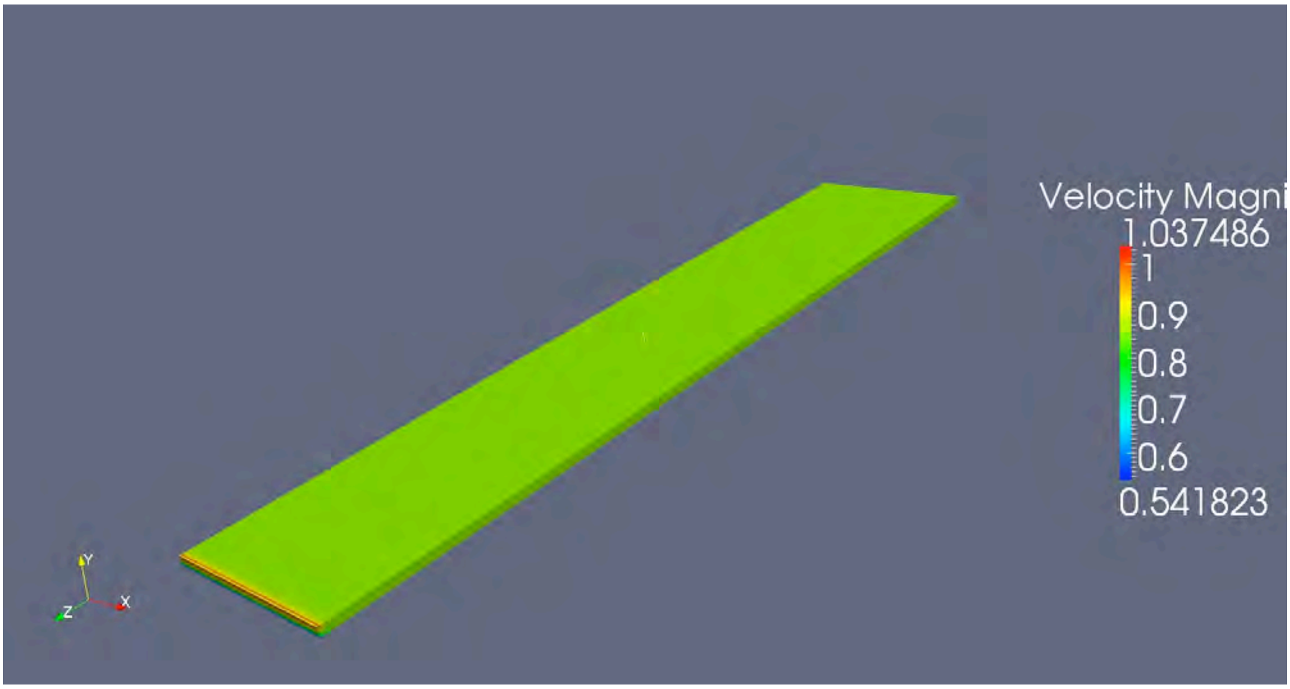

Figure 29 uH =1 first time step

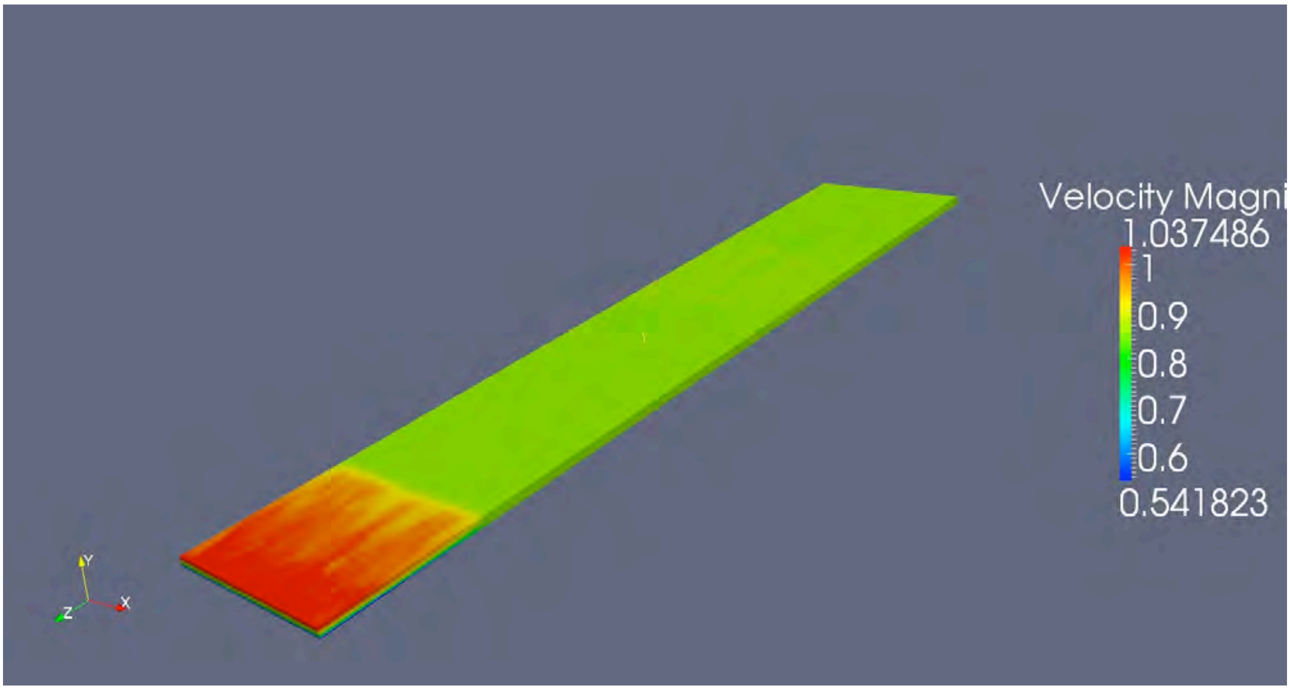

Figure 30 uH =1 time step 200

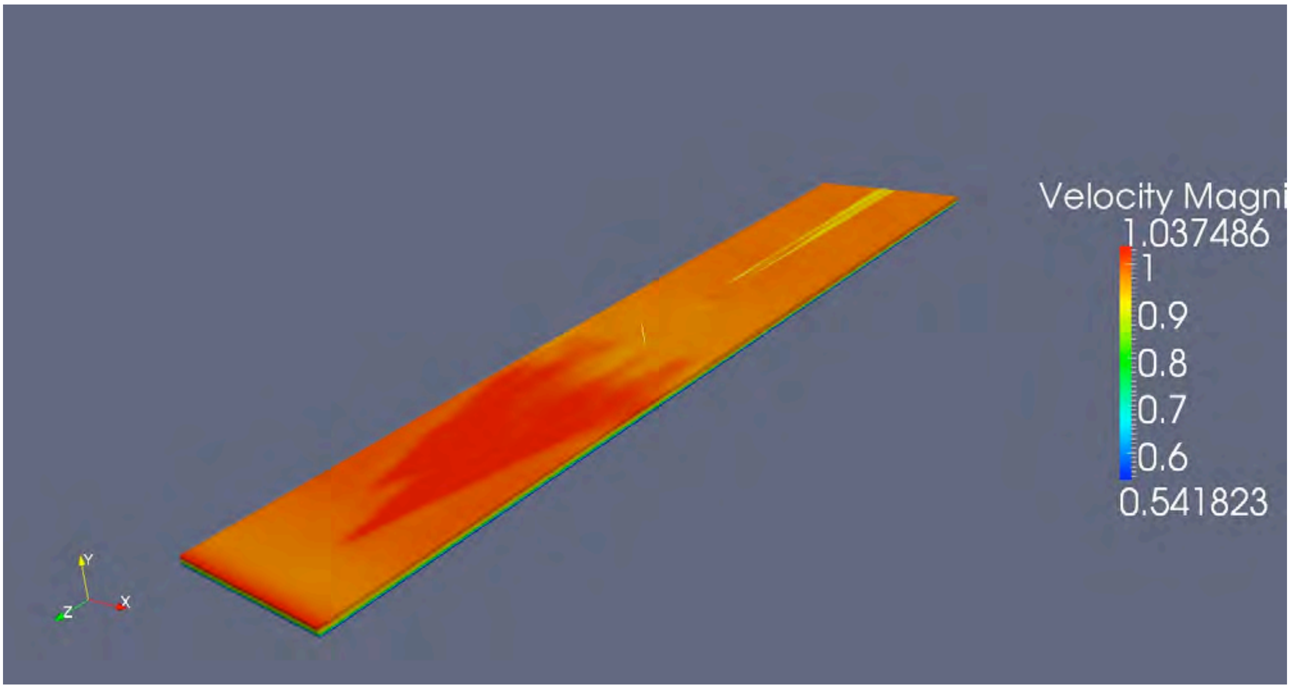

Figure 31 uH =1 time step 400

Figure 32 shows a psudo-colour plot of the velocity distribution through the centre line of the turbines for the low flow condition of uH=1.0ms^{-1}. The acceleration of the flow passing the array and the wakes of the first and second row of turbines can clearly be seen. The maximum velocity deficit is 25% and this occurs in the near wake region. It should be noted that the actual incident velocity at the hub height is 0.95ms^{-1}. This is slightly lower than the uH as the actual hub height is at 25m, whereas the reference height used for the boundary condition is at 40m. This reduction is realistic, since the flow velocities tabulated in tidal streaming atlases (and computed at tidal diamonds) are measured in the surface layer of the water column, whereas the turbine hub will be located in the top of the turbulent boundary layer.

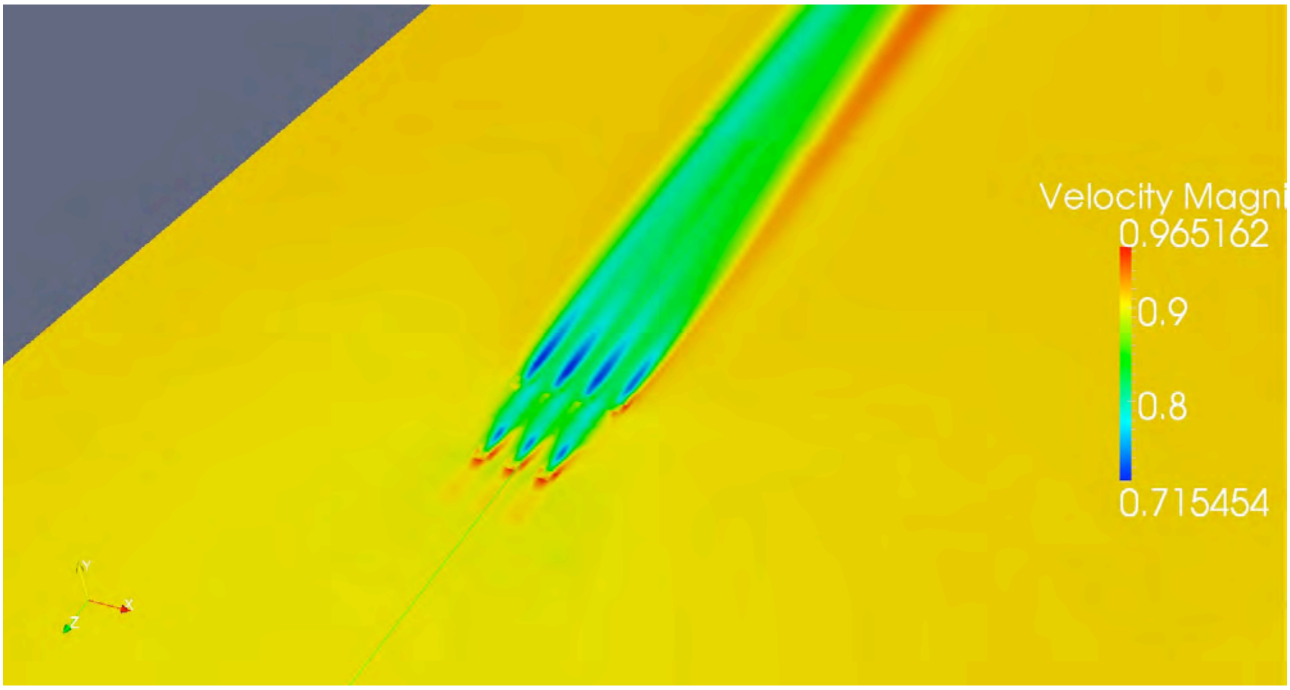

Figure 32 Y plane cut through flow field showing flow topography of the seven staggered turbines for uH=1m/s at 1436 time steps

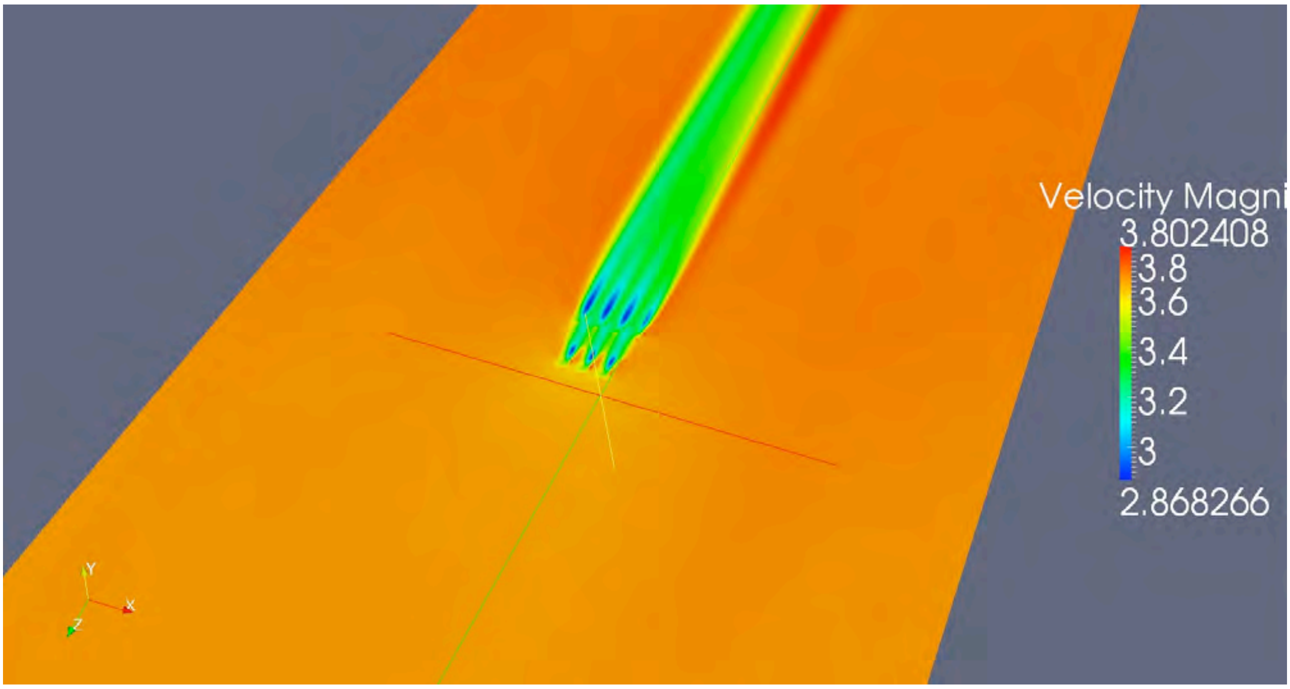

Figure 33 Y plane cut through flow field showing flow topography of the seven staggered turbines for uH=4m/s at 1436 time steps

Figure 33 shows a pseudo-colour plot of the velocity distribution through the centre line of the turbines for the high flow condition of uH=4.0ms^{-1}. The acceleration of the flow passing the array and the wakes of the first and second row of turbines can clearly be seen. The maximum velocity deficit in this case is 24% and this, again, occurs in the near wake region. As before, the incident velocity at the hub is slightly lower than the uH as the actual hub height is at 25m, whereas the reference height used for the boundary condition is at 40m. It is interesting to note that a small increase in the mean flow at the edges of the straight downstream of the turbine array can also be seen, indicating that e blockage is starting to have an effect.

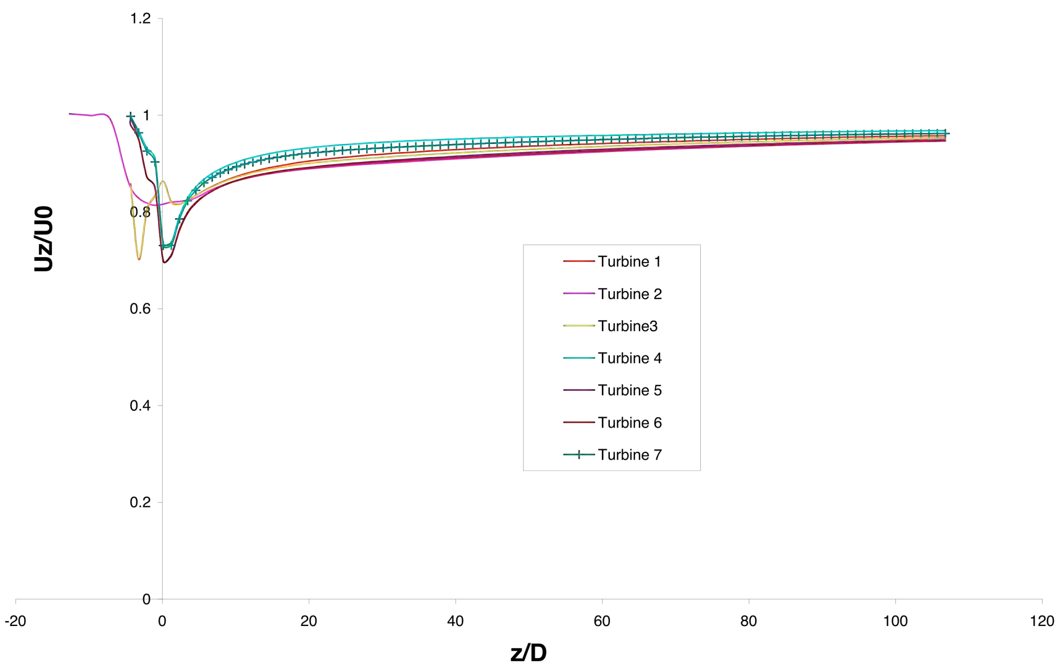

Figure 34 Axial normalised velocity distributions through the centrelines of the seven turbines for uH=4.0m/s.

Figure 34 shows the axial normalised velocity distributions through the centrelines of the seven turbines. Turbines 1, 2 and 3 are located in the upstream row of the array, while turbines 4 to 7 are located in the downstream row. The velocity deficit curves for turbines 1 and 3 (on the outside of the array) show a more pronounced deficit than turbine 2 immediately prior to the second row of turbines. As their wakes pass the 2nd row a slight acceleration of the flow can be seen, before the red (turbine 1) and yellow (turbine 3) curves merge with the distribution of turbine 2 (magenta). For the downstream row, the outer turbines (4 – cyan and 7 – cyan with crosses) show less velocity deficit than the two central turbines (5 – purple and 6 – brown). These differences persist far down stream of the array and are still evident 100 turbine diameters downstream, when the wakes have mixed almost completely. It is interesting to note that there is still a significant velocity deficit 120 diameters down stream at the downstream boundary of the domain.

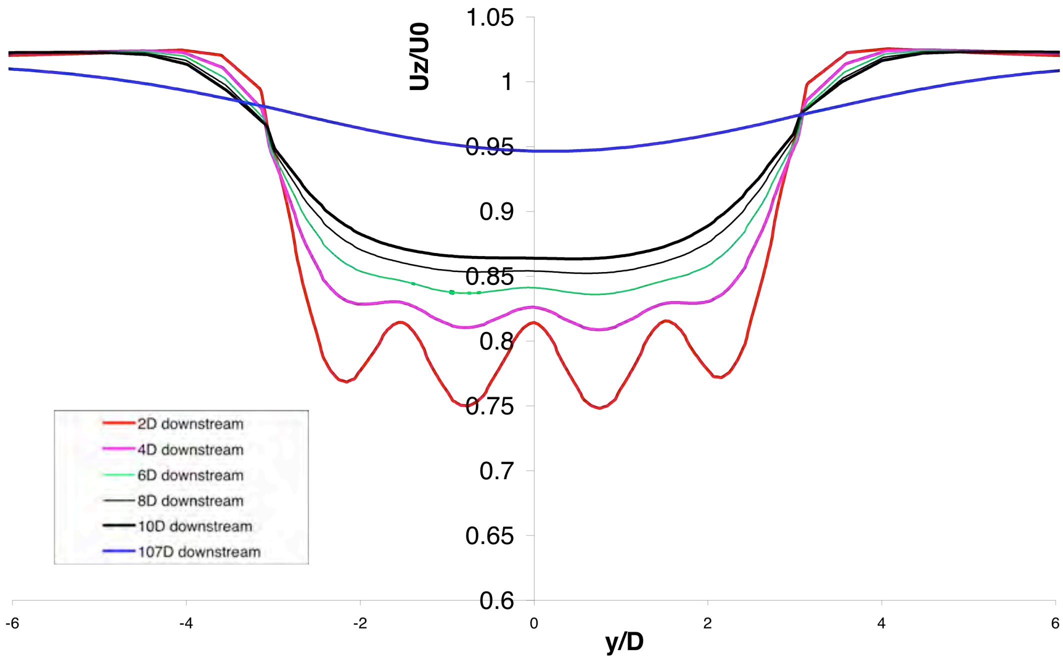

Figure 35 Normalised cross-stream velocity distributions at hub height for 2, 4, 6, 8, 10 and 107 diameters downstream of the array for uH=4.0ms^{-1}

Figure 35 shows the normalised velocity distributions across the width of the straight, at between 2 and 107 diameters downstream of the second row of turbines. The plot shows the difference in the velocity deficit between individual turbines in the near wake and that by six diameters downstream these have mixed to provide an almost uniform parabolic wake. By 107 diameters down stream of the array the array wake is still present with an approximately 5% maximum velocity deficit and a more classical Gaussian shape.

Figure 36 shows the vertical normalised velocity distribution on the centre line of the straight, down stream of the array. At 2 and 4 diameters down stream the velocity deficit due to the turbines is clearly visible, as is the acceleration of the flow passing beneath the array – this leads to a characteristic s-shaped velocity profile. By 12 diameters, the tidal boundary layer is being re-established and this acceleration region is no longer observed. By 107 diameters downstream, a classical power law boundary layer is observed, though the maximum normalised velocity is only about 95% of the free stream velocity.

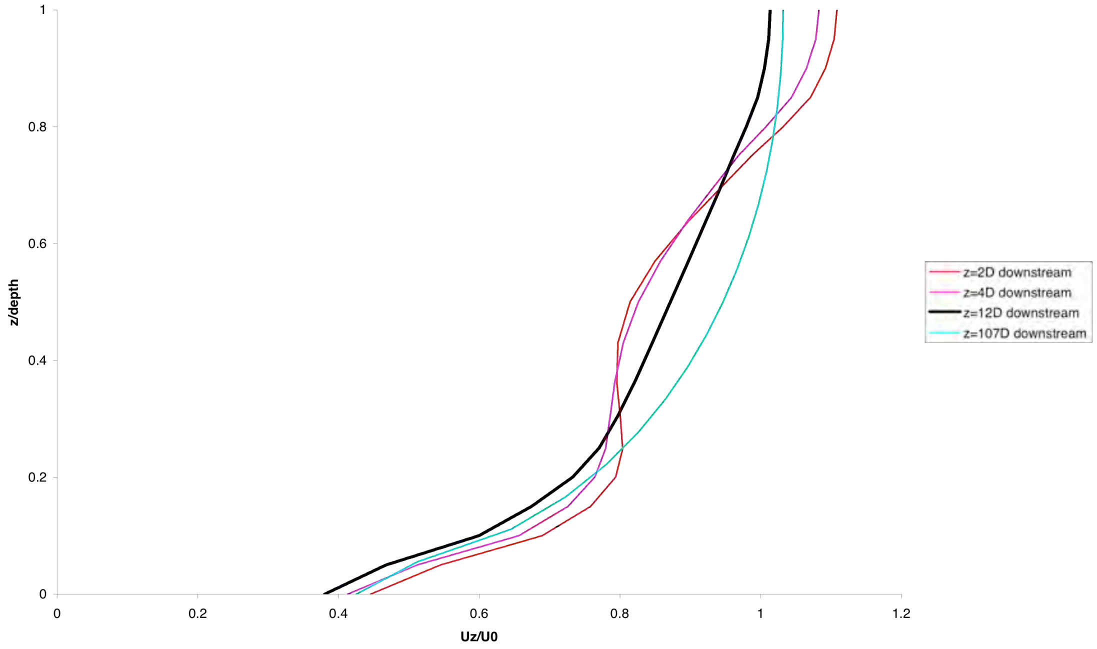

Figure 36 Vertical profile of normalised velocity, on the centre line of the channel, downstream of a staggered array of tidal turbines in a tidal straight at uH=4.0ms^{-1}

In performance of the turbines, Figure 37 shows the convergence history of the dimensionless thrust coefficient (C_T) for each of the seven turbines in the array. It can be seen that the upstream row of turbines operates at an effective C_T of 0.65, while there is a reduction in the thrust developed by the outer and inner turbines in the second row. The overall C_T and power (C_P) performance of the turbines at varying tip speed ratios is shown in Figure 38. This shows that the front row of the turbines is performing as expected.

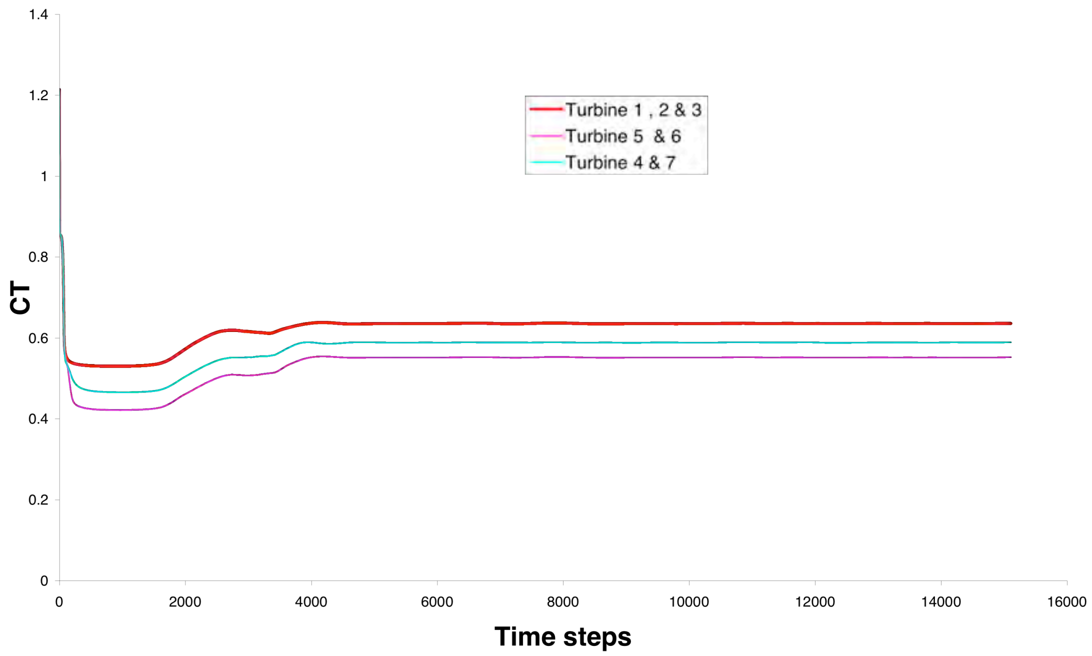

Figure 37 Dimensionless thrust coefficient for each of the turbines, plotted against iteration number for a staggered array of seven turbines operating at a TSR of 4.5 in a tidal channel with uH=4.0ms^{-1}

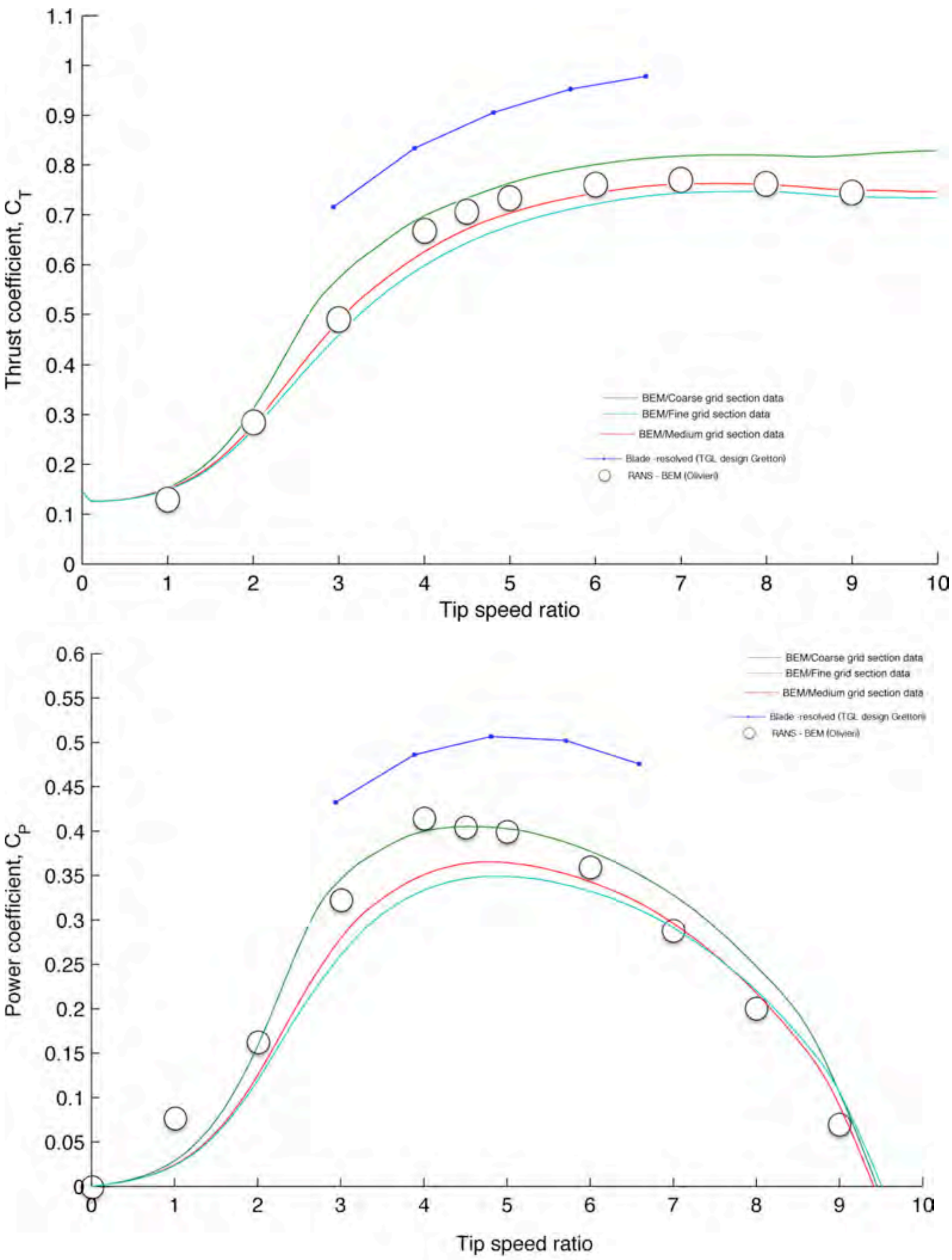

Figure 38 Thurst and Power coeficients as a function of tip speed ratio for the TGL style turbine rotor, showing BEMT-AD predictions, BEM predictions and blade resolved preditions.

5. Conclusions and further work

The work covered in this technical report has investigated the final aspects of D5a, which has been more extensively covered, to explain the minor differences seen in the wake between the theoretical analysis and the experimental results from the University of Manchester. This has used test case 13 to provide a less ambiguous test, compared to the other test cases given in D5a. The results provided in figure 28 are particularly encouraging, since agreement is extremely close.

The main test work of this deliverable has been displayed as a series of flow topologies, providing evidence of the applicability of the D5b model. The wake development is similar to that given for the staggered case. It has also displayed that the code and the Actuator Disc models are numerically stable for the unsteady flow array case and that the code is easily parallelized on distributed super-computers using MPI. In the present cases the code has been run on the EDDIE cluster at the University of Edinburgh over 120 cores, using a high speed myranet interconnect, leading to a speed up factor of more than 10 compared to a single processor. The implementation of the BEMT-AD model has been modified over that presented in D5a to allow a parallel implementation.

In terms of future deliverables two are of note: D3, which involving the full development of a free surface model will be integrated into to array model to examine deformation to the free surface passing the array (in D7), and D6 which will extend the wake parameterisation models from WG3 WP2 D5 for arrays of turbines. It should also be noted that demonstrating (in this deliverable) that the BEMT-AD model runs in parallel will allow very large scale simulations (such as those foreseen in WG3 WP2 D7 and D8) to be run utilising up to 1000 cores.

Appendix A

Contents of the accompanying CD

Appendix B

usinv.f90

```
!-------------------------------------------------------------------------------

!                      Code_Saturne version 2.0.0-rc1
!                      --------------------------

!     This file is part of the Code_Saturne Kernel, element of the
!     Code_Saturne CFD tool.

!     Copyright (C) 1998-2008 EDF S.A., France

!     contact: saturne-support@edf.fr

!     The Code_Saturne Kernel is free software; you can redistribute it
!     and/or modify it under the terms of the GNU General Public License
!     as published by the Free Software Foundation; either version 2 of
!     the License, or (at your option) any later version.

!     The Code_Saturne Kernel is distributed in the hope that it will be
!     useful, but WITHOUT ANY WARRANTY; without even the implied warranty
!     of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
!     GNU General Public License for more details.

!     You should have received a copy of the GNU General Public License
!     along with the Code_Saturne Kernel; if not, write to the
!     Free Software Foundation, Inc.,
!     51 Franklin St, Fifth Floor,
!     Boston, MA  02110-1301  USA

!-------------------------------------------------------------------------------

Module connectivity
  Integer(8)    :: nbcell(1000000,6)
  Integer(8)       con(1000000,6)
  double precision  clcd_val(1000)
contains
```

```
  subroutine error_message(text)

    character (*) :: text

  !if (irangp.le.0) then

    print*,text

  !endif

    stop

  end subroutine error_message

End Module connectivity


Module turbine_design

          integer     n_be

          Integer                            :: nMax_elms_start(8)

          Integer                            :: nMax_elms_stop(8)

          double precision                   :: rotor_radius

          double precision                   :: blade_scale

          double precision                   :: Frontal_Area(8)

          type :: b_section

             double precision                :: alpha

             double precision                :: cl

             double precision                :: cd

             type(b_section), pointer        :: next

          end type b_section

          type :: blade_element

             double precision                :: rad

             double precision                :: theta

             double precision                :: chord

             integer                         :: rec_count

             character(30),pointer           :: Bsection

             type (b_section), pointer       :: ptr_bsection

          end type blade_element

          type :: AD_element

             double precision                :: rad

             double precision                :: chord

             double precision                :: twist

             double precision                :: elm_area

             double precision                :: azim_deg

             double precision                :: dFX

             double precision                :: dFY

             double precision                :: dFZ

             double precision                :: frac_v
```

```
            integer                         :: nCel  ! Code Saturne cell number
            integer                         :: Nrec1
            integer                         :: Nrec2
            type (b_section), pointer       :: ptr_Belement1
            type (b_section), pointer       :: ptr_Belement2
         end type AD_element
         type(AD_element), allocatable      :: AD_position(:)
         type(blade_element), allocatable   :: blade_position(:)
End Module turbine_design



subroutine usiniv &
     !================

     ( idbia0 , idbra0 ,                                              &
     ndim   , ncelet , ncel   , nfac   , nfabor , nfml   , nprfml , &
     nnod   , lndfac , lndfbr , ncelbr ,                            &
     nvar   , nscal  , nphas  ,                                     &
     nideve , nrdeve , nituse , nrtuse ,                            &
     ifacel , ifabor , ifmfbr , ifmcel , iprfml , maxelt , lstelt , &
     ipnfac , nodfac , ipnfbr , nodfbr ,                            &
     idevel , ituser , ia     ,                                     &
     xyzcen , surfac , surfbo , cdgfac , cdgfbo , xyznod , volume , &
     dt     , rtp    , propce , propfa , propfb , coefa  , coefb  , &
     rdevel , rtuser , ra     )

  use turbine_design
  use connectivity
  !================================================================================
  ! Purpose:
  ! -------

  !    User subroutine.

  !    Initialize variables

  ! This subroutine is called at beginning of the computation
  ! (restart or not) before the loop time step
```

```
! This subroutine enables to initialize or modify (for restart)

!     unkown variables and time step values


! rom and viscl values are equal to ro0 and viscl0 or initialize

! by reading the restart file

! viscls and cp variables (when there are defined) have no value

! excepted if they are read from a restart file


! Physical quantities are defined in the following arrays:

!  propce (physical quantities defined at cell center),

!  propfa (physical quantities defined at interior face center),

!  propfa (physical quantities defined at border face center).

!

! Examples:

!  propce(iel, ipproc(irom  (iphas))) means rom  (iel, iphas)

!  propce(iel, ipproc(iviscl(iphas))) means viscl(iel, iphas)

!  propce(iel, ipproc(icp   (iphas))) means cp   (iel, iphas)

!  propce(iel, ipproc(ivisls(iscal))) means visls(iel, iscal)

!  propfa(ifac, ipprof(ifluma(ivar))) means flumas(ifac, ivar)

!  propfb(ifac, ipprob(irom (iphas))) means romb  (ifac, iphas)

!  propfb(ifac, ipprob(ifluma(ivar))) means flumab(ifac, ivar)


! Modification of the behaviour law of physical quantities (rom, viscl,

! viscls, cp) is not done here. It is the purpose of the user subroutine

! usphyv


! Cells identification

! ====================


! Cells may be identified using the 'getcel' subroutine.

! The syntax of this subroutine is described in the 'usclim' subroutine,

! but a more thorough description can be found in the user guide.



!-------------------------------------------------------------------------------

! Arguments

!__________________.____._____.________________________________________________.

! name             !type!mode ! role                                           !

!__________________!____!_____!________________________________________________!

! idbia0           ! i  ! <-- ! number of first free position in ia            !
```

```
! idbra0          ! i  ! <-- ! number of first free position in ra          !
! ndim            ! i  ! <-- ! spatial dimension                            !
! ncelet          ! i  ! <-- ! number of extended (real + ghost) cells      !
! ncel            ! i  ! <-- ! number of cells                              !
! nfac            ! i  ! <-- ! number of interior faces                     !
! nfabor          ! i  ! <-- ! number of boundary faces                     !
! nfml            ! i  ! <-- ! number of families (group classes)           !
! nprfml          ! i  ! <-- ! number of properties per family (group class) !
! nnod            ! i  ! <-- ! number of vertices                           !
! lndfac          ! i  ! <-- ! size of nodfac indexed array                 !
! lndfbr          ! i  ! <-- ! size of nodfbr indexed array                 !
! ncelbr          ! i  ! <-- ! number of cells with faces on boundary       !
! nvar            ! i  ! <-- ! total number of variables                    !
! nscal           ! i  ! <-- ! total number of scalars                      !
! nphas           ! i  ! <-- ! number of phases                             !
! nideve, nrdeve  ! i  ! <-- ! sizes of idevel and rdevel arrays            !
! nituse, nrtuse  ! i  ! <-- ! sizes of ituser and rtuser arrays            !
! ifacel(2, nfac) ! ia ! <-- ! interior faces -> cells connectivity         !
! ifabor(nfabor)  ! ia ! <-- ! boundary faces -> cells connectivity         !
! ifmfbr(nfabor)  ! ia ! <-- ! boundary face family numbers                 !
! ifmcel(ncelet)  ! ia ! <-- ! cell family numbers                          !
! iprfml          ! ia ! <-- ! property numbers per family                  !
!  (nfml, nprfml) !    !     !                                              !
! maxelt          ! i  ! <-- ! max number of cells and faces (int/boundary)  !
! lstelt(maxelt)  ! ia ! --- ! work array                                   !
! ipnfac(nfac+1)  ! ia ! <-- ! interior faces -> vertices index (optional)  !
! nodfac(lndfac)  ! ia ! <-- ! interior faces -> vertices list (optional)   !
! ipnfbr(nfabor+1) ! ia ! <-- ! boundary faces -> vertices index (optional) !
! nodfbr(lndfbr)  ! ia ! <-- ! boundary faces -> vertices list (optional)   !
! idevel(nideve)  ! ia ! <-> ! integer work array for temporary development  !
! ituser(nituse)  ! ia ! <-> ! user-reserved integer work array             !
! ia(*)           ! ia ! --- ! main integer work array                      !
! xyzcen          ! ra ! <-- ! cell centers                                 !
!  (ndim, ncelet) !    !     !                                              !
! surfac          ! ra ! <-- ! interior faces surface vectors               !
!  (ndim, nfac)   !    !     !                                              !
! surfbo          ! ra ! <-- ! boundary faces surface vectors               !
!  (ndim, nfabor) !    !     !                                              !
! cdgfac          ! ra ! <-- ! interior faces centers of gravity            !
```

```
!  (ndim, nfac)    !    !     !                                               !
! cdgfbo           ! ra ! <-- ! boundary faces centers of gravity              !
!  (ndim, nfabor)  !    !     !                                               !
! xyznod           ! ra ! <-- ! vertex coordinates (optional)                  !
!  (ndim, nnod)    !    !     !                                               !
! volume(ncelet)   ! ra ! <-- ! cell volumes                                   !
! dt(ncelet)       ! ra ! <-- ! time step (per cell)                           !
! rtp(ncelet, *)   ! ra ! <-- ! computed variables at cell centers at current  !
!                  !    !     ! time steps                                     !
! propce(ncelet, *)! ra ! <-- ! physical properties at cell centers            !
! propfa(nfac, *)  ! ra ! <-- ! physical properties at interior face centers   !
! propfb(nfabor, *)! ra ! <-- ! physical properties at boundary face centers   !
! coefa, coefb     ! ra ! <-- ! boundary conditions                            !
!  (nfabor, *)     !    !     !                                               !
! rdevel(nrdeve)   ! ra ! <-> ! real work array for temporary development      !
! rtuser(nrtuse)   ! ra ! <-> ! user-reserved real work array                  !
! ra(*)            ! ra ! --- ! main real work array                           !
!__________________!____!_____!________________________________________________!

!     Type: i (integer), r (real), s (string), a (array), l (logical),
!           and composite types (ex: ra real array)
!     mode: <-- input, --> output, <-> modifies data, --- work array
!===============================================================================

implicit none

!===============================================================================
! Common blocks
!===============================================================================

include "dimfbr.h"
include "paramx.h"
include "pointe.h"
include "numvar.h"
include "optcal.h"
include "cstphy.h"
include "cstnum.h"
include "entsor.h"
include "lagpar.h"
include "lagran.h"
```

```
include "parall.h"
include "period.h"
include "ppppar.h"
include "ppthch.h"
include "ppincl.h"
!===============================================================================

! Arguments

integer          ilelt, nlelt, nlelt2
integer          idbia0 , idbra0
integer          ndim   , ncelet , ncel   , nfac   , nfabor
integer          nfml   , nprfml
integer          nnod   , lndfac , lndfbr , ncelbr
integer          nvar   , nscal  , nphas
integer          nideve , nrdeve , nituse , nrtuse

integer          ifacel(2,nfac) , ifabor(nfabor)
integer          ifmfbr(nfabor) , ifmcel(ncelet)
integer          iprfml(nfml,nprfml), maxelt, lstelt(maxelt)
integer          ipnfac(nfac+1), nodfac(lndfac)
integer          ipnfbr(nfabor+1), nodfbr(lndfbr)
integer          idevel(nideve), ituser(nituse), ia(*)

double precision xyzcen(ndim,ncelet)
double precision surfac(ndim,nfac), surfbo(ndim,nfabor)
double precision cdgfac(ndim,nfac), cdgfbo(ndim,nfabor)
double precision xyznod(ndim,nnod), volume(ncelet)
double precision dt(ncelet), rtp(ncelet,*), propce(ncelet,*)
double precision propfa(nfac,*), propfb(nfabor,*)
double precision coefa(nfabor,*), coefb(nfabor,*)
double precision rdevel(nrdeve), rtuser(nrtuse), ra(*)

! Local variables

logical :: switch1,switch2
real    :: random
!real, parameter :: pi = 3.141592653589790
double precision, parameter :: RotorR   = 9.0  ! metres
```

```
  double precision  zmax, zmin

  double precision            angle_switch

  double precision  temprtp(ncelet,1)

  double precision x_dist, y_dist, z_dist, radius, disc_patch,area_sum

  integer          idebia, idebra,impout(6),ii

  integer          ielt,iel, iutile, iel1,iel2, i, j, ival,ival2

  integer          ifac, inb, inb2, itest, n_passes, ipass,n,nbe,Nrec,Nfarm

  integer(8)       ihuge

  integer,  parameter          :: Nfarmax    = 7  ! number of turbines modelled

  integer,  parameter          :: NbemCellmax = 10000 ! number of field cells involved in actuator
discs

  double precision   xTurb_centre(7) != (/-0.405,0.0,0.405/)

  double precision  chord1, chord2, twist1, twist2, frac_val1,frac_val,radius1,radius2, v_small

  double precision  deg2rad, dist_diff, cell_size

  Integer           ifac2, n_of_f, jmax,jmin,isnbb,Number_Of_Faces,ifbn,nbelm

  Logical           ifind, iswitch1,iswitch2

  type (b_section), pointer :: current, bucket

    !===============================================================================

  ! 1.  Initialization of local variables

  !===============================================================================

  idebia = idbia0

  idebra = idbra0

  ihuge  = 2.0e10

  v_small = 1.0d-6

  deg2rad = pi/180.0

  xTurb_centre(1) = 27.0

  xTurb_centre(2) = 0.0

  xTurb_centre(3) = -27.0

  xTurb_centre(4) = 40.5

  xTurb_centre(5) = 13.5

  xTurb_centre(6) = -13.5

  xTurb_centre(7) = -40.5

  rotor_radius = RotorR   ! metres

  do i = 1, ncel

     do j = 1, 6

        con(i,j) = -ihuge

        nbcell(i,j) = -ihuge

     enddo

  enddo
```

```
!*****************Find smallest cell size at inlet

cell_size = huge(1.0)

call getfbr('INLET', nlelt, lstelt)

nlelt2   = nlelt - 1

do ielt = 1, nlelt2

  ifac = lstelt(ielt)

  ifac2 =lstelt(ielt + 1)

  dist_diff = dabs( cdgfbo(2,ifac) - cdgfbo(2,ifac2) )

  if (cell_size > dist_diff) then

     cell_size =  dist_diff

  endif

enddo

! Global Minimum

if (irangp.ge.0) then

   call parmin(cell_size)

endif

if (irangp.le.0) then

  print*,'pi = ',pi

  print*,'smallest cell size = ',cell_size

endif

!*****************************************************************************


!===============================================================================

! 2. Unknown variables initialization:

!      ONLY done if there is no restart computation

!===============================================================================


! --- Example:  isca(1) is the variable number in RTP related to the first

!               user-defined scalar variable

!               rtp(iel,isca(1)) is the value of this variable in cell number

!               iel.


!**************************** isuite if block checking for restart**************

if (isuite.eq.0) then

 ! do ii = 1, 1


  !  impout(ii) = impusr(ii)


  !enddo

  !open(impout(1),file='test_result.dat')
```

```
  !================================================================================
  ! 3. Building connectivity between particular indexed cell in the flow field and
  !     its local neighbour cells. For a hexahedral cell this involves 6 neighbours
  !================================================================================
     do ifac = 1, nfac
        iel1 = ifacel(1, ifac)
        iel2 = ifacel(2,ifac)
        switch1 = .true.
        switch2 = .true.
        do i = 1, 6
           if ((switch1).and.(con(iel1,i).eq.(-ihuge) )) then
              con(iel1,i) = ifac
              switch1 = .false.
           endif
           if ((switch2).and.(con(iel2,i).eq.(-ihuge) )) then
              con(iel2,i) = ifac
              switch2 = .false.
           endif
        enddo
     enddo

       do i = 1, ncel
          rtp(iel, isca(1) ) = 0.0
      enddo
  !========================================
  !            Checking parallel code
  !========================================
     ii = ncel
     ! global sum
     if (irangp.ge.0) then
      call parcpt(ii)
     endif
     if (irangp.le.0) then
       print*,'total number of cells =',ii
     endif
  !===============================================
! ----------------------------------------------------------------------
! Now setup rotor blade design assocaited with all turbines of the farm
```

```
! -----------------------------------------------------------------------


  blade_scale = 1.0

   allocate(AD_position(NbemCellmax))

   allocate(blade_position(100))

   call read_blade_design(n_be)

   if (n_be == 0) then

     !if (irangp.ge.0) then

       print*,'stopping no blade design data'

     !endif

    stop

   endif

   if (irangp.le.0) then

     print*,'n_be =',n_be

   endif

  nbelm = 0


 !-----------------------------------------------------------------------------------

 !     Setup each of the Nfarm turbines associated with modelled tidal turbine farm

 !-----------------------------------------------------------------------------------

 do Nfarm = 1,Nfarmax

  nMax_elms_start(Nfarm) = nbelm + 1

  area_sum = 0.d0

  if (Nfarm == 1) then

   call getcel('X < 36.0 and X > 18.0 and Y > -9.0 and Y < 9.0 and Z > -5.891 and Z < 2.091',   &

               nlelt,lstelt)

   zmin = -5.891 ;zmax = 2.091

  else if (Nfarm == 2) then

   call getcel('X < 9.0 and X > -9.0 and Y > -9.0 and Y < 9.0 and Z > -5.891 and Z < 2.091',   &

               nlelt,lstelt)

   zmin = -5.891 ;zmax = 2.091

  else if (Nfarm == 3) then

   call getcel('X < -18.0 and X > -36.0 and Y > -9.0 and Y < 9.0 and Z > -5.891 and Z < 2.091',   &

               nlelt,lstelt)

   zmin = -5.891 ;zmax = 2.091

  else if (Nfarm == 4) then

   call getcel('X < 49.5 and X > 31.5 and Y > -9.0 and Y < 9.0 and Z > -77.891 and Z < -69.911',   &

               nlelt,lstelt)

   zmin = -77.891 ;zmax = -69.911

  else if (Nfarm == 5) then
```

```
   call getcel('X < 22.5 and X > -4.5 and Y > -9.0 and Y < 9.0 and Z > -77.891 and Z < -69.911',   &
               nlelt,lstelt)
   zmin = -77.891 ;zmax = -69.911
  else if (Nfarm == 6) then
   call getcel('X < -4.5 and X > -22.5 and Y > -9.0 and Y < 9.0 and Z > -77.891 and Z < -69.911',
&
               nlelt,lstelt)
   zmin = -77.891 ;zmax = -69.911
  else if (Nfarm == 7) then
   call getcel('X < -31.5 and X > -49.5 and Y > -9.0 and Y < 9.0 and Z > -77.891 and Z < -69.911',
&
               nlelt,lstelt)
   zmin = -77.891 ;zmax = -69.911
  endif
  do ilelt = 1, nlelt
     iel = lstelt(ilelt)
     x_dist = xyzcen(1,iel) - xTurb_centre(Nfarm)
     y_dist = xyzcen(2,iel)
     z_dist = xyzcen(3,iel)
     radius = dsqrt((x_dist * x_dist) + (y_dist * y_dist) )
     if ((radius <= rotor_radius).and.(z_dist > zmin).and.(z_dist < zmax)) then
       do i = 1, 6
        !*********************************
        if (con(iel,i) > -1) then
         disc_patch = surfac(3,con(iel,i))
         if (disc_patch > 0.d0) then
           area_sum = area_sum + disc_patch * 0.5
                            ! used as a maker in debugging
           rtp(iel, isca(1) ) = 10.0
         endif
        endif
        !******************************
       enddo
     endif
  enddo
  if (irangp.ge.0) then
    call parsom(area_sum)
  endif
  Frontal_Area(Nfarm) = area_sum
 if (irangp.le.0) then
```

```
    print*,'time zero   area_sum = ',Frontal_Area(Nfarm),'  Turbine number = ',Nfarm

 endif




! -----------------------------
! Now setup the Actuator Disc
! -----------------------------

  if (Nfarm == 1) then
   call getcel('X < 36.0 and X > 18.0 and Y > -9.0 and Y < 9.0 and Z > -5.891 and Z < 2.091',   &
                nlelt,lstelt)
   zmin = -5.891 ;zmax = 2.091
  else if (Nfarm == 2) then
   call getcel('X < 9.0 and X > -9.0 and Y > -9.0 and Y < 9.0 and Z > -5.891 and Z < 2.091',   &
                nlelt,lstelt)
   zmin = -5.891 ;zmax = 2.091
  else if (Nfarm == 3) then
   call getcel('X < -18.0 and X > -36.0 and Y > -9.0 and Y < 9.0 and Z > -5.891 and Z < 2.091',   &
                nlelt,lstelt)
   zmin = -5.891 ;zmax = 2.091
  else if (Nfarm == 4) then
   call getcel('X < 49.5 and X > 31.5 and Y > -9.0 and Y < 9.0 and Z > -77.891 and Z < -69.911',   &
                nlelt,lstelt)
   zmin = -77.891 ;zmax = -69.911
  else if (Nfarm == 5) then
   call getcel('X < 22.5 and X > -4.5 and Y > -9.0 and Y < 9.0 and Z > -77.891 and Z < -69.911',   &
                nlelt,lstelt)
   zmin = -77.891 ;zmax = -69.911
  else if (Nfarm == 6) then
   call getcel('X < -4.5 and X > -22.5 and Y > -9.0 and Y < 9.0 and Z > -77.891 and Z < -69.911',
&
                nlelt,lstelt)
   zmin = -77.891 ;zmax = -69.911
  else if (Nfarm == 7) then
   call getcel('X < -31.5 and X > -49.5 and Y > -9.0 and Y < 9.0 and Z > -77.891 and Z < -69.911',
&
                nlelt,lstelt)
   zmin = -77.891 ;zmax = -69.911
  endif
  do ilelt = 1, nlelt
```

```
iel = lstelt(ilelt)
x_dist = xyzcen(1,iel) - xTurb_centre(Nfarm)
y_dist = xyzcen(2,iel)
z_dist = xyzcen(3,iel)
radius = dsqrt((x_dist * x_dist) + (y_dist * y_dist) )
! --------------------------------------------------------------
! ~~~~~~~Create Actuactor Disc from avialable mesh cells~~~~~~~~
! --------------------------------------------------------------
if ((radius <= rotor_radius).and.(z_dist > zmin).and.(z_dist < zmax)) then
  nbelm = nbelm + 1
  AD_position(nbelm)%nCel = iel
  AD_position(nbelm)%rad = radius
  ! -----------------------------------------------------
  !  find frontal area of cell and azimuthal angles
  !     about the centre on the actuator disc
  ! ------------------------------------------------------
  area_sum = 0.d0
  do i = 1, 6
    if (con(iel,i) > -1) then
     disc_patch = surfac(3,con(iel,i))
     if (disc_patch > 0.d0) then
      area_sum = area_sum + disc_patch * 0.5
     endif
    endif
  enddo
  ! ****************Global sum ***************
  !if (irangp.ge.0) then
  !  parsom(area_sum)                                    !remove  ?
  !endif
  !******************************************
  AD_position(nbelm)%elm_area  = area_sum
  AD_position(nbelm)%azim_deg = datan2(y_dist,x_dist)
  AD_position(nbelm)%frac_v   = 0.0
  ! -------------------------------------------------------
  !  Calculate blade design chord and twist from blade
  !   design table
  ! ------------------------------------------------------
  ifind = .true.
  do n = 1, n_be - 1
```

```
radius1 = blade_scale * (blade_position(n)%rad)

radius2 = blade_scale * (blade_position(n + 1)%rad)

! ----- set up Actuator Disc elements----------------------

if ( (radius >= radius1) .and. (radius <=  radius2)) then

  ifind = .false.

  frac_val1 = radius2 - radius1

  if (dabs(frac_val1) < v_small) then

     call error_message("Error with blade design table")

  endif

  frac_val = (radius - radius1)/frac_val1

  AD_position(nbelm)%frac_v = frac_val

  chord1 = blade_scale * (blade_position(n)%chord)

  chord2 = blade_scale * (blade_position(n+1)%chord)

  AD_position(nbelm)%chord = chord1 + (chord2 - chord1) * frac_val

  twist1 = deg2rad * (blade_position(n)%theta)

  twist2 = deg2rad * (blade_position(n+1)%theta)

  AD_position(nbelm)%twist = twist1 + (twist2 - twist1) * frac_val

  ! ------------------------------------------

  nullify(AD_position(nbelm)%ptr_Belement1)

  allocate(AD_position(nbelm)%ptr_Belement1)

  AD_position(nbelm)%ptr_Belement1 = blade_position(n)%ptr_bsection

  Nrec = blade_position(n)%rec_count

  AD_position(nbelm)%Nrec1 = Nrec

  current => AD_position(nbelm)%ptr_Belement1

  bucket  => blade_position(n)%ptr_bsection

  do i = 1, Nrec

    current%alpha = deg2rad * bucket%alpha

    current%cl    = bucket%cl

    current%cd    = bucket%cd

    allocate(current%next)

    nullify( current%next%next )

    current => current%next

    bucket  => bucket%next

  enddo

  ! -------------------------------------

  nullify(AD_position(nbelm)%ptr_Belement2)

  allocate(AD_position(nbelm)%ptr_Belement2)

  AD_position(nbelm)%ptr_Belement2 = blade_position(n+1)%ptr_bsection

  Nrec = blade_position(n+1)%rec_count

  AD_position(nbelm)%Nrec2 = Nrec
```

```
          current => AD_position(nbelm)%ptr_Belement2
          bucket  => blade_position(n+1)%ptr_bsection
          do i = 1, Nrec
            current%alpha = deg2rad * bucket%alpha
            current%cl    = bucket%cl
            current%cd    = bucket%cd
            allocate(current%next)
            nullify( current%next%next )
            current => current%next
            bucket  => bucket%next
          enddo
          ! -------------------------------------
          exit
        else if (radius < radius1) then
            AD_position(nbelm)%chord = blade_scale * blade_position(1)%chord
            AD_position(nbelm)%twist = deg2rad * blade_position(1)%theta
            ! -----------------------------------------
            nullify(AD_position(nbelm)%ptr_Belement1)
            allocate(AD_position(nbelm)%ptr_Belement1)
            AD_position(nbelm)%ptr_Belement1 = blade_position(1)%ptr_bsection
            Nrec = blade_position(1)%rec_count
            AD_position(nbelm)%Nrec1 = Nrec
            current => AD_position(nbelm)%ptr_Belement1
            bucket  => blade_position(1)%ptr_bsection
            do i = 1, Nrec
             current%alpha = deg2rad * bucket%alpha
             current%cl    = bucket%cl
             current%cd    = bucket%cd
             allocate(current%next)
             nullify( current%next%next )
             current => current%next
             bucket  => bucket%next
            enddo
            ! -------------------------------------
            nullify(AD_position(nbelm)%ptr_Belement2)
            allocate(AD_position(nbelm)%ptr_Belement2)
            AD_position(nbelm)%ptr_Belement2 = blade_position(1)%ptr_bsection
            Nrec = blade_position(1)%rec_count
            AD_position(nbelm)%Nrec2 = Nrec
```

```
              current => AD_position(nbelm)%ptr_Belement2

              bucket  => blade_position(1)%ptr_bsection

              do i = 1, Nrec

               current%alpha = deg2rad * bucket%alpha

               current%cl    = bucket%cl

               current%cd    = bucket%cd

               allocate(current%next)

               nullify( current%next%next )

               current => current%next

               bucket  => bucket%next

              enddo

              ! -------------------------------------

          endif

         ! ----------------------------------------------------------

      enddo

  !   if (ifind) then

   !     print*, 'element ',nbelm,' at radius = ',AD_position(nbelm)%rad

   !  endif

    endif

    ! --------------------------------------------------------------

    ! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    ! --------------------------------------------------------------

 enddo

!----------------------------------------------------------------------------------

!  enddo --  Setup each of the Nfarm turbines associated with modelled tidal turbine farm

!----------------------------------------------------------------------------------

! ----start and  stop element no for each turbine

 nMax_elms_stop(Nfarm) = nbelm

!----------------------------------------------------------

!  print*,'*************************************'

!  print*,' Processor number = ',irangp

!  print*,' Turbine number of farm = ',Nfarm

!  print*,' nMax_elms start = ',nMax_elms_start(Nfarm)

!  print*,' nMax_elms stop = ',nMax_elms_stop(Nfarm)

 ! print*,'*************************************'

enddo


 ! --------------------------------------------------

 ! Close files at final time step

 ! --------------------------------------------------
```

```
    !if (irangp.le.0) then

      !do ii = 1, 1

      !  close(impout(ii))

     ! enddo

    !endif
  ! ------------------------------------------------
  ! Close files at final time step
  ! ------------------------------------------------
  !*********************************** isuite if block***************
  endif    ! *************end of if block of isuite if block
 if (irangp.le.0) then
   print*,'end of usinv'
 endif
return
contains
        subroutine read_blade_design(n)
         use turbine_design
         implicit none

         type(b_section), pointer :: current,previous
         double precision rad,theta,chord,dummy
         integer n, itd ,ic,isec
         character a_nam1 * 30
         character a_nam2 * 30
         rad = 1.0
         n   = 0
         open(unit=9,FILE='blade.txt',status='old')
         do while(rad > 0)
          read(9,*)rad,theta,chord,dummy,a_nam1,a_nam2
          if (rad > 0) then
           n = n + 1
           blade_position(n)%rad = rad
           blade_position(n)%theta = theta
           blade_position(n)%chord = chord
           dummy = dummy * 100.0
```

```
    if (mod(dummy,1.0) < 0.5) then
       itd = int(dummy)
    else
       itd = int(dummy) + 1
    endif
    allocate(blade_position(n)%Bsection)
    blade_position(n)%Bsection = a_nam1
    call read_blade_section(n,itd,blade_position(n))
   endif
  enddo
  close(unit=9)
  return
 end subroutine read_blade_design
  !==================================================
  subroutine read_blade_section(ibdpos,ithk, bld_ptr)
   use turbine_design
   implicit none
   type(blade_element)       :: bld_ptr
   type (b_section), pointer :: current
   double precision alpha,cl,cd
   integer     count, ithk_chd,ithk,ibld, ichks,ic2
   integer     new_cnt,ithknew,ibdpos
   character   nam_given *20
   character file_type *6
   character file_name * 30
   character file_name1 * 30
   character icheck * 2
   logical ex, swit
   integer, parameter :: no_of_bsecs = 2
   integer, parameter :: lift_first =  1 ! 1 is for yes otherwise no

   file_type = '.txt'
   swit      = .true.
   nam_given = bld_ptr%Bsection
   count = index(nam_given,' ') - 1
   file_name = nam_given(1:count) // file_type
   if ((ithk > 9).and.(ithk < 100)) then
     icheck    = nam_given(count-1:count)
     read(icheck,*) ithk_chd
     write(icheck,'(i2)')ithk_chd
```

```
  file_name = nam_given(1:count-2) &
     // icheck// file_type
else
  ithk_chd = ithk
  swit      = .false.
   inquire(FILE= file_name,exist= ex)
   if (ex.eqv..true.) then
     if (irangp.le.0) then             !new
      print*,file_name,' found'
     endif
   else
    print*,'Error no ',file_name,' file included'
    stop
   endif
endif
!~~~~~~~Select the necessary blade section files~~~~
if (swit) then
 do ichks = 1, no_of_bsecs
  ithknew = ithk_chd
  do while(ithknew > 9)
   inquire(FILE= file_name,exist= ex)
   if (ex.eqv..true.) then
     if (irangp.le.0) then               !new
      print*,file_name,' found'
     endif
     exit
   endif
   ithknew = ithknew - 1
   write(icheck,'(i2)')ithknew
   file_name = nam_given(1:count-2) &
     // icheck// file_type
  enddo
  if (ex .eqv..false.) then
   if (irangp.le.0) then                 !new
    print*,'warning ',file_name,ibdpos
   endif
  else
    exit
  endif
```

```
 enddo

endif
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 allocate(bld_ptr%ptr_bsection)
 if (ex) then
  !print*,'the file is ',file_name
  open(unit=10,FILE= file_name,status='old')
  current => bld_ptr%ptr_bsection
  alpha = 0.0
  count = 1
  do
    if ( lift_first == 1) then
       read(10,*)alpha,cl,cd
    else
       read(10,*)alpha,cd,cl
    endif
    if (alpha == -999) then
    count = count - 1
     exit
    endif
    current%alpha = alpha
    current%cl    = cl
    current%cd    = cd
    allocate( current%next )
    count = count + 1
    current%next%alpha = -999
    current%next%cl    = -999
    current%next%cd    = -999
    nullify( current%next%next )
    current => current%next
  enddo
  close(unit=10)
 else
    count = 0
    current => bld_ptr%ptr_bsection
    current%alpha = -999
    current%cl    = -999
    current%cd    = -999
    allocate( current%next )
    nullify( current%next%next )
```

```
          endif

          !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

          bld_ptr%rec_count = count

         return

        end subroutine read_blade_section

        !=====================================================

end subroutine usiniv
```

Appendix C

usclim.f90

```
!-------------------------------------------------------------------------------

subroutine usclim &
!================

 ( idbia0 , idbra0 ,                                             &
   ndim   , ncelet , ncel   , nfac   , nfabor , nfml   , nprfml , &
   nnod   , lndfac , lndfbr , ncelbr ,                           &
   nvar   , nscal  , nphas  ,                                    &
   nideve , nrdeve , nituse , nrtuse ,                           &
   ifacel , ifabor , ifmfbr , ifmcel , iprfml , maxelt , lstelt , &
   ipnfac , nodfac , ipnfbr , nodfbr ,                           &
   icodcl , itrifb , faceBC ,                                    &
   idevel , ituser , ia     ,                                    &
   xyzcen , surfac , surfbo , cdgfac , cdgfbo , xyznod , volume , &
   dt     , rtp    , rtpa   , propce , propfa , propfb ,          &
   coefa  , coefb  , rcodcl ,                                    &
   w1     , w2     , w3     , w4     , w5     , w6     , coefu  , &
   rdevel , rtuser , ra     )

!===============================================================================

implicit none

!===============================================================================
! Common blocks
!===============================================================================

include "paramx.h"
include "pointe.h"
include "numvar.h"
include "optcal.h"
include "cstphy.h"
include "cstnum.h"
include "entsor.h"
```

```
include "parall.h"

include "period.h"

include "ihmpre.h"


!===============================================================================

! Arguments


integer          idbia0 , idbra0

integer          ndim   , ncelet , ncel   , nfac   , nfabor

integer          nfml   , nprfml

integer          nnod   , lndfac , lndfbr , ncelbr

integer          nvar   , nscal  , nphas

integer          nideve , nrdeve , nituse , nrtuse


integer          ifacel(2,nfac) , ifabor(nfabor)

integer          ifmfbr(nfabor) , ifmcel(ncelet)

integer          iprfml(nfml,nprfml), maxelt, lstelt(maxelt)

integer          ipnfac(nfac+1), nodfac(lndfac)

integer          ipnfbr(nfabor+1), nodfbr(lndfbr)

integer          icodcl(nfabor,nvar)

integer          itrifb(nfabor,nphas), faceBC(nfabor,nphas)

integer          idevel(nideve), ituser(nituse), ia(*)


! I have _NO_ idea what all this is. Left until I work out what it all does.


double precision xyzcen(ndim,ncelet)

double precision surfac(ndim,nfac), surfbo(ndim,nfabor)

double precision cdgfac(ndim,nfac), cdgfbo(ndim,nfabor)

double precision xyznod(ndim,nnod), volume(ncelet)

double precision dt(ncelet), rtp(ncelet,*), rtpa(ncelet,*)

double precision propce(ncelet,*)

double precision propfa(nfac,*), propfb(nfabor,*)

double precision coefa(nfabor,*), coefb(nfabor,*)

double precision rcodcl(nfabor,nvar,3)

double precision w1(ncelet),w2(ncelet),w3(ncelet)

double precision w4(ncelet),w5(ncelet),w6(ncelet)

double precision coefu(nfabor,ndim)

double precision rdevel(nrdeve), rtuser(nrtuse), ra(*)


! Local variables
```

```
integer :: fileUnit
integer :: face, phase
integer :: numElems, n, m

real :: x, y, z, pressure, u, v, w, k, epsilon, omega
real :: dpfac, dufac, dvfac, dwfac, dkfac, depsfac, domgfac

real, parameter :: kappa=0.41, C_mu=0.09, uH=1.0, H=40, y0=0.2
character(*), parameter :: profileType="log"

real :: uFriction

! Variables specific to D.Olivieri's model
real :: yOlivieri
real, parameter :: hOlivieri=25


!===============================================================================



! -----------------------------------------------------------------------
! --- Specify and derive constants:

phase = 1

! Will probably use this later
! fileUnit = impusr(1)



! -----------------------------------------------------------------------
! --- Prescribe inlet boundary condition:



! call getfbr('INLET or BOTTOM or TOP', nlelt, lstelt)


! '1' is the inlet boundary
call getfbr('INLET', numElems, lstelt)
```

```
! Note that the list of elements is actually a list of faces

do n=1, numElems

   ! Get face index from list
   face = lstelt(n)

   ! Set boundary condition type. 'ientre' = inlet (Dirichlet, sort of).
   faceBC(face, phase) = ientre

   ! Get coordinates of the current face
   x = cdgfbo(1, face)

   ! David Olivieri's co-ordinate system is centred around the turbine hub.
   yOlivieri = cdgfbo(2, face)
   ! Transpose this to height above seabed
   y = yOlivieri + hOlivieri


   z = cdgfbo(3, face)

   u = 0
   v = 0
   w = 0
   k = 0
   epsilon = 0

   if(profileType=="linear") then
      w = (y/40) * uH

   elseif (profileType=="log") then

      uFriction = (uH * kappa) / log(H/y0)

      if(y>y0) then
         w = (uFriction/kappa) * log(y/y0)
      else
         w = 0
```

```
      end if



      ! k-eps turbulence model
      if(iturb(phase) == 20) then
         ! Calculate profiles for u, v, w, k and epsilon
         k = (uFriction**2.) / sqrt(C_mu)

         if(y>y0) then
            epsilon = (uFriction**3.) / (kappa * y)
         else
            epsilon = 0
         end if

!         rcodcl(face, ik(phase),  1)  = k
!         rcodcl(face, iep(phase), 1) = epsilon



      ! k-omega turbulence model - currently disabled
      elseif(iturb(phase) == 60) then
         print*, "error: can't use k-omega SST for rough wall boundary layers"
         stop

      end if
   else
      print*, "error: usclim.f90: unknown profile type"
      stop

   end if

   ! --------------------------------------------------
   ! Specify values according to turbulence model:

   rcodcl(face, iu(phase), 1) = u
   rcodcl(face, iv(phase), 1) = v
   rcodcl(face, iw(phase), 1) = -w



  ! --------------------------------------------------
```

```
enddo ! END loop through faces

!===============================================================================

end subroutine usclim
```

Appendix D

ustsns.f90

```
!-------------------------------------------------------------------------------

!                      Code_Saturne version 2.0.1
!                      --------------------------

!     This file is part of the Code_Saturne Kernel, element of the
!     Code_Saturne CFD tool.

!     Copyright (C) 1998-2009 EDF S.A., France

!     contact: saturne-support@edf.fr

!     The Code_Saturne Kernel is free software; you can redistribute it
!     and/or modify it under the terms of the GNU General Public License
!     as published by the Free Software Foundation; either version 2 of
!     the License, or (at your option) any later version.

!     The Code_Saturne Kernel is distributed in the hope that it will be
!     useful, but WITHOUT ANY WARRANTY; without even the implied warranty
!     of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
!     GNU General Public License for more details.

!     You should have received a copy of the GNU General Public License
!     along with the Code_Saturne Kernel; if not, write to the
!     Free Software Foundation, Inc.,
!     51 Franklin St, Fifth Floor,
!     Boston, MA  02110-1301  USA

!-------------------------------------------------------------------------------

subroutine ustsns &
!================

 ( idbia0 , idbra0 ,                                              &
   ndim   , ncelet , ncel   , nfac   , nfabor , nfml   , nprfml , &
```

```
   nnod   , lndfac , lndfbr , ncelbr ,                            &
   nvar   , nscal  , nphas  , ncepdp , ncesmp ,                   &
   nideve , nrdeve , nituse , nrtuse ,                            &
   ivar   , iphas  ,                                              &
   ifacel , ifabor , ifmfbr , ifmcel , iprfml , maxelt , lstelt , &
   ipnfac , nodfac , ipnfbr , nodfbr ,                            &
   icepdc , icetsm , itypsm ,                                     &
   idevel , ituser , ia     ,                                     &
   xyzcen , surfac , surfbo , cdgfac , cdgfbo , xyznod , volume , &
   dt     , rtpa   , propce , propfa , propfb ,                   &
   coefa  , coefb  , ckupdc , smacel ,                            &
   crvexp , crvimp ,                                              &
   dam    , xam    ,                                              &
   w1     , w2     , w3     , w4     , w5     , w6     ,          &
   rdevel , rtuser , ra     )

!===============================================================================
! Purpose:
! -------

!    User subroutine.

!    Additional right-hand side source terms for velocity components equation
!    (Navier-Stokes)

!
! Usage
! -----
! The routine is called for each velocity component. It is therefore necessary
! to test the value of the variable ivar to separate the treatments of the
! components iu(iphas), iv(iphas) or iw(iphas).
!
! The additional source term is decomposed into an explicit part (crvexp) and
! an implicit part (crvimp) that must be provided here.
! The resulting equation solved by the code for a velocity component u is:
!
!  rho*volume*du/dt + .... = crvimp*u + crvexp
!
! Note that crvexp and crvimp are defined after the Finite Volume integration
! over the cells, so they include the "volume" term. More precisely:
```

```
!   - crvexp is expressed in kg.m/s2
!   - crvimp is expressed in kg/s
!
! The crvexp and crvimp arrays are already initialized to 0 before entering the
! the routine. It is not needed to do it in the routine (waste of CPU time).
!
! For stability reasons, Code_Saturne will not add -crvimp directly to the
! diagonal of the matrix, but Max(-crvimp,0). This way, the crvimp term is
! treated implicitely only if it strengthens the diagonal of the matrix.
! However, when using the second-order in time scheme, this limitation cannot
! be done anymore and -crvimp is added directly. The user should therefore test
! the negativity of crvimp by himself.
!
! When using the second-order in time scheme, one should supply:
!   - crvexp at time n
!   - crvimp at time n+1/2
!
!
! The selection of cells where to apply the source terms is based on a getcel
! command. For more info on the syntax of the getcel command, refer to the
! user manual or to the comments on the similar command getfbr in the routine
! usclim.

!-------------------------------------------------------------------------------
! Arguments
!__________________.____._____.________________________________________________.
! name             !type!mode ! role                                           !
!__________________!____!_____!________________________________________________!
! idbia0           ! i  ! <-- ! number of first free position in ia            !
! idbra0           ! i  ! <-- ! number of first free position in ra            !
! ndim             ! i  ! <-- ! spatial dimension                              !
! ncelet           ! i  ! <-- ! number of extended (real + ghost) cells        !
! ncel             ! i  ! <-- ! number of cells                                !
! nfac             ! i  ! <-- ! number of interior faces                       !
! nfabor           ! i  ! <-- ! number of boundary faces                       !
! nfml             ! i  ! <-- ! number of families (group classes)             !
! nprfml           ! i  ! <-- ! number of properties per family (group class)  !
! nnod             ! i  ! <-- ! number of vertices                             !
! lndfac           ! i  ! <-- ! size of nodfac indexed array                   !
```

```
! lndfbr          ! i  ! <-- ! size of nodfbr indexed array                 !
! ncelbr          ! i  ! <-- ! number of cells with faces on boundary       !
! nvar            ! i  ! <-- ! total number of variables                    !
! nscal           ! i  ! <-- ! total number of scalars                      !
! nphas           ! i  ! <-- ! number of phases                             !
! ncepdp          ! i  ! <-- ! number of cells with head loss terms         !
! ncssmp          ! i  ! <-- ! number of cells with mass source terms       !
! nideve, nrdeve  ! i  ! <-- ! sizes of idevel and rdevel arrays            !
! nituse, nrtuse  ! i  ! <-- ! sizes of ituser and rtuser arrays            !
! ivar            ! i  ! <-- ! index number of the current variable         !
! iphas           ! i  ! <-- ! index number of the current phase            !
! ifacel(2, nfac) ! ia ! <-- ! interior faces -> cells connectivity         !
! ifabor(nfabor)  ! ia ! <-- ! boundary faces -> cells connectivity         !
! ifmfbr(nfabor)  ! ia ! <-- ! boundary face family numbers                 !
! ifmcel(ncelet)  ! ia ! <-- ! cell family numbers                          !
! iprfml          ! ia ! <-- ! property numbers per family                  !
!  (nfml, nprfml) !    !     !                                              !
! maxelt          ! i  ! <-- ! max number of cells and faces (int/boundary)   !
! lstelt(maxelt)  ! ia ! --- ! work array                                   !
! ipnfac(nfac+1)  ! ia ! <-- ! interior faces -> vertices index (optional)    !
! nodfac(lndfac)  ! ia ! <-- ! interior faces -> vertices list (optional)     !
! ipnfbr(nfabor+1) ! ia ! <-- ! boundary faces -> vertices index (optional)    !
! nodfbr(lndfbr)  ! ia ! <-- ! boundary faces -> vertices list (optional)     !
! icepdc(ncepdp)  ! ia ! <-- ! index number of cells with head loss terms     !
! icetsm(ncesmp)  ! ia ! <-- ! index number of cells with mass source terms   !
! itypsm          ! ia ! <-- ! type of mass source term for each variable     !
!  (ncesmp,nvar)  !    !     !  (see ustsma)                                !
! idevel(nideve)  ! ia ! <-- ! integer work array for temporary developpement !
! ituser(nituse   ! ia ! <-- ! user-reserved integer work array             !
! ia(*)           ! ia ! --- ! main integer work array                      !
! xyzcen          ! ra ! <-- ! cell centers                                 !
!  (ndim, ncelet) !    !     !                                              !
! surfac          ! ra ! <-- ! interior faces surface vectors               !
!  (ndim, nfac)   !    !     !                                              !
! surfbo          ! ra ! <-- ! boundary faces surface vectors               !
!  (ndim, nfavor) !    !     !                                              !
! cdgfac          ! ra ! <-- ! interior faces centers of gravity            !
!  (ndim, nfac)   !    !     !                                              !
! cdgfbo          ! ra ! <-- ! boundary faces centers of gravity            !
!  (ndim, nfabor) !    !     !                                              !
```

```
! xyznod           ! ra ! <-- ! vertex coordinates (optional)                  !
!  (ndim, nnod)    !    !     !                                                !
! volume(ncelet)   ! ra ! <-- ! cell volumes                                   !
! dt(ncelet)       ! ra ! <-- ! time step (per cell)                           !
! rtpa             ! ra ! <-- ! calculated variables at cell centers           !
!  (ncelet, *)     !    !     !  (preceding time steps)                        !
! propce(ncelet, *)! ra ! <-- ! physical properties at cell centers            !
! propfa(nfac, *)  ! ra ! <-- ! physical properties at interior face centers   !
! propfb(nfabor, *)! ra ! <-- ! physical properties at boundary face centers   !
! coefa, coefb     ! ra ! <-- ! boundary conditions                            !
!  (nfabor, *)     !    !     !                                                !
! ckupdc(ncepdp,6) ! ra ! <-- ! head loss coefficient                          !
! smacel           ! ra ! <-- ! value associated to each variable in the mass  !
!  (ncesmp,nvar)   !    !     !  source terms or mass rate (see ustsma)        !
! crvexp           ! ra ! --> ! explicit part of the source term               !
! crvimp           ! ra ! --> ! implicit part of the source term               !
! dam(ncelet)      ! ra ! --- ! work array                                     !
! xam(nfac,2)      ! ra ! --- ! work array                                     !
! w1,2,3,4,5,6     ! ra ! --- ! work arrays                                    !
!  (ncelet)        !    !     !  (computation of pressure gradient)            !
! rdevel(nrdeve)   ! ra ! <-> ! real work array for temporary developpement    !
! rtuser(nituse    ! ra ! <-- ! user-reserved real work array                  !
! ra(*)            ! ra ! --- ! main real work array                           !
!__________________!____!_____!________________________________________________!

!     Type: i (integer), r (real), s (string), a (array), l (logical),
!           and composite types (ex: ra real array)
!     mode: <-- input, --> output, <-> modifies data, --- work array
!===============================================================================
  use turbine_design
  use connectivity
  implicit none

!===============================================================================
! Common blocks
!===============================================================================

include "dimfbr.h"
include "paramx.h"
```

```
include "pointe.h"

include "numvar.h"

include "entsor.h"

include "optcal.h"

include "cstphy.h"

include "cstnum.h"

include "lagpar.h"

include "lagran.h"

include "parall.h"

include "period.h"

include "ppppar.h"

include "ppthch.h"

include "ppincl.h"


!===============================================================================


! Arguments


integer          idbia0 , idbra0

integer          ndim   , ncelet , ncel   , nfac   , nfabor

integer          nfml   , nprfml

integer          nnod   , lndfac , lndfbr , ncelbr

integer          nvar   , nscal  , nphas

integer          ncepdp , ncesmp

integer          nideve , nrdeve , nituse , nrtuse

integer          ivar   , iphas


integer          ifacel(2,nfac) , ifabor(nfabor)

integer          ifmfbr(nfabor) , ifmcel(ncelet)

integer          iprfml(nfml,nprfml), maxelt, lstelt(maxelt)

integer          ipnfac(nfac+1), nodfac(lndfac)

integer          ipnfbr(nfabor+1), nodfbr(lndfbr)

integer          icepdc(ncepdp)

integer          icetsm(ncesmp), itypsm(ncesmp,nvar)

integer          idevel(nideve), ituser(nituse), ia(*)


double precision xyzcen(ndim,ncelet)

double precision surfac(ndim,nfac), surfbo(ndim,nfabor)

double precision cdgfac(ndim,nfac), cdgfbo(ndim,nfabor)

double precision xyznod(ndim,nnod), volume(ncelet)
```

```
double precision dt(ncelet), rtpa(ncelet,*)
double precision propce(ncelet,*)
double precision propfa(nfac,*), propfb(nfabor,*)
double precision coefa(nfabor,*), coefb(nfabor,*)
double precision ckupdc(ncepdp,6), smacel(ncesmp,nvar)
double precision crvexp(ncelet), crvimp(ncelet)
double precision dam(ncelet ),xam(nfac ,2)
double precision w1(ncelet),w2(ncelet),w3(ncelet)
double precision w4(ncelet),w5(ncelet),w6(ncelet)
double precision rdevel(nrdeve), rtuser(nrtuse), ra(*)


! Local variables


character*80     chaine
integer          idebia, idebra
integer          iel, ipcrom, ipp, iutile, ifac,i,Nfarm
integer          ilelt, nlelt,iel1,iel2, nold_tstep,Nrec,ii,impout(10)
integer         :: ncall = 0                                                  ! fortran static
variable
double precision ckp, qdm, x_dist, y_dist, radius, disc_patch, z_dist
double precision F_prandtl, tval1,tval2, density, chord, frac_val, elm_area, azim_ang
double precision ux, uy, uz, u_theta, phi, cl, cd, cl1, cl2,cd1
double precision cd2,alpha,alpha1,density_Av
double precision  rad2deg,val_d, W_rel2, W_rel
double precision Cn_2D, Ct_2D, Cn_3D, Ct_3D, dF_theta, dF_z, dF_cel_theta, dF_cel_z, k_value
double precision accm_T,accm_P, C_T,C_p
!real, parameter :: pi = 3.141592653589740
double precision, parameter  :: U_inf = 1.0 ! m/s
double precision,  parameter :: omega = 0.5  ! rotational speed of turbine in radians per second
integer,  parameter          :: Nb    = 3  ! number of blades with the turbine
integer,  parameter          :: Nfarmax    = 7  ! number of turbines modelled
Logical, parameter           :: iflow      = .true.  ! clockwise rotation from upstream & axial flow
in negative z axis direction
type (b_section), pointer :: ptr_bsection
Logical           ichk



! blade design variables
integer n, nbelm
type(b_section), pointer :: current,previous,check
```

```
!===============================================================================

  ! -- setup output files --
    do ii = 1,Nfarmax + 1

      impout(ii) = impusr(ii)

    enddo
  if (irangp.le.0) then
    open(impout(1),file='CT_Cp_results1.dat')
    open(impout(2),file='CT_Cp_results2.dat')
    open(impout(3),file='CT_Cp_results3.dat')
    open(impout(4),file='CT_Cp_results4.dat')
    open(impout(5),file='CT_Cp_results5.dat')
    open(impout(6),file='CT_Cp_results6.dat')
    open(impout(7),file='CT_Cp_results7.dat')
    open(impout(8),file='Clcdmap.dat')
  endif
  ! ------------------------

!===============================================================================
! 1. Initialization
!===============================================================================

idebia = idbia0
idebra = idbra0
ncall = ncall + 1
rad2deg = 180.0/pi
ipp    = ipprtp(ivar)
iphas  = 1
if(iwarni(ivar).ge.1) then
  chaine = nomvar(ipp)
  write(nfecra,1000) chaine(1:8)
endif

ipcrom = ipproc(irom  (iphas))
!================================================================================
! 2. Calculating current momentum blade element forces
!================================================================================
```

```
if ( ncall == 1) then
 do Nfarm = 1, Nfarmax
     accm_T = 0.0
     accm_P = 0.0
  ! ******** nbelm loop *******************
if (nMax_elms_stop(Nfarm) > 1) then
  do nbelm = nMax_elms_start(Nfarm), nMax_elms_stop(Nfarm)
     iel      = AD_position(nbelm)%nCel
     x_dist   = xyzcen(1,iel)
     y_dist   = xyzcen(2,iel)
     z_dist   = xyzcen(3,iel)
     density  = propce(iel,ipcrom)
     radius   = AD_position(nbelm)%rad
     chord    = AD_position(nbelm)%chord
     frac_val = AD_position(nbelm)%frac_v
     elm_area = AD_position(nbelm)%elm_area
     azim_ang = AD_position(nbelm)%azim_deg
     ! -------------------------------------------------------
     !  Calculate blade element forces in blade fixed coordinates
     ! ------------------------------------------------------


        ! -----------------------------------------
        ! Note   ux,uy,uz are in flow fixed coords
        ! -----------------------------------------
     ux       = rtpa(iel, iu(iphas) )
     uy       = rtpa(iel, iv(iphas) )
     uz       = rtpa(iel, iw(iphas) )
        ! ----------------------------------------------------------
        ! Note  u_theta is equiv to uy in in blade fixed coordinates
        ! ----------------------------------------------------------
    if (iflow) then
     uz  = -uz
    endif
     u_theta  = - (uy * dcos(azim_ang)) + (ux * dsin(azim_ang) ) + (omega * radius)
     phi      = datan2(uz, u_theta)
     alpha    = phi - AD_position(nbelm)%twist
     Nrec     = AD_position(nbelm)%Nrec1
     if (Nrec == 0) then
```

```
 print*,'Nrec = ',Nrec
 print*,'radius = ',radius,' twist = ',AD_position(nbelm)%twist, &
      'phi = ',phi,' nbelm = ',nbelm
 stop
endif
ptr_bsection => AD_position(nbelm)%ptr_Belement1
call find_cl_cd(alpha,Nrec,ptr_bsection,cl1,cd1,ichk)
Nrec     = AD_position(nbelm)%Nrec2
if (Nrec == 0) then
 print*,'Nrec = ',Nrec
 print*,'radius = ',radius,' twist = ',AD_position(nbelm)%twist, &
      'phi = ',phi,' nbelm = ',nbelm
 stop
endif
ptr_bsection => AD_position(nbelm)%ptr_Belement2
call find_cl_cd(alpha,Nrec,ptr_bsection,cl2,cd2,ichk)
cl = cl1 + frac_val * (cl2 - cl1)
cd = cd1 + frac_val * (cd2 - cd1)
!************ recoding L/D values on the disc *******
if (cd > 0.0) then
  clcd_val(nbelm) = cl/cd
else
  clcd_val(nbelm) = 0.0
endif
 !***************************************************
Cn_2D = cl * dcos(phi) + cd * dsin(phi)
Ct_2D = cl * dsin(phi) - cd * dcos(phi)
 ! -----------------------------------------------------------
 !  Calculating the Prandtl Tip Loss factor F_prandtl
 ! ------------------------------------------------------------
if (dabs(radius) < tiny(1.0)) call error_message("error1 ustsns function")
tval1 = Nb * (1.0 - rotor_radius/radius)
tval2 = 2.0 * dsin(phi)
if (dabs(tval2) > tiny(1.0)) then
  F_prandtl = (2.0/pi) * dacos(dexp(tval1/tval2))
  if ((F_prandtl < 0).or.(F_prandtl > 1.0)) call error_message("error2 ustsns function")
else
 F_prandtl = 1.0
endif
Cn_3D    = F_prandtl * Cn_2D
```

```
    Ct_3D    = F_prandtl * Ct_2D
     ! -------------------------------------------------------------------------
     !  Note  W_rel2   =  (uz * uz) + (u_theta * u_theta) has not been included here
     !    since source terms crimpi & crvexpi require k * W_rel2 format for forces
     ! ------------------------------------------------------------------------

    dF_theta     = 0.5 * chord  * density * Ct_3D
    dF_z         = 0.5 * chord  * density * Cn_3D
    dF_cel_theta = Nb * dF_theta * elm_area/(2.0 * pi * radius)
    dF_cel_z     = Nb * dF_z     * elm_area/(2.0 * pi * radius)
    ! -------------------------------------------------------
    !  Convert blade element forces into flow fixed coordinates
    ! ------------------------------------------------------
    AD_position(nbelm)%dFX = - dF_cel_theta * dsin(azim_ang)
    AD_position(nbelm)%dFY =   dF_cel_theta * dcos(azim_ang)
    if (iflow) then
      AD_position(nbelm)%dFZ = dF_cel_z
    else
      AD_position(nbelm)%dFZ = - dF_cel_z
    endif
    W_rel2 = (u_theta * u_theta) + (uz * uz)
    accm_T = accm_T +  ( dF_cel_z * W_rel2 )/density
    accm_P = accm_P +  ( radius * omega * dF_cel_theta * W_rel2 )/density
  enddo
  ! ******** nbelm loop ********************
endif
    if (irangp.ge.0) then
      call parsom(accm_T)
      call parsom(accm_P)
    endif
    density_Av = 1.0
   if (irangp.le.0) then
    C_T = accm_T/(0.5 * density_Av * Frontal_Area(Nfarm)  * U_inf * U_inf)
    C_p = accm_P/(0.5 * density_Av * Frontal_Area(Nfarm) * U_inf * U_inf * U_inf)
    if (Nfarm == 1) then
      write(impout(1),"(2i5,2g17.9)") ntcabs,Nfarm,C_T,C_p
    else if (Nfarm == 2) then
     write(impout(2),"(2i5,2g17.9)") ntcabs,Nfarm,C_T,C_p
    else if (Nfarm == 3) then
```

```
      write(impout(3),"(2i5,2g17.9)") ntcabs,Nfarm,C_T,C_p
     else if (Nfarm == 4) then
      write(impout(4),"(2i5,2g17.9)") ntcabs,Nfarm,C_T,C_p
     else if (Nfarm == 5) then
      write(impout(5),"(2i5,2g17.9)") ntcabs,Nfarm,C_T,C_p
     else if (Nfarm == 6) then
      write(impout(6),"(2i5,2g17.9)") ntcabs,Nfarm,C_T,C_p
     else if (Nfarm == 7) then
      write(impout(7),"(2i5,2g17.9)") ntcabs,Nfarm,C_T,C_p
     endif
    endif

 enddo ! end of loop for Nfarm
endif



!===============================================================================
! 2. Example of arbitrary source term for component u:

!                              S = A * u + B

!            appearing in the equation under the form:

!                         rho*du/dt = S (+ standard Navier-Stokes terms)



!In the following example:
!  A = -rho*CKP
!  B =  XMMT
!
!with:
!  CKP = 1.D0 [1/s       ] (return term on velocity)
!  MMT = 100.D0 [kg/m2/s2] (momentum production by volume and time unit)
!
!which yields:
!     crvimp(iel) = volume(iel)* A = - volume(iel)*(rho*CKP )
!     crvexp(iel) = volume(iel)* B =   volume(iel)*(XMMT    )

! ----------------------------------------------
```

```
! It is quite frequent to forget to remove this example when it is
!  not needed. Therefore the following test is designed to prevent
!  any bad surprise.

!iutile = 0

!if(iutile.eq.0) return

! ----------------------------------------------

! -----------------------------------------------------
!  Calculate the momentum source terms Sx, Sy and Sz for actuator disc representation
! ------------------------------------------------------
if(nMax_elms_stop(Nfarmax) > 2) then
  do nbelm = 1, nMax_elms_stop(Nfarmax)
     iel      = AD_position(nbelm)%nCel
     ux       = rtpa(iel, iu(iphas) )
     uy       = rtpa(iel, iv(iphas) )
     uz       = rtpa(iel, iw(iphas) )

     radius   = AD_position(nbelm)%rad
     azim_ang = AD_position(nbelm)%azim_deg
     u_theta  = - (uy * dcos(azim_ang)) + (ux * dsin(azim_ang) ) + (omega * radius)
     W_rel2 = (u_theta * u_theta) + (uz * uz)
     !--------------Sx source term ---------------------------------
     if (ivar.eq.iu(1)) then
       k_value     = - AD_position(nbelm)%dFX
!       crvimp(iel) = min(0.0, (- 2.0 * k_value * dsin(azim_ang) * u_theta) )
       crvimp(iel) = - 2.0 * k_value * dsin(azim_ang) * u_theta
       crvexp(iel) = - k_value * ( W_rel2 - (2.0 * dsin(azim_ang) * u_theta * ux) )
     !-------------Sy source term ----------------------------------
     else if (ivar.eq.iv(1)) then
       k_value     = - AD_position(nbelm)%dFY
!       crvimp(iel) =   min(0.0, (2.0 * k_value * dcos(azim_ang) * u_theta))
       crvimp(iel) =   2.0 * k_value * dcos(azim_ang) * u_theta
       crvexp(iel) =   - k_value * ( W_rel2 + (2.0 * dcos(azim_ang) * u_theta * uy) )
     !-------------Sz source term ----------------------------------
     else if (ivar.eq.iw(1)) then
       k_value     = - AD_position(nbelm)%dFZ
```

```
!       crvimp(iel) = min(0.0, (- 2.0 * k_value * uz) )

       crvimp(iel) = - 2.0 * k_value * uz

       crvexp(iel) =   k_value * ( (uz * uz) - (u_theta * u_theta) )

     endif

  enddo

endif

! -----------------------------------------------------

!  end of loop for momentum source terms consists of Sx, Sy and Sz

! ------------------------------------------------------

  !------------------------------------------------------------------------

  ! deallocate dynamic arrays associated with the 2D blade design data base

  !  the Actuator Disc elements

  !------------------------------------------------------------------------



  if ((ncall == 3).and.( ntcabs >= ntmabs)) then

              do n = 1, n_be

                current => blade_position(n)%ptr_bsection

                 do while ( associated( current ) )

                   previous => current

                   current => current%next

                   deallocate( previous )

                 end do

              enddo    ! loop of n_be

              deallocate(blade_position)

              do n = 1, nMax_elms_stop(Nfarmax)

                current => AD_position(n)%ptr_Belement1

                 do while ( associated( current ) )

                   previous => current

                   current => current%next

                   deallocate( previous )

                 end do

              enddo

              do n = 1, nMax_elms_stop(Nfarmax)

                current => AD_position(n)%ptr_Belement2

                 do while ( associated( current ) )

                   previous => current

                   current => current%next

                   deallocate( previous )

                 end do
```

```
             enddo
             deallocate(AD_position)
            ! --------------------------------------------------
            ! Close files at final time step
            ! --------------------------------------------------
            if (irangp.le.0) then

            do ii = 1, 1

             close(impout(ii))

            enddo

            endif
            ! --------------------------------------------------
            ! Close files at final time step
            ! --------------------------------------------------

  !---------------------------------------------------------------------------
 endif
!--------
! Formats
!--------

 1000 format(' User source termes for variable ',A8,/)

!----
! End
!----
if (ncall == 3) then
   if (irangp.le.0) then
     print*,' time step = ',ntcabs
   endif
   ncall = 0
endif
return
contains
         subroutine find_cl_cd(alpha,Nrec,ptr_bsect,cl,cd,icheck)
          use turbine_design
```

```
implicit none

integer      irec,Nrec,n

double precision cl,cd,cl1,cl2,cd1,cd2,alpha,alpha1,alpha2, interp1, interp

double precision,  parameter :: v_small = 1.0d-6

double precision, parameter :: alfmin = -3.14

double precision, parameter :: cd_max = 1.07

type (b_section), pointer :: ptr_bsect

logical icheck


  icheck = .true.

  n = Nrec - 1

  if (alpha > alfmin) then

   do irec = 1,n

    alpha1 = ptr_bsect%alpha

    if (alpha1 == -999) call error_message("error1 with find_cl_cd")

    cl1    = ptr_bsect%cl

    cd1    = ptr_bsect%cd

    ptr_bsect => ptr_bsect%next

    alpha2 = ptr_bsect%alpha

    if (alpha2 == -999) call error_message("error2 with find_cl_cd")

    cl2    = ptr_bsect%cl

    cd2    = ptr_bsect%cd

    if ((alpha >= alpha1).and.(alpha <= alpha2)) then

      icheck = .false.

      exit

    endif

   enddo

   interp1 = (alpha2 -alpha1)

   if (dabs(interp1) < v_small)then

     call error_message("error4 with find_cl_cd")

   endif

   interp = (alpha -alpha1)/interp1

   cl = cl1 +  interp *(cl2 -cl1)

   cd = cd1 + interp *(cd2 -cd1)

   if (cd > cd_max) cd = cd_max

  else

    alpha1 = ptr_bsect%alpha

    if (alpha1 == -999) call error_message("error1 with find_cl_cd")

    cl1    = ptr_bsect%cl

    cd2    = ptr_bsect%cd
```

```
            ptr_bsect => ptr_bsect%next
            alpha2 = ptr_bsect%alpha
            if (alpha1 == -999) call error_message("error2 with find_cl_cd")
            cl2    = ptr_bsect%cl
            cd1    = ptr_bsect%cd
            interp1 = (alpha2 -alpha1)
            if (dabs(interp1) < v_small)then
             call error_message("error4 with find_cl_cd")
            endif
            interp = (alpha -alpha1)/interp1
            cl = cl1 + interp *(cl2 -cl1)
            cd = 0.5 * (cd2  + cd1)
          endif
          return
       end subroutine find_cl_cd
end subroutine ustsns
```

Bibliography

[1] Creech, A.C.W. (2009) *A three-dimensional numerical model of a horizontal axis, energy extracting turbine.* PhD thesis, Heriot-Watt University.

[2] Scottish Power Renewables. (2010). *Sound of Islay Demonstration Tidal Array – Non-technical Summary.* Technical Report (available online, from http://www.scottishpowerrenewables.com/pages/sound_of_islay.asp)

[3] Creech, A.C.W. (2010). *Development of a computational fluid dynamics mesoscale tidal channel model* - PerWaT WG3WP2_D4. Technical Report.

[4] Gretton, G.I. *Development of a computational fluid dynamic model for ahorizontal axis tidal turbine.* - PerWaT WG3WP2_D1. Technical Report.

[5] Stallard, T., Collings, R. (2010) *Calibration report for Scale Model Experiments.* PerWaT WG4WP2_D3 v2.0. Technical Report.

[6] Uribe, J. Discretision in Code_Saturne. Technical training manual. *University of Manchester,* 2011 (available online).

[7] Stallard, T. (2012) *Recent email contact entitled "CORRECTED – COMPARISON WITH YOUR Test 13 experimental data"*-26/09/12