



NTNU – Trondheim
Norwegian University of
Science and Technology

CFD Analysis of a Pelton Turbine in OpenFOAM

Jone Rivrud Rygg

Master of Science in Mechanical Engineering
Submission date: June 2013
Supervisor: Torbjørn Kristian Nielsen, EPT
Co-supervisor: Bjørn Winther Solemslie, EPT

Norwegian University of Science and Technology
Department of Energy and Process Engineering

EPT-M-2013-97

MASTEROPPGAVE

for

Stud.techn. Jone Rivrud Rygg

Våren 2013

Peltonturbin*Pelton turbine***Bakgrunn**

Blant Norges 1050 kraftverk så er det ca. 30% som har Peltonturbiner installert. Mange av disse turbinene er mer enn 40 år gamle og det er tid for oppgraderinger og eventuelt utskifting av disse. Ved en slik oppgradering søker man å øke virkningsgraden til kraftverket ved å blant annet redesigne turbinen som brukes. I en slik designprosess gjøres det ofte modellforsøk samt *Computational Fluid Dynamics* (CFD), og det er disse to elementene som vil være fokus for oppgaven

Mål

Gjennomfør modelltester av en modellturbiner designet ved Vannkraftlaboratoriet og gjennomføre verifikasiing av CFD-modelleringer av turbinene.

Oppgaven bearbeides ut fra følgende punkter

1. Gjennomføre og verifikasiere CFD-simuleringer av kommersiell modellturbinene i OpenFoam.
2. Sammenligne simuleringsresultatene fra OpenFoam og CFX med tidligere målinger gjennomført ved laboratoriet.
3. Undersøke muligheter for å forbedre simuleringene, da spesielt studere innløpsbetingelsene.
4. Gjennomføre simuleringer på modellturbin designet ved Vannkraftlaboratoriet hvis denne er tilgjengelig
5. Gjennomføre modelltester av turbin designet ved Vannkraftlaboratoriet hvis denne er tilgjengelig
6. Verifikasiere simuleringer på modellturbin designet ved Vannkraftlaboratoriet i OpenFoam hvis denne er tilgjengelig.

" " _ "

Senest 14 dager etter utlevering av oppgaven skal kandidaten levere/sende instituttet en detaljert fremdrifts- og eventuelt forsøksplan for oppgaven til evaluering og eventuelt diskusjon med faglig ansvarlig/veiledere. Detaljer ved eventuell utførelse av dataprogrammer skal avtales nærmere i samråd med faglig ansvarlig.

Besvarelsen redigeres mest mulig som en forskningsrapport med et sammendrag både på norsk og engelsk, konklusjon, litteraturliste, innholdsfortegnelse etc. Ved utarbeidelsen av teksten skal kandidaten legge vekt på å gjøre teksten oversiktlig og velskrevet. Med henblikk på lesning av besvarelsen er det viktig at de nødvendige henvisninger for korresponderende steder i tekst, tabeller og figurer anføres på begge steder. Ved bedømmelsen legges det stor vekt på at resultatene er grundig bearbeidet, at de oppstilles tabellarisk og/eller grafisk på en oversiktlig måte, og at de er diskutert utførlig.

Alle benyttede kilder, også muntlige opplysninger, skal oppgis på fullstendig måte. For tidsskrifter og bøker oppgis forfatter, tittel, årgang, sidetall og eventuelt figurnummer.

Det forutsettes at kandidaten tar initiativ til og holder nødvendig kontakt med faglærer og veileder(e). Kandidaten skal rette seg etter de reglementer og retningslinjer som gjelder ved alle (andre) fagmiljøer som kandidaten har kontakt med gjennom sin utførelse av oppgaven, samt etter eventuelle pålegg fra Institutt for energi- og prosessteknikk.

Risikovurdering av kandidatens arbeid skal gjennomføres i henhold til instituttets prosedyrer. Risikovurderingen skal dokumenteres og inngå som del av besvarelsen. Hendelser relatert til kandidatens arbeid med uheldig innvirkning på helse, miljø eller sikkerhet, skal dokumenteres og inngå som en del av besvarelsen. Hvis dokumentasjonen på risikovurderingen utgjør veldig mange sider, leveres den fulle versjonen elektronisk til veileder og et utdrag inkluderes i besvarelsen.

I henhold til ”Utfyllende regler til studieforskriften for teknologistudiet/sivilingeniørstudiet” ved NTNU § 20, forbeholder instituttet seg retten til å benytte alle resultater og data til undervisnings- og forskningsformål, samt til fremtidige publikasjoner.

Besvarelsen leveres digitalt i DAIM. Et faglig sammendrag med oppgavens tittel, kandidatens navn, veileders navn, årstall, instituttnavn, og NTNUs logo og navn, leveres til instituttet som en separat pdf-fil. Etter avtale leveres besvarelse og evt. annet materiale til veileder i digitalt format.

- Arbeid i laboratorium (vannkraftlaboratoriet, strømningsteknisk, varmeteknisk)
 Feltarbeid

NTNU, Institutt for energi- og prosessteknikk, 14. januar 2013



Olav Bolland
Instituttleader



Torbjørn K. Nielsen
Faglig ansvarlig/veileder

Medveileder: Bjørn Winther Solemslie

Abstract

During the spring of 2012, Lorentz Fjellanger Barstad developed a method for modelling the flow in a Pelton turbine subject to a high-speed water jet using the Computational Fluid Dynamics (CFD) software ANSYS CFX. The torque measurement was validated against experimental data. The aim of this master's thesis has been to develop a similar method with the Open Source tool OpenFOAM and to compare the two models.

A method has been created using the OpenFOAM solver interDyMFoam, capable of handling two-phase flow together with mesh motion. The approach has been to use both a stationary and a rotating mesh domain to allow for the relative motion between the high-speed jet and the turbine buckets. The Arbitrary Mesh Interface (AMI) was used as a boundary condition for the patches between the two domains to allow simulation between them. Meshing was done both with the built-in tool snappyHexMesh and with ANSYS Meshing. The latter gave the best control over mesh refinement and the mesh quality.

The results achieved from the method were unfortunately not as desired. Much water seems to accumulate between the buckets, giving severe backwash. The measured torque was significantly larger than both the experimental torque and the torque measured with the ANSYS CFX method. Additionally, the torque measurement curve from OpenFOAM contained instabilities and did not coincide well with the one generated in ANSYS CFX. The measured maximum torque of the method seemed to go towards the actual solution when the density of the mesh increased, but at the same time it gave more noise in the output, and made smoothing of the results necessary. The computational time needed for the simulations has been problematic, being almost thirty times that of ANSYS CFX.

Sammendrag

I løpet av våren 2012 utviklet Lorentz Fjellanger Barstad en metode for å modellere strømningen i en peltonturbin påvirket av en vannstråle med høy hastighet. Det ble gjort ved hjelp av Computational Fluid Dynamics (CFD) programvaren ANSYS CFX. Dreiemomentet beregnet med metoden ble deretter validert mot eksperimentelle data. Målet med denne masteroppgaven har vært å utvikle en lignende metode med Open Source verktøyet OpenFOAM og sammenligne de to modellene.

En metode har blitt utviklet ved hjelp av OpenFOAM løsener interDyMFoam, som er i stand til å håndtere to-fase strømning sammen med mesh som er i bevegelse. Dette er gjort ved å bruke både et stasjonært og et roterende domene for å tillate relativ bevegelse mellom vannstrålen og turbinskovlene. Arbitrary Mesh Interface (AMI) ble brukt som grensebetingelse for overflatene mellom de to domenene for å muliggjøre simulering mellom dem. Meshing ble gjort både med det innebygde verktøyet snappyHexMesh og med ANSYS Meshing. Sistnevnte ga den beste kontrollen over forfining og kvalitet i meshet.

De beregnede resultatene med den utviklede metoden ble dessverre ikke som ønsket. Mye vann syntes å bli samlet opp mellom skovlene, noe som ga mye bakvask. Det målte dreiemomentet var betydelig større enn både det eksperimentelt målte dreiemomentet, og momentet beregnet i ANSYS CFX-modellen. I tillegg inneholdt målekurven for dreiemomentet fra OpenFOAM ustabiliteter og sammenfalt dårlig med den generert i ANSYS CFX. Det målte maksimale dreiemoment fra modellen så ut til å gå mot den virkelige verdien når tettheten av meshet økte, men ga på samme tid mere støy, slik at glattning av resultatene ble nødvendig. Beregningstiden for simuleringene har vært et problem, da den var nesten 30 ganger så stor som for tilsvarende simuleringer i ANSYS CFX.

Preface

This master's thesis was written at the Waterpower Laboratory, Department of Energy and Process Engineering at the Norwegian University of Science and Technology during the spring of 2013. The objective of this thesis was to develop a method for modelling the flow applied to a Pelton turbine by a high-speed water jet in the Open Source CFD package OpenFOAM.

In agreement with the author's supervisors Torbjørn K. Nielsen and Bjørn Winther Solemslie, the work in this master's thesis has been based on task number 1 and 2 in the project description. The model turbine design of Solemslie is not yet available, and the process of developing the model in OpenFOAM demanded more work than expected. As a consequence task number 1 and 2 have been prioritised rather than point three, improving the CFD models.

The project has been challenging, but inspiring, and has given me great insight within numerical flow analysis.

I would like to thank Bjørn Winther Solemslie, PhD-candidate and co-supervisor, for his great support. Supervisor Torbjørn K. Nielsen also deserves my gratitude, in addition to DynaVec for allowing me to use their design in the numerical analysis, and the NTNU HPC Group for letting me use the supercomputing facilities. Several others have been helpful in answering my questions and helping out. Thank you!

Thank you Jenny for all your support.



Jone Rivrud Rygg
Trondheim, June 7 2013

Contents

Abstract	v
Abstract in Norwegian	vii
Preface	ix
List of Figures	xv
List of Tables	xvii
Nomenclature	xix
1 Introduction	1
2 Theory	3
2.1 The Pelton Turbine	3
2.1.1 Energy conversion	3
2.1.2 Optimal Rotational Speed	6
2.2 Model testing of Pelton turbines	6
2.2.1 Dimensionless terms and scaling	7
2.2.2 Efficiency scale-up procedure in IEC 60193	7
2.3 Computational Fluid Dynamics	8
2.3.1 Governing equations	8
2.3.2 Meshing	9
2.3.3 Courant Friedrich Levy criteria	10
2.3.4 Discretisation	10
2.3.5 Solver procedures	11
2.3.6 Turbulence modelling	13
2.3.7 Verification and validation	15
2.4 Previous work on CFD for Pelton turbines	16
3 Setup of CFD analysis in OpenFOAM	19
3.1 Solution approach	20
3.1.1 Rotational movement of the mesh	20

3.2	OpenFOAM case structure	22
3.3	Geometry and meshing	23
3.3.1	Computational domain	24
3.3.2	Mesh generation with snappyHexMesh	24
3.3.3	Mesh generation with ANSYS	26
3.3.4	Defining zones for mesh movement	27
3.4	Pre-processing	28
3.4.1	Boundary and initial conditions	28
3.4.2	Configuring mesh movement	30
3.4.3	Fluid properties	32
3.4.4	Turbulence	32
3.4.5	Solver settings	32
3.4.6	Simulation control	34
3.5	Post-processing	36
3.6	Simulations	37
4	Results	39
4.1	Torque measurement	40
4.2	Convergence and stability	42
4.3	Flow visualisation	44
4.4	Computational time	48
5	Discussion	51
5.1	Torque prediction	51
5.2	Convergence and stability	52
5.3	Flow visualisation	52
5.4	Computational time	53
6	Conclusion and Further Work	55
Bibliography		57
Appendices		I
A	snappyHexMesh Workflow	III
A.1	Preparing the geometry	III
A.2	Creation of the background mesh	III
A.3	Configuring snappyHexMesh	IV
A.4	Improving the mesh	V
A.5	Define AMI-interface patches	V
A.6	<i>blockMeshDict</i> example	VI
A.7	Example <i>snappyHexMeshDict</i>	VIII
B	OpenFOAM case setup	XXIII
B.1	Velocity field U	XXIII
B.2	Dynamic pressure field p_{rgh}	XXV
B.3	Phase fraction field α_{phase1}	XXVII

B.4	fvSchemes	XXIX
B.5	fvSolution	XXXI
B.6	transportProperties	XXXIII
B.7	dynamicMeshDict	XXXV
C	Scripts for mesh generation and manipulation	XXXVII
C.1	snappyHexMex execution - <i>Allrun1.pre</i>	XXXVII
C.2	Mesh manipulation - <i>Allrun2.pre</i>	XXXVIII

List of Figures

2.1	Main dimensions in a Pelton turbine	4
2.2	Energy conversion in a Pelton turbine	5
2.3	Inlet and outlet velocities in a Pelton bucket	5
2.4	Time averaging for a statistically steady flow and ensemble averaging for an unsteady flow	13
2.5	Diffusion effect on flow with wrongly aligned cells, using a low reso- lution discretisation scheme	16
3.1	OpenFOAM file structure	22
3.2	Principal sketches of computational domains	24
3.3	Mesh refinement in ANSYS	27
3.4	Overview of boundaries	29
3.5	Comparison of boundary conditions at time step 0.007 , $\alpha_1 > 0.3$.	31
4.1	Curves for torque applied to the middle bucket in $AM2$ calculated by OpenFOAM, filtered and smoothed in Matlab	40
4.2	Curves for torque applied to the middle bucket in $AM3$ calculated by OpenFOAM, filtered and smoothed in Matlab	41
4.3	Comparison of the normalised torque curves generated in Open- FOAM and ANSYS CFX	42
4.4	Plot of residuals for $AM3$	43
4.5	$AM3$ Interface Courant Number	43
4.6	Flow visualisation of the $AM3$ simulation displaying all cells with a volume fraction of water larger than 0.5, colored by the water velocity	45
4.7	Comparison of the jet velocity before and after it is affected by the jet split	46
4.8	Comparison of the jet velocity and volume fraction across the jet split	47
4.9	Isovolum for $\alpha_1 > 0.1$ in one of the turbine buckets	48
4.10	Effect of high velocity on the back of bucket on the time step size. .	49
4.11	$AM3$ high velocities at time 0.016	50

List of Tables

2.1	Turbulence models listed by the number of extra transport equations needed	14
3.1	Important keywords in the <i>snappyHexMesh</i> dictionary	25
3.2	Boundary conditions used in the OpenFOAM simulation	28
3.3	Settings used in <i>fvSchemes</i>	34
3.4	Overview of parameters in <i>system/controlDict</i>	35
3.5	Simulation and mesh details	37
4.1	Overview of simulation runs	39

Nomenclature

		Re	Reynolds number
η_h	hydraulic efficiency	U	Velocity field in OpenFOAM
ω	angular velocity, s^{-1}	V	water velocity, m/s (section 3)
ϕ	friction coefficient	w	relative velocity, m/s
Φ_B	specific flow rate	We	Weber number
σ	surface tension coefficient, N/m	Subscripts	
v	kinematic viscosity, m^2/s	1*	inlet nozzle
d_s	jet diameter, m	1	inlet turbine
z_0	number of nozzles	2	outlet turbine
alpha1	Volume fraction in OpenFOAM	3	tailwater
B	bucket width, m	ED	reduced value
c	absolute velocity, m/s	opt	optimal
c	peripheral velocity, m/s	u	tangential component of velocity
D	turbine diameter, m	Abbreviations	
Fr	Froude number	AMI	Arbitrary Mesh Interface
g	acceleration of gravity, m/s^2	CFD	Computational Fluid Dynamics
H	head, mWc	CFD-Post	ANSYS CFX Post-Processor
n	rotational speed, rpm	CFL	Courant–Friedrichs–Lewy
p_rgh	Dynamic pressure field in Open-FOAM	CFX	CFD code by ANSYS
Q	volume flow, m^3/s	CFX-Pre	ANSYS CFX Pre-Processor
R	jet radius, m (section 3)	GAMG	Geometric-Algebraic Multi-Grid

GGI	Generalized Grid Interface	PISO	Pressure Implicit with Splitting of Operators
IEC	International Electrotechnical Commission	RANS	Reynolds Averaging Navier-Stokes
NTNU	Norwegian University of Science and Technology	RMS	Root Mean Square
PCG	Preconditioned Conjugate Gradient	SST	Shear Stress Transport
		STL	Stereolithography

Chapter 1

Introduction

Hydro power is a source of renewable energy that has been used globally for the generation of electricity since the 19th century. In Norway, it is the main source of electrical energy, and hydro power plants have been built throughout the country. About 30 % of the 1050 hydro power plants in Norway are equipped with Pelton turbines. Many of these have been running for more than 40 years and are due for refurbishing. When replacing an installed turbine the aim is to increase the efficiency of the power plant by redesigning the turbine. Model testing and Computational Fluid Dynamics (CFD) are used in the design process to optimise the result.

The flow in a Pelton turbine is complex and, even though numerous turbines have been built throughout the years, there are still phenomena that are not fully understood. Using CFD can facilitate our understanding of the interaction between the high-speed water jet and the rotating Pelton buckets.

OpenFOAM is a powerful Open Source software for resolving flows numerically, and its popularity is increasing globally. Being distributed without licensing costs, it gives new users the possibility to experiment with and utilise CFD. As the source code is publicly available, it has been especially popular with scientists. In addition to providing the transparency needed for researching applications, being distributed as Open Source makes it possible to modify or develop program modules. However, OpenFOAM is far less intuitive than many commercial packages, mainly because of its lack of an integrated Graphical User Interface (GUI). The documentation is basic, and not regularly maintained.

The aim of this thesis is to develop a method for predicting the torque in a Pelton turbine using OpenFOAM, and to compare it with the results from ANSYS CFX and experiments conducted at the Waterpower laboratory. The method developed in this thesis will be a useful contribution to other studies carried out at the Waterpower laboratory at NTNU. One such example is the design process of a reference Pelton design that PhD-candidate Bjørn Winther Solemslie is working on at the

time of writing. Developing a method for predicting the torque in a Pelton turbine using OpenFOAM will also make possible the use of CFD for Pelton turbines without the need for expensive commercial CFD software. Overall, the method that is developed in this thesis will be a valuable contribution to the future study and development of the Pelton turbine.

Chapter 2

Theory

This section presents relevant theory for Pelton turbines together with an overview of CFD and model testing aspects.

2.1 The Pelton Turbine

The Pelton turbine is a hydro power turbine used for high heads and relatively low volume flows. Pelton turbines are almost exclusively used for heads higher than 600 meters. The turbine is appreciated for its wide efficiency area, which is due to its regulation capabilities made possible by a varying number and opening of nozzles[6]. The main dimensions of a Pelton turbine and nozzle are shown in Figure 2.1.

2.1.1 Energy conversion

The Pelton turbine is an impulse turbine, meaning that all the energy converted in the turbine is due to the velocity energy of the water. The turbine runs in a turbine casing with atmospheric pressure, and there is no pressure difference between the turbine inlet and outlet.

Figure 2.2 gives a schematic overview of the energy conversion, and shows that

- The water has a high potential energy in 1^* , when it enters the nozzle.
- The pressure energy is converted to kinetic energy in the nozzle.
- When reaching the bucket at 1, the kinetic energy of the water is converted to rotational energy in the Pelton turbine.

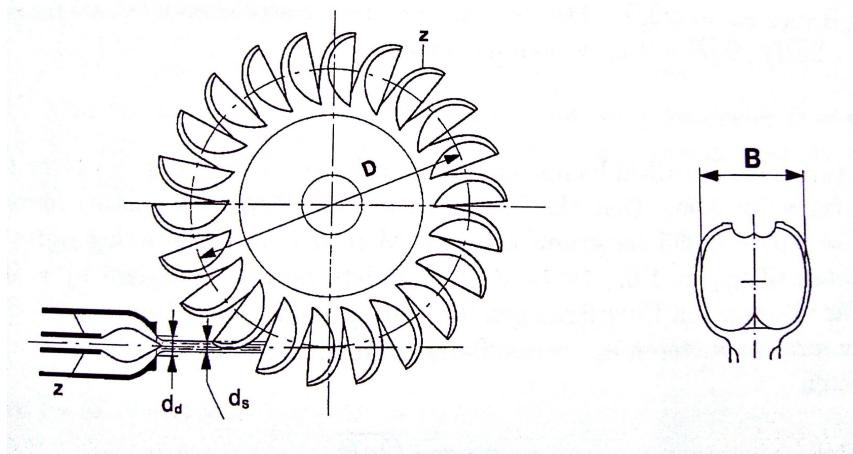


Figure 2.1: Main dimensions in a Pelton turbine[7]

- The remaining energy at the outlet of the turbine (step 2) consists of a small velocity $\frac{c_{m2}^2}{2}$ relative to the jet and a potential energy gh_3 relative to the tail water.

Figure 2.3 shows the velocities of the flow in a Pelton turbine. Entering the bucket, the jet has the diameter d_1 , absolute velocity $c_1 = c_{u1}$, velocity relative to the bucket w_1 , and a bucket speed u_1 . Exiting the bucket, there is an absolute velocity c_2 , a relative velocity w_2 , and a peripheral velocity u_2 . The bucket is moving with the peripheral velocity $u = u_1 = u_2$ [6].

For an ideal Pelton turbine the inlet velocity is

$$c_{1,\text{theoretical}} = \sqrt{2gH} \quad (2.1)$$

A friction coefficient ϕ can be used to account for the losses in the system.

$$c_1 = \phi \cdot c_{1,\text{theoretical}} = \phi \cdot \sqrt{2gH} \quad (2.2)$$

The energy converted in a Pelton turbine can be found with the use of an energy analysis, by assuming that it is a function of the peripheral and absolute velocity at the inlet and the outlet[6],

$$E_m = \omega(c_{u1}r_1 - c_{u2}r_2) = c_{u1}u_1 - c_{u2}u_2 \quad (2.3)$$

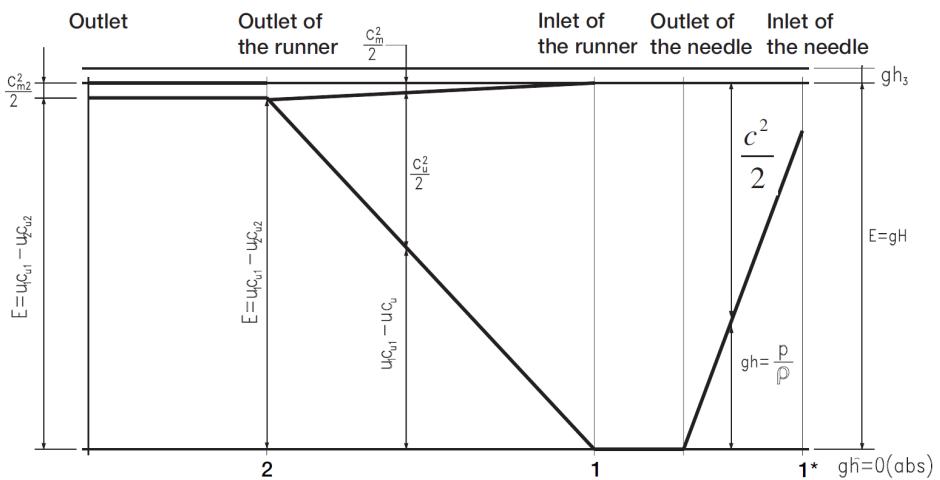


Figure 2.2: Energy conversion in a Pelton turbine[7]

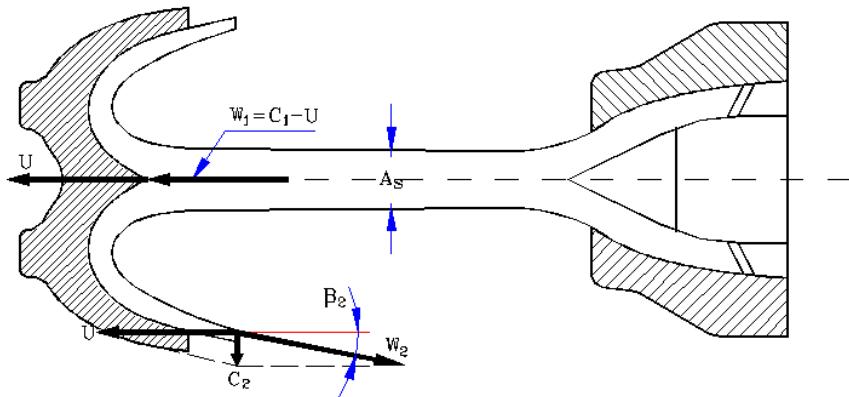


Figure 2.3: Inlet and outlet velocities in a Pelton bucket[6]

where E_m is the mechanical energy transferred from the water to the turbine. The available energy is both the energy that is converted to mechanical energy and the energy that is lost in the turbine. This can be expressed as

$$E = gH \quad (2.4)$$

We can now define the hydraulic efficiency using equations 2.3 and 2.4, and get

$$\eta_h = \frac{1}{gH} (c_{u1}u_1 - c_{u2}u_2) \quad (2.5)$$

Equation 2.5 is known as the Euler equation.

2.1.2 Optimal Rotational Speed

To find the optimal rotational speed we use the Euler equation and assume that the velocity c_{u2} , when leaving the turbine, is equal to zero.

$$\eta_h = \frac{c_{u1}u_1}{gH} \quad (2.6)$$

Assuming that $c_{u1} = c_1$ the optimal rotational speed u_{opt} is calculated

$$u_{opt} = \frac{\eta_h g H}{c_1} = \frac{\eta_h g H}{\phi \sqrt{2gH}} = \frac{\eta_h c_1}{2\phi^2} \quad (2.7)$$

By using equation 2.8[6] and 2.9 the optimal angular velocity ω can be calculated.

$$\omega = \frac{u}{r} = \frac{\eta_h c_1}{\phi^2 D} = \frac{\eta_h \sqrt{2gH}}{\phi \cdot D} [rad/s] \quad (2.8)$$

$$n = \frac{60}{2\pi} \cdot \omega = \frac{30 \cdot \eta_h \sqrt{2gH}}{\pi \cdot \phi \cdot D} [rpm] \quad (2.9)$$

2.2 Model testing of Pelton turbines

One or more model tests are usually conducted to verify the performance and operational area of the turbine when designing new turbines. Even with the increased computational power of today, model tests are needed to get a clear picture of the performance and undiscovered errors of the design. Additionally, it is used by the turbine producer to guarantee the hydraulic performance of a turbine to the customer.

2.2.1 Dimensionless terms and scaling

To compare a turbine model with a geometrically similar prototype, expressions for the reduced flow and the reduced velocity are used. The values are reduced using the head H of the machine as shown below.

$$Q_{ED} = \frac{Q}{D^2 \sqrt{gH_e}} \quad (2.10)$$

$$n_{ED} = \frac{nD}{\sqrt{gH_e}} \quad (2.11)$$

Equal Q_{ED} and n_{ED} give the same velocity diagrams both in the model and the prototype turbine[19]. There will, however, be a dissimilarity due to the higher Reynolds number in the prototype, resulting in lower friction losses. Thus, the efficiency will be higher in the prototype than in the model. Additionally, there are differences in the relative roughness between the model and the prototype turbine that will have to be taken into account. Consequently, the test results have to be scaled.

The method for scaling test results is different in reaction and impulse turbines. For reaction machines, scaling laws are well established and in use, while scaling for Pelton turbines is considerably more difficult. The various flow phenomena have to be taken into account when scaling. These are the: pipe flow, free-jet flow, unsteady flow with free surface in the buckets, and two-phase flow in the casing. The size and distribution of the mentioned flow phenomena, however, are not sufficiently known[12].

2.2.2 Efficiency scale-up procedure in IEC 60193

The international standard *IEC 60193* of the *International Electrotechnical Commission* applies to laboratory testing of model turbines. The equations given in IEC 60193, Annex K[15] give a procedure to approximate the efficiency scale-up for Pelton turbines. This procedure will be summarised below.

The specific flow rate Φ_B , the Froude number Fr , the Weber number We and the Reynolds number Re are calculated for the model. The expected corresponding dimensionless values for the prototype are also calculated. Formulas 2.12, 2.13, 2.14 and 2.15 are used, where Q is the volume flow, z_0 is the number of nozzles, E is the specific energy, B is the bucket width, g is the gravitational constant, ρ is the density, V is the velocity, σ^* is the surface tension coefficient and ν is the kinematic viscosity.

$$\Phi_B = \frac{4Q}{z_0 \cdot \pi \cdot (2E)^{1/2} \cdot B^2} \quad (2.12)$$

$$Fr = \left(\frac{E}{g \cdot B} \right)^{1/2} \quad (2.13)$$

$$We = \left(\frac{\rho B v^2}{\sigma^*} \right)^{1/2} \quad (2.14)$$

$$Re = \frac{B \cdot V}{\nu} \quad (2.15)$$

The following ratios of similitude can then be calculated:

$$C_{Fr} = \frac{Fr_P}{Fr_M} \quad (2.16)$$

$$C_{We} = \frac{We_P}{We_M} \quad (2.17)$$

$$C_{Fr} = \frac{Re_P}{Re_M} \quad (2.18)$$

Ultimately, the model hydraulic efficiencies η_{hM} are scaled up to prototype conditions using the following formula:

$$\Delta\eta_h = \Delta\eta_{hP} - \Delta\eta_{hM} = \Delta\eta_{Fr} + \Delta\eta_{We} + \Delta\eta_{Re} \quad (2.19)$$

$$\Delta\eta_h = 5,7 \cdot \Phi_B^2 (1 - C_{Fr}^{0,3}) + 1,95 \cdot 10^{-6} \frac{C_{We} - 1}{\Phi_B^2} + 10^{-8} \frac{(C_{Re} - 1)^2}{\Phi_B^2} \quad (2.20)$$

2.3 Computational Fluid Dynamics

This section covers general CFD theory and an overview of selected software. It is based mainly on Versteeg[29], Ferziger[11], Kristoffersen[17] and White[30].

2.3.1 Governing equations

The governing equations of Computational Fluid Dynamics represent the conservation laws of physics:

1. Conservation of mass - The continuity equation

$$\frac{\partial \rho}{\partial t} + \operatorname{div}(\rho \mathbf{V}) = 0 \quad (2.21)$$

2. Conservation of momentum - The Navier-Stokes equation

$$\rho \frac{D\mathbf{V}}{Dt} = \rho g + \nabla \cdot \tau'_{ij} - \nabla p \quad (2.22)$$

3. Conservation of energy - First law of thermodynamics

$$\rho \frac{Dh}{Dt} = \frac{Dp}{Dt} + \nabla(k\nabla T) + \Phi \quad (2.23)$$

where $\frac{D\mathbf{V}}{Dt} = \frac{\partial \mathbf{V}}{\partial t} + (\mathbf{V} \cdot \nabla) \mathbf{V}$, the last term Φ in equation (2.23) is the dissipation function and the viscous stress tensor τ'_{ij} is

$$\tau'_{ij} = \mu \left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i} \right) + \delta_{ij} \lambda \cdot \operatorname{div} \mathbf{V} \quad (2.24)$$

The Navier-stokes equation can be simplified if we assume incompressible flow and a constant viscosity

$$\rho \frac{D\mathbf{V}}{Dt} = \rho g - \nabla p + \mu \nabla^2 \mathbf{V} \quad (2.25)$$

The same applies to the continuity equation, which becomes

$$\nabla \cdot \mathbf{V} = 0 \quad (2.26)$$

2.3.2 Meshing

The first step taken in creating a CFD simulation is dividing the domain into a grid of discrete control volumes, for which the different variables will be calculated. The quality of the mesh used for solving a problem is critical to the quality of the solution. Mesh generation is dependent of the simulation that will be performed, and the effects that will be investigated. It is also a compromise between ensuring sufficient quality and limiting the number of cells, as this highly affects computational time. This is an iterative process, and a fair amount of experience with CFD is necessary to create a high quality mesh.

Structured meshes, identified by regular connectivity and consecutive numbering of elements[11], are most efficient with regards to calculation time, and are therefore often preferable. However, it is not possible to generate structured meshes for many complex geometries, as is the case for the Pelton bucket. Consequently, it

is necessary to use an unstructured type of grid for parts of the computational domain.

In OpenFOAM, mesh quality is determined with the *checkMesh* application. This application evaluates certain parameters that are essential for mesh quality, such as[28]:

- **Aspect Ratio:** Ratio between the longest and shortest edges. 1.0 is ideal.
- **Cell volumes:** Very small or large cells (relative to each other) should be avoided.
- **Mesh non-orthogonality:** The angle between the line connecting two cell centres and the normal of their common face. 0 is ideal.
- **Skewness:** The distance between the intersection of the line connecting two cell centres with their common face and the centre of that face. Smaller is better.

Additionally, the *growth ratio* of the cell size is an important quality parameter that should not exceed 1.25[8]. By limiting the growth ratio, there will not be big "jumps" in cell volumes, that could in worst case give instabilities or inaccuracies.

2.3.3 Courant Friedrich Levy criteria

The *Courant number*, or the *Courant Friedrich Levy criterion*, is important to consider when performing a CFD analysis. In a one-dimensional grid it is defined as

$$\text{Courant} = \frac{u\delta t}{\delta x} \quad (2.27)$$

where u is the fluid speed, δt is the timestep and δx is the mesh size. For a three-dimensional case this is generalised to a scale taking into account the dimension of the control volume[4].

The *CFL*-criteria means that if the Courant number is less than or equal to 1.0, the fluid does not travel more than one cell for each timestep. For time dependent explicit analyses, this criterion is necessary for stability and the Courant number should be limited below 1.0. Meanwhile, for implicit solvers like Ansys CFX, this is not a requirement for stability, and the Courant number can be larger than 1.0[3].

2.3.4 Discretisation

The governing equations have to be discretised in time and space to be solved numerically. Gaussian finite volume integration is based on summing values on cell faces, which must be interpolated from cell centres, and is the most common choice

for discretisation in OpenFOAM. The discretisation schemes can be of any order, but higher order schemes will be more complex and can give stability issues.

OpenFOAM provides great flexibility with regards to the choice of discretisation schemes and interpolation between points. A central differencing scheme based on the two nearest neighbour points on each side of the cell center, the Gauss linear scheme, is commonly used. This gives a second order accurate scheme as illustrated below.

$$\frac{\partial u}{\partial x} = \frac{u_{i+1} - u_{i-1}}{2\delta x} + O(\delta x^2) \quad (2.28)$$

For discretisation in time, the *Forward Euler* scheme is used, giving first order accuracy:

$$\frac{\partial u}{\partial t} = \frac{u_i^{n+1} - u_i^n}{\delta t} + O(\delta t) \quad (2.29)$$

Together this is known as the *Forward-Time Central-Space* scheme, giving first-order convergence in time and second-order convergence in space.

The *Upwind* discretisation scheme shown in equation 2.30 is commonly used for discretisation in space when stability issues are present, but can give unsatisfactory accuracy.

$$\frac{\partial u}{\partial t} = \frac{u_i^n - u_{i-1}^n}{\delta x} + O(\delta x) \quad (2.30)$$

The *vanLeer* option is a variant of *Total variation diminishing*. It can, for instance, be used together with the Gauss scheme to better resolve fields that are prone to have areas with strong shocks or large discontinuities. In those cases, using *vanLeer* can decrease the risk of *checker-board errors*.

OpenFOAM offers enhanced versions of some of the schemes for scalars that need to be strictly bounded. A thorough description of these options can be found in the User Guide[22], section 4.4.1.1.

2.3.5 Solver procedures

After discretising the equations as presented in the previous section, a set of algebraic equations are obtained. Those equations need to be solved with an iterative technique by: 1) guessing a solution, 2) linearising the equations about that solution, and 3) improving the solution until it converges[11].

Several linear solvers are available, and it should be noted that OpenFOAM distinguishes between symmetric and asymmetric matrices, depending on the structure of the equation being solved. However, the solver will give an error message if a

symmetric linear solver is used for an asymmetric matrix, or the other way around. Some linear solvers, such as the *Preconditioned conjugate gradient (PCG)*, will also require the use of a preconditioner to increase its performance.

As the matrix solvers are iterative, they will produce a *residual* as a measure of the error in the solution. A smaller residual gives a more accurate solution. A *solver tolerance* has to be specified to control the accuracy of the linear solver. Specifying a low tolerance might be time consuming, and it is therefore important to consider an appropriate compromise between accuracy and the number of iteration loops needed to achieve convergence.

The *Geometric-algebraic multi-grid solver (GAMG)* is commonly used to solve the pressure equations. The principle behind GAMG is to first calculate an intermediate solution on a coarse mesh, and then map it onto a finer grid using the first solution as an initial guess. This is considered a relatively quick method. The High Performance Computing Group at NTNU investigated the parallel performance of OpenFOAM on the *Vilje* supercomputer[20], and recommends using the GAMG solver for the pressure equation for any incompressible analyses up to about 9 nodes/144 processes.

The flow pressure gradient contributes to all three momentum equations in the Navier-Stokes equations, complicating their solution. Therefore, special methods are used for pressure-velocity coupling. *Pressure Implicit with Splitting of Operators (PISO)* is a common choice for transient analyses. It can be summarised as shown below[28][11].

1. Solve the discretised momentum equation to compute an intermediate velocity field, using the latest solutions as the starting estimate.
2. Compute the mass fluxes at the cells faces.
3. Solve the pressure equation.
4. Correct the mass fluxes at the cell faces.
5. Correct the velocities on the basis of the new pressure field, update initial values for pressure and velocity.
6. Repeat step 2 until 5 the desired number of times.
7. Advance to the next time step.

Pressure-velocity coupling is an important aspect for segregated solvers, also known as uncoupled solvers. A segregated solver means that the three momentum equations are solved sequentially, before using the updated velocity field to calculate the pressure equation for continuity. OpenFOAM uses the principle of segregated solvers, while ANSYS CFX is an example of a coupled solver. In a coupled solver, all three momentum equations and the pressure equation are solved simultaneously in the same matrix, thus eliminating the need for pressure-velocity coupling. A coupled solver will need significantly more resources for each time step, but will

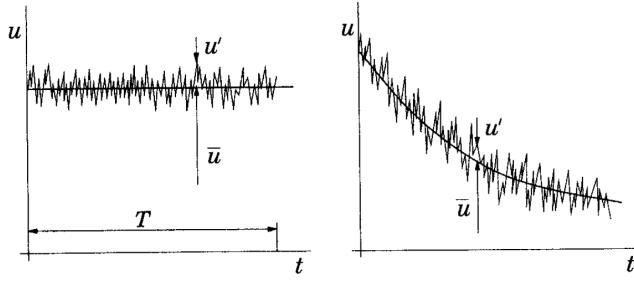


Figure 2.4: Time averaging for a statistically steady flow (left) and ensemble averaging for an unsteady flow [11].

at the same time converge with fewer time steps. This is further described in the ANSYS CFX-Solver Theory Guide, section 11.2[3].

2.3.6 Turbulence modelling

All flows relevant to engineering problems, from the very simple to the advanced ones, become unstable and turbulent above a certain Reynolds number. Thus, there is a need for tools that can represent the effects of turbulence. There are several ways to address turbulence numerically, ranging from Direct Numerical Simulation (DNS), via the simpler Large Eddy Simulation (LES), to Reynolds Averaging Navier-Stokes (RANS). The latter is the most commonly used for practical and industrial applications.

RANS assumes that for large time windows, relative to the period of the fluctuations, the mean velocity \bar{U} varies slowly with time as illustrated in Figure 2.4. The same method is used for the pressure variable. The variable U can be represented as the mean \bar{U} plus a fluctuating value U' , so that

$$U = \bar{U} + U' \quad (2.31)$$

where

$$\bar{U} = \frac{1}{T} \int_{t_0}^{t_0+T} U dt = \overline{\bar{U} + U'} = \bar{U} + \overline{U'} \quad (2.32)$$

and $\overline{U'}$ is by definition equal to zero. The averaged velocities and pressure can then be inserted into the Navier-Stokes equation.

$$\rho \frac{D\bar{V}}{Dt} + \rho \frac{\partial}{\partial x_j} (\overline{u'_i u'_j}) = \rho g + \mu \nabla^2 \bar{V} - \nabla p \quad (2.33)$$

Rewriting the equation for the mean viscous stress tensor components expressed by τ_{ij} gives

$$\rho \frac{D\bar{V}}{Dt} = \rho g + \nabla \cdot \tau_{ij} - \nabla \bar{p} \quad (2.34)$$

$$\tau_{ij} = \mu \left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i} \right) - \rho \bar{v}'_i \bar{v}'_j \quad (2.35)$$

Averaging the momentum equations yields six additional unknowns, which require additional equations to be solved. Therefore, turbulence models have been developed to predict the Reynolds stresses and to close the system of mean flow equations (equations 2.21 and 2.34). These are classified by the number of extra transport equations that need to be solved[29].

No. of extra transport equations	Name
Zero	Mixing length model
One	Spalart-Allmaras model
Two	k- ϵ model k- ω model
Seven	Algebraic stress model Reynolds stress model

Table 2.1: Turbulence models listed by the number of extra transport equations needed [29].

The standard k- ϵ model is extensively tested and well-established, and is known to have an excellent performance for many industrially relevant flows. However, it has a poor performance when it comes to certain unconfined flows, as well as flows with large extra strains (e.g. curved boundary layers, swirling flows), and rotating flows[29]. According to Perrig[24], the k- ϵ -model has shown poor performance in the bucket regions where the flow is suspected to high shear stresses.

The Wilcox k- ω model does not require wall-damping functions in low Reynolds number applications, which is why it is more accurate and robust in near wall areas. It is, however, very sensitive to the free stream value in ω , as the eddy viscosity becomes indeterminate or infinite when $k \rightarrow 0$ and $\omega \rightarrow 0$.

The k- ϵ model is much less dependent than the k- ω model on the assumed values in the free stream. Menter[18] proposed a hybrid model, blending the k- ϵ and k- ω models in near wall regions, and using the standard k- ϵ model in high turbulent areas far from the wall. This is known as the *Shear Stress Transport (SST) model*.

2.3.7 Verification and validation

For the results of a CFD simulation to be useful, it is essential to be aware of the quality and uncertainties it contains. This process is called verification and validation. Solution verification is defined as:

The process of determining that a model implementation accurately represents the developer's conceptual description of the model and the solution to the model[1]

The goal of verification is to quantify the errors in the implementation and solution of a certain calculation, and to present an estimation of the accuracy of the calculation. Common verification steps are checking the code, the iterative convergence, the consistency, the grid convergence, and the temporal convergence[17].

Validation is defined as:

The process of determining the degree to which a model is an accurate representation of the real world[1]

This involves verification of the numerical calculation as described above and comparing CFD results to experimental data or results from Direct Numerical Simulations. It is important to keep in mind the uncertainties of experimental data.

Numerical solutions are always approximations. Ferziger[11] lists three kinds of systematic errors:

- **Modeling errors:** The difference between the real world and the mathematical solution.
- **Discretisation errors:** The difference between the exact solutions of the conservation equations and the exact solution of the discretised equation system.
- **Iteration errors:** The difference between the iterative and the exact solution of the discretized equation system.

Numerical diffusion can occur as a result of misaligning the mesh with the flow, especially when using low order discretisation schemes. Perfect alignment is normally not possible, and diffusion will therefore to some extent be present. Figure 2.5 illustrates how diffusion affects the flow when a low resolution discretisation scheme is used[4].

Additionally, uncertainties in the input data, code errors and user faults will affect the quality of the solution.

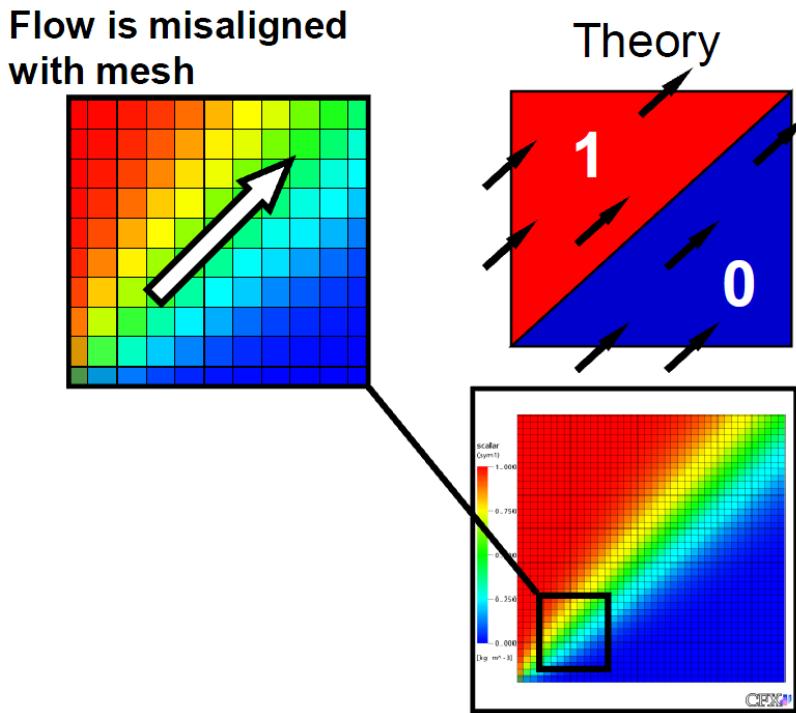


Figure 2.5: Diffusion effect on flow with wrongly aligned cells, using a low resolution discretisation scheme. The top right figure shows the ideal grid alignment[4]

2.4 Previous work on CFD for Pelton turbines

The use of CFD for Pelton turbine design is challenging due to the turbine's free surface, complex geometry and the relative motion between the jet and bucket. These effects are difficult to implement numerically. Hence, it is especially important to study the possibilities and limitations related to its use.

The doctoral thesis of Morten Hana[14] is to the author's knowledge the first study of CFD for Pelton turbines. Hana carried out a 2-dimensional simplified case, a 3-dimensional fixed jet case, and a 3-dimensional calculation with motion. The software RIPPLE, Flow-3D and CFX-4 were used, comparing the results and verifying them experimentally. Hana concluded that commercial CFD codes could in fact replace the graphical method developed by Henrik Christie[7], which was earlier used for evaluating Pelton turbine designs.

The PhD-thesis of Alexandre Perrig[24] investigated the free surface flow in the buckets using four experimental and numeric approaches: 1) Unsteady onboard wall pressure measurements, 2) high-speed flow visualisations, 3) onboard water film thickness measurements, and 4) CFD simulations. The 2-Phase Homogeneous

Model and the 2-Fluid Model were compared with experimental data, and Perrig found that the latter appeared to be the more accurate.

The master's thesis *An experimental and numerical study of the free surface Pelton bucket flow* of Lars Erik Klemetsen[16], investigated a simplified representation of the free surface flow through a static Pelton turbine. The thesis included measurements of the pressure distribution through the majority of the flow domain, together with the location of the air-water interface. The experimental results were compared with a numerical analysis in both CFX and Fluent. Measurements of the jet energy distribution were included and were found to be important for the numerical results. With the grid resolution above a certain limit, the definition of the inlet was found to be the main parameter.

The master's thesis of Lorentz Fjellanger Barstad[5] had the objective of developing and validating a numerical model for the torque applied to a non-stationary Pelton-bucket. The model was developed in ANSYS CFX and based on a model turbine supplied by the turbine producer DynaVec¹. Mesh independency was identified at approximately 4.5 million elements. Head independency was found to be likely, but not verified properly. A comparison of experimental and simulation results showed a torque over-prediction of approximately 1.5 % for this specific geometry, at a head of 75 m.

¹DynaVec is a company based in Trondheim, Norway, that designs, manufactures, and installs pumps and turbines in areas with sand erosion and corrosion problems.

Chapter 3

Setup of CFD analysis in OpenFOAM

A CFD model method for a Pelton turbine has already been developed and validated for ANSYS CFX at the Waterpower laboratory[5]. This thesis has investigated the possibilities of developing a similar method for OpenFOAM, with the goal of comparing the results from the two.

OpenFOAM version 2.1.1 has been used in this thesis. However, version 2.2.0 became available during the spring of 2013, and has been used to run some simulations as it was installed on NTNU's high performance computer *Vilje*¹ that was used for several of the simulations.

The simulations that were conducted form the basis for the analysis in this thesis. Case files used in one of the simulations are found in Appendix B. Additional case files, as well as an animation of a completed simulation, are found in the files delivered together with this thesis. The mesh and geometries are not included, however, as a commercial turbine design has been used for the simulations.

¹Vilje is ranked number 44 on the top 500 ranking of the most powerful commercially available computer systems (<http://www.hpc.ntnu.no>)

3.1 Solution approach

A good resolution of the flow is critical to correctly represent the various flow phenomena. Special attention will be paid to the following effects:

- Unsteady free surface flow in the buckets
- High speed jet with free surface
- Two-phase flow in the casing

Together this gives a complex solver and case setup, and an appropriate mesh is essential to achieve an accurate solution. At the same time, the computational cost will be high as a result of this type of simulation, and the mesh should not be refined more than necessary. Thus, it is important to refine the mesh in the necessary areas, while keeping it coarser where possible. Mesh quality criteria, such as for instance *Growth Rate* and *Skewness*, should be carefully monitored.

The approach of modelling three buckets, rather than the whole turbine, is chosen to reduce the calculation time, while still being able to model the effects of jet cut-off from the next bucket and possible backwash from the water in the previous bucket. To further simplify the calculations, only half of the three buckets and the jet are modelled, and a symmetry plane is used. Together, these simplifications make it necessary to correct the calculated results before comparing with the experimental data. These corrections are further described in Section 3.5, Post-processing.

When developing the model it was necessary to use a solver that could handle certain needs, such as:

- Two-phase flow
- Rotational movement of the mesh
- Sliding interface between rotating and stationary mesh

The *interDyMFoam* solver was found to satisfy these needs and chosen for the analysis.

3.1.1 Rotational movement of the mesh

In the CFX-simulation, a sliding mesh interface is used between the stationary and rotating mesh. One approach would be to use the same type of interface in OpenFOAM, as functionality is present both in OpenFOAM version 2.1.1, and in the Extend-Project's² OpenFOAM 1.6-ext. The different interfaces are described below.

²The Extend-Project aims to open OpenFOAM to community contributed extensions[26]. It provides repositories to facilitate worldwide collaboration, and is responsible for the Unofficial OpenFOAM Wiki[28]. The Extend-Project is not supported by the developer of the official OpenFOAM software.

The Arbitrary Mesh Interface (AMI)

AMI was introduced with OpenFOAM version 2.1.0, to allow simulation across disconnected, but adjacent, mesh domains. The principle behind AMI is to project the interface patch of one domain onto that of the other domain and interpolate[27].

The Generalised Grid Interface (GGI)

OpenFOAM 1.6-ext provides an implementation of the Generalised Grid Interface (GGI) to allow interaction between meshes. It was available before AMI was implemented in the official OpenFOAM distribution, and has thus been more widely used. Certain cases testing GGI have been made available by the Extend-Project and others. Grunde Olimstad, a former student at the Waterpower laboratory, used GGI successfully in his Master's thesis[21], calculating characteristics for a reversible pump turbine. Olimstad emphasised the need to use the boundary condition *cyclicGgi*, as it is independent of face numbering while this is not true for the *cyclic* boundary condition. Face numbering can be a problem when converting meshes from the Fluent format.

Another approach, proposed by Håkan Nilsson[13] at the Chalmers University of Technology, is to make a single domain-rotating mesh with a rotating inlet boundary condition, rather than using the two-domain approach of the CFX method. By avoiding the sliding mesh approach computational costs will be reduced. This method could also be applied in CFX.

In this project, it was chosen to use the Arbitrary Mesh Interface (AMI) for handling the sliding mesh interface between the two domains. The reason was that it is available in the official version of OpenFOAM, and does not require any new boundary conditions to be developed.

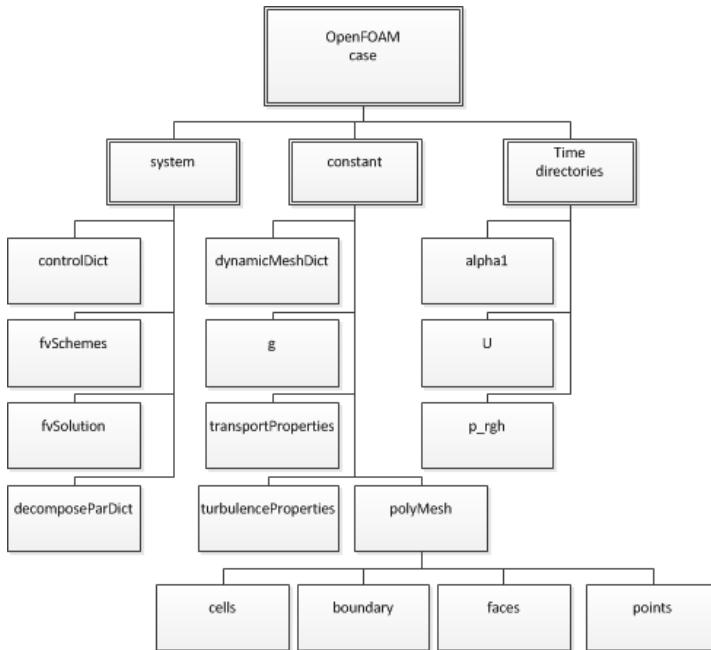


Figure 3.1: OpenFOAM file structure

3.2 OpenFOAM case structure

OpenFOAM cases are made up of files in a folder structure. There are separate files for velocity, pressure, and turbulence fields, as well as for necessary constants and settings. The files that are necessary in order to run a simulation are shown in Figure 3.1, and are commonly mentioned as *dictionaries*. Depending on the case and settings specified, additional files might be necessary, for instance when using the built-in meshing tools of OpenFOAM. These files might not be included in Figure 3.1, but will be mentioned when relevant.

Constant directory

This folder contains physical constants needed for the simulation, as well as a description of the mesh in the folder *polyMesh*.

System directory

Parameters for the solver settings and for controlling the simulation are found in this folder. The start and stop as well as time step settings are found in *controlDict*, while *decomposeParDict* contains the settings for decomposing a case to run it in parallel.

Time directories

Initial settings for the simulation can be found in the *0*-directory, while results are written by OpenFOAM to directories named by the time step, for instance *0.001*.

Fields such as the velocity field (∂/U), the dynamic pressure field (∂/p_{rgh}), or turbulent quantities like epsilon ($\partial/epsilon$) are each stored in separate files.

3.3 Geometry and meshing

OpenFOAM comes with various tools for mesh generation and manipulation.

blockMesh

The *blockMesh* tool is a native mesh generation tool that can be used for defining simple geometries and meshes. The dictionary *constant/polyMesh/blockMeshDict* is necessary.

snappyHexMesh

For more complex geometries *snappyHexMesh* should be used. In those cases, geometry files have to be supplied in the *Stereolithography (STL)* format, and a bounding domain needs to be defined with the *blockMesh* tool. *snappyHexMesh* will then mesh the surface geometries in the STL file, using the bounding domain as a base mesh. *snappyHexMesh* provides numerous options for the mesh generation and gives the user a lot of control over the result. The dictionary *system/snappyHexMeshDict* is necessary.

fluent3DMeshToFoam

There are tools for converting from various mesh formats to an OpenFOAM mesh. *fluent3DMeshToFoam* is such a tool and is used for converting meshes generated with ANSYS Meshing (Fluent *.msh*-format) to the OpenFOAM format.

snappyHexMesh was initially chosen as the preferred mesh generator due to its great flexibility and the possibility for automated mesh generation. However, the quality of the meshes generated with *snappyHexMesh* turned out to be less than satisfactory, often causing the simulations to crash. The simulations crashed mainly due to bad control of mesh refinement and important quality parameters, such as skewness. It was therefore decided to rather use ANSYS Meshing. The process of mesh generation with *snappyHexMesh* will nonetheless be included in this thesis for future reference.

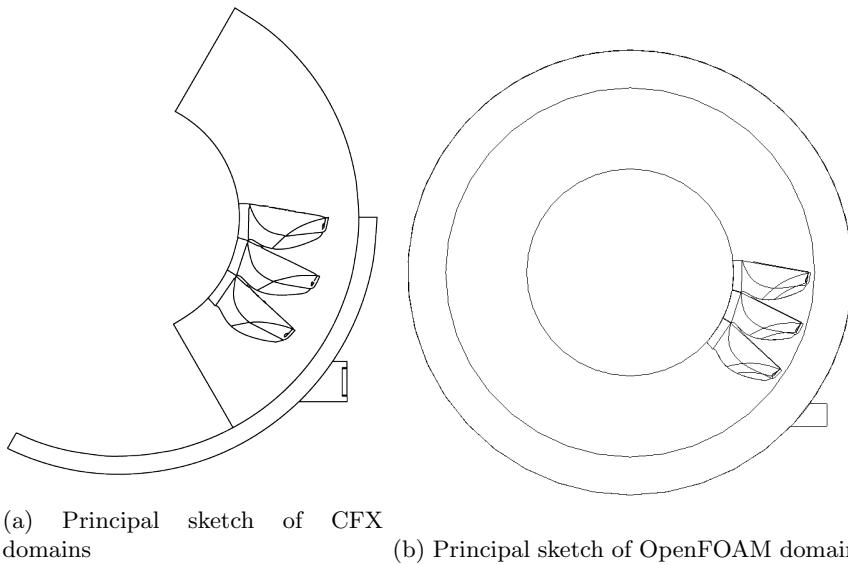


Figure 3.2: Principal sketches of computational domains

3.3.1 Computational domain

The CFD model of a Pelton turbine in OpenFOAM has been developed on the basis of the geometries and the setup used in the CFX model. Both the stationary and the rotating domains have been extended so that they are completely circular in order to satisfy the requirements of overlapping patches in the OpenFOAM sliding mesh interface (AMI). The principal sketch of the domains used in CFX is shown in Figure 3.2a, while the domains used in OpenFOAM are illustrated in Figure 3.2b.

The mesh consists of one stationary domain (outer) with the jet inlet, and one rotating domain (inner) with three half buckets. The boundaries are described in section 3.4.1.

3.3.2 Mesh generation with *snappyHexMesh*

The elements listed below are needed for mesh generation with *snappyHexMesh*. The workflow used for mesh generation with *snappyHexMesh* is further described in Appendix A.

- Background mesh generated with *blockMesh*
- Geometry files in *.stl* or *.obj* format
- Dictionary file *system/snappyHexMeshDict*

Activation rows	
castellatedMesh	Create the castellated mesh
snap	Do the surface snapping stage
addLayers	Add surface layers
castellatedMeshControls	Castellated mesh controls sub-dictionary
geometry	Sub-dictionary of all surface geometries
maxGlobalCells	Total cell limit (approx.)
locationInMesh	Location vector inside the region to be meshed
features	Refinement level for cells intersected by its edges
refinementSurfaces	Refinement level for cells intersected by its surfaces
refinementRegions	Refinement level for cells in relation to a surface
meshQualityControls	Generic mesh quality settings sub-dictionary
maxNonOrtho	Maximum non-orthogonality allowed
maxBoundarySkewness	Max skewness allowed on boundaries
maxInternalSkewness	Max skewness allowed on internal mesh
minVol	Minimum pyramid volume

Table 3.1: Important keywords in the *snappyHexMesh* dictionary system/*snappyHexMeshDict*[22]

The geometry files were made in ICEM CFD 14.0 by manipulating a mesh made in ANSYS Meshing, as exporting .stl-files is not possible in ANSYS Meshing or Design Modeler. The geometries were then scaled and converted to .obj with *surfaceTransformPoints*.

The background mesh was defined in *constant/polyMesh/blockMeshDict*, consisting of four blocks. The inner diameter of the rotating domain coincided with that of the rotating mesh generated in ANSYS Meshing, while the outer diameter of the *blockMesh* was larger than that of the original stationary domain. These dimensions allowed the desired domains to be extracted from the *blockMesh*-generated background mesh. All surfaces were extracted from the background mesh after it was generated, and the corresponding surface files were generated in *constant/triSurface*.

Mesh generation is an iterative process, and *snappyHexMeshDict* needs to be tuned to achieve the desired mesh refinement. An example *snappyHexMesh* is attached in Appendix A.7 as a complete reference of the parameters used. The parameters are many and can be overwhelming at first. The most important parameters, however, are listed in Table 3.1 to give an overview. After specifying the geometries (*geometry* sub-dictionary), it is important to specify *locationInMesh* correctly. In this case, the specified geometries created an enclosed region, smaller than the background mesh created with *blockMesh*. It was desirable that the cells outside the enclosed region were removed, thus the *locationInMesh* had to be set to a location inside this region.

The maximum mesh size is specified in *maxLocalCells*. Cell refinement can be specified in three ways:

1. **Explicit feature edge refinement:** Cells intersected by the edges of a specified geometry are refined
2. **Surface based refinement:** Cells intersected by geometry surfaces are refined
3. **Region-wise refinement:** Cells related to a surface are refined. Those can be cells inside or outside a closed surface, or cells that are within a specified distance of the surface

A combination of all the three refinement options is used. The refinement level is specified as a number, with a higher number giving more refinement, and 0 being the level of the background mesh specified in *blockMeshDict* and generated by *blockMesh*.

3.3.3 Mesh generation with ANSYS

The procedure to generate the meshes in ANSYS is based on the method of Barstad[5], but modified to fit with the way OpenFOAM handles geometry. Instead of generating separate files for the rotating and the stationary domain, both domains are generated as one, before splitting them into two adjacent meshes at the rotating-stationary mesh interface. A *Body of Influence* is created across the domains to ensure an equal surface and refinement at the interface of both the rotating and stationary domain. This also eliminated the need for importing and merging two separate meshes in OpenFOAM, thus increasing the effectiveness of the workflow. The location of the rotating-stationary mesh interface was also moved closer to the turbine buckets to avoid disturbances in the jet shape as long as possible.

Another obvious difference between the meshes of Barstad[5] and the ones used in this simulation is that those used in OpenFOAM were extended to be completely circular in order to support the use of the Arbitrary Mesh Interface (AMI) (see section 3.1.1).

The mesh refinements can be seen in Figure 3.3. Refinements are applied in the entire area of the *Body of Influence* to achieve a sufficiently fine mesh for resolving the air-water-interface of the jet both before and after reaching the buckets. A face sizing is applied to both the middle bucket and the first bucket exposed to the jet. Inflation layers in the bucket surfaces are also necessary to model the boundary layer where the jet interacts with the buckets. This was implemented in one mesh, but unfortunately the simulation did not finish in time to be included in this thesis.

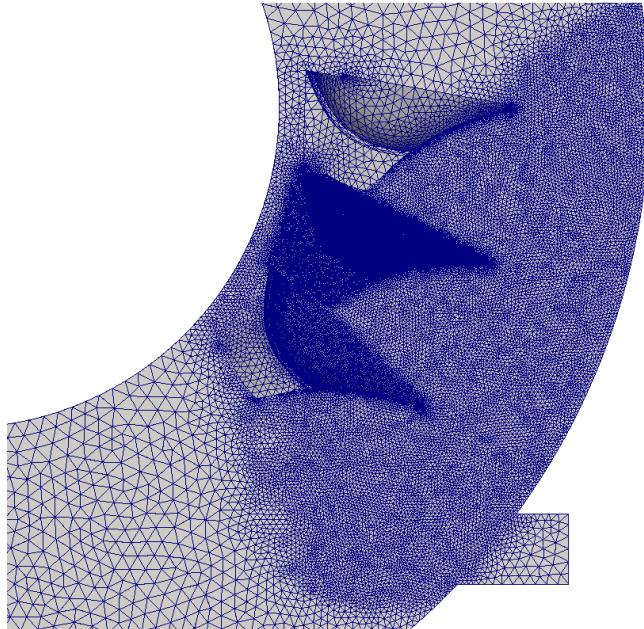


Figure 3.3: Mesh refinement in ANSYS

3.3.4 Defining zones for mesh movement

To implement the movement of the rotational domain it is necessary to group the different parts of the mesh in cell zones. Firstly, faces to be used in the AMI-patches need to be defined. Secondly, the cells of the rotating domain need to be grouped together.

The *topoSet* application is used to create the necessary zones. The zones must be defined in the dictionary *system/createAMIfaces.topoSetDict*. The procedure is executed with the *Allrun2.pre* script, and is roughly as follows: A cylindrical *cellSet* is created by specifying the dimensions of a cylinder coinciding with the interface between the rotating and stationary domain. It is important to specify accurate dimensions, or the AMI-surfaces will be uneven and of a low quality, and will most likely crash the simulation. The *cellSet* is used as a starting point to generate a *cellZoneSet* named *outerCells*, containing the stationary domain. The *faceZoneSet* named *rotif* is then created, containing the cell faces that will be converted into AMI-patches.

createBaffles is then executed to write the *rotif*-faces to the AMI-patches in *constant/polyMesh/boundary*.

	alpha1		U	
Boundary	Type	Value	Type	Value
jetinlet	fixedValue	1	fixedValue	(-38.38 0 0)
jetwalls	zeroGradient	-	fixedValue	0
mb, mbu bb, bbu tb, tbu	zeroGradient	-	fixedValue	0 0
rotsym	symmetryPlane	-	symmetryPlane	-
rotopen statopen	inletOutlet	0	pressureInlet- OutletVelocity	0
AMI1 AMI2	cyclicAMI	0	cyclicAMI	0

	p_rgh	
Boundary	Type	Value
jetinlet	zeroGradient	-
jetwalls	zeroGradient	-
mb, mbu bb, bbu tb, tbu	zeroGradient	-
rotsym	symmetryPlane	-
rotopen statopen	totalPressure	0
AMI1 AMI2	cyclicAMI	0

Table 3.2: Boundary conditions used in the OpenFOAM simulation

3.4 Pre-processing

Simulations in OpenFOAM are set up by creating and configuring the case files shown in Figure 3.1. The details of that process is described in this section.

3.4.1 Boundary and initial conditions

An overview of the boundary conditions can be found in Table 3.2 and Figure 3.4. The conditions are specified in the files *0/alpha1*, *0/U*, and *0/p_rgh*, for the volume fraction, velocity and the dynamic pressure, respectively.

The domains are initially filled with air, while water is defined to enter through the jet inlet by setting the *alpha1* field to a value of 1, with a speed of 38.38 m/s. This speed corresponds to a water height of 75 m.

All walls are defined as no slip walls by specifying zero velocity in all directions.

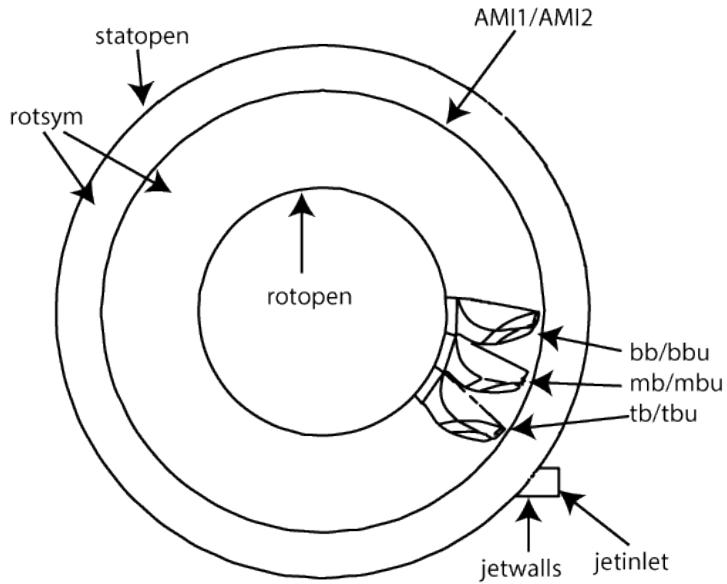


Figure 3.4: Overview of boundaries

mb, *bb* and *tb* contribute to positive torque and refer to the upper part of the middle, back, and top turbine buckets, respectively, while *mbu*, *bbu* and *tbu* refer to the back side of the buckets.

The jet wall has the same diameter as the nozzle ring of the model turbine tested by Barstad[5]. Friction is applied to the water jet by the wall with the aim of giving it a more realistic velocity profile compared to that of a free slip wall. Instead of using Barstad's method, with a small no-slip wall area adjacent to the jet inlet, a longer area of jet walls are modelled at the borders of the jet. The aim is to better reflect the effect of the jet nozzle ring on the jet velocity profile. Alternatively, testing an approach using a free slip jet wall could be of interest.

As previously mentioned, only half of the turbine buckets are modelled, thus a symmetry plane is used as a boundary condition. *rotsym* refers to the symmetry boundary for the entire fluid domain.

Since the Pelton turbine runs in a turbine casing with atmospheric pressure (see section 2.1.1), it is necessary to define corresponding boundary conditions. An outlet is also necessary so that fluid can exit the domain and the continuity can be fulfilled. *rotopen* and *statopen* are considered as outlets that are free to the atmosphere, allowing both outflow and inflow according to the direction of the internal flow. For *alpha1*, the *inletOutlet* boundary condition is specified, which provides a generic outflow condition, and also contains a specified inflow condition for the case of return flow. Thus, the condition is dependent of the direction of the flux crossing the patch. For positive fluxes (out of domain) the zero-gradient

condition is applied. For the opposite case (into domain) the user-specified fixed value is applied[27]. In this case, the inlet value is set to zero, so that only air will enter the domain if a negative flux is present. A similar approach is used for U , so that the *zeroGradient* is applied to outward flow. The *totalPressure* condition is used for the dynamic pressure field p_{rgh} , applying a *fixedValue* condition calculated from the specified total pressure and the local velocity[22].

The surface tension between the fluid interface and a wall surface can be specified by using the *alphaContactAngle* boundary condition[22] for the *alpha1* field. This also requires that values are provided for a static contact angle *theta0*; leading and trailing edge dynamic contact angles *thetaA* and *thetaR*, respectively; and a velocity scaling function for dynamic contact angle, *uTheta*. In addition, the keyword *limit* controls the gradient of *alpha1* on the wall and must be specified. The option *gradient*, that limits the wall-gradient so that *alpha1* is bounded on the wall, has been tested in this thesis. It is important to note that the flux has to be corrected to be zero at the wall for all options of *limit* except *zeroGradient*. This is done by setting the following boundary condition for p_{rgh} :

```
patchName
{
    type      fixedFluxPressure;
    adjoint   no;
}
```

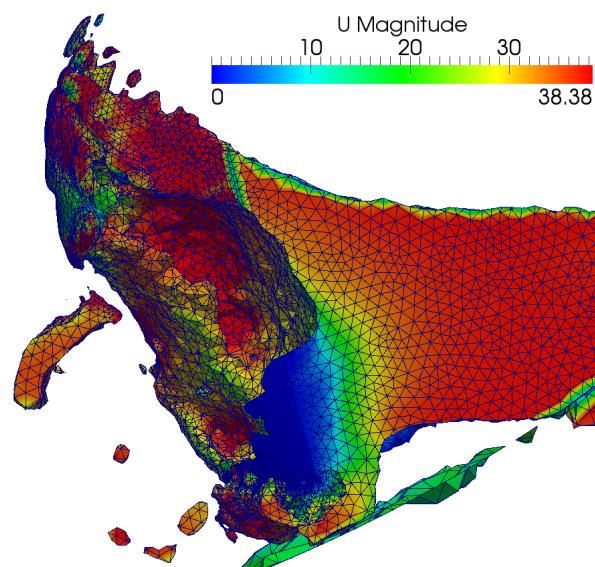
Surface tension effects can be neglected by specifying the *zeroGradient* boundary condition on *alpha1*, as presented in the OpenFOAM User Guide section 2.3.3[22].

A visual comparison of boundary conditions can be seen in Figure 3.5. One simulation was run with the *constantAlphaContactAngle* boundary condition in the buckets instead of the *zeroGradient* condition used otherwise. *zeroGradient* seemed to give a better result, and it was therefore decided to continue using this condition.

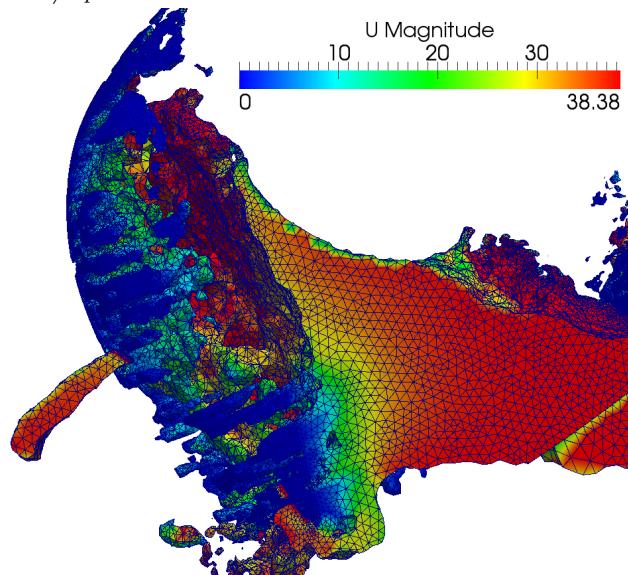
3.4.2 Configuring mesh movement

The simulations were set up with fixed speed rotational movement of the inner domain. The movement was specified in *constant/dynamicMeshDict*, giving the angular velocity for the desired *cellZone* containing the cells of the rotational domain. The angular velocity was calculated using equation 2.8, assuming ideal conditions. As the angular velocity needed to be specified in degrees per second, it can be expressed as

$$\omega = \frac{\sqrt{2gH}}{D} \frac{180}{\pi} \quad (3.1)$$



(a) *constantAlphaContactAngle* used as a boundary condition for turbine buckets in *0/alpha1*



(b) *zeroGradient* used as a boundary condition for turbine buckets in *0/alpha1*

Figure 3.5: Comparison of boundary conditions at time step *0.007*, *alpha1>0.3*

3.4.3 Fluid properties

The fluid properties are found in the *constant/transportProperties* file. The two phases have separate sub-dictionaries *phase1* and *phase2*. The transport model was selected to be *Newtonian* for both phases, meaning that the kinematic viscosity is single valued and specified under the keyword *nu*[22]. The density was specified under the keyword *rho*. Parameters for other viscosity models can be specified in sub-dictionaries such as *CrossPowerLawCoeffs* and *BirdCarreauCoeffs*.

The surface tension coefficient *sigma* was specified outside the phase sub-dictionaries. Additionally, the viscosity *nu* for *phase1* (water) was copied outside the phase sub-dictionaries as a workaround for using the *forces* function in multiphase. This is further described in section 3.5.

Gravity was specified in *constant/g* simply by entering the desired value vector. In this case, a gravity of 9.82 ms^{-2} in the negative z-direction was used, specified as

```
dimensions      [0 1 -2 0 0 0];
value          ( 0 0 -9.82 );
```

where *dimensions* gives the unit of the supplied value.

3.4.4 Turbulence

The turbulence model was chosen in *constant/turbulenceProperties*. Using a *Reynolds Averaging Navier-Stokes (RANS)* approach to solve turbulence can be done by specifying *RASmodel* for the *simulationType* keyword. When doing this, a dictionary file *constant/RASProperties* also needs to be created. This contains three options: *RASModel*, where a model for solving the RANS-equations should be specified, i.e. *kEpsilon* for the k- ϵ model described in section 2.3.6; *turbulence* can be used to switch the model on or off; and enabling *printCoeffs* prints a copy of relevant coefficient dictionaries if the user wishes to override the default coefficients for the chosen turbulence model.

In this thesis, only the laminar model has been used to simplify the simulations. The laminar model is chosen by specifying *laminar* as *simulationType* in *turbulenceProperties*. No additional parameters need to be specified when the laminar model is used.

3.4.5 Solver settings

The choices made with regards to linear solvers and discretisation schemes are essential for both stability and simulation speed, and should be carefully chosen.

The linear-solver control is specified in *system/fvSolution*. Each discretised equation needs to have a linear-solver chosen, and the solution tolerances specified.

The linear-solver iterates so that the equation residual reaches the desired tolerance, and stops when either: 1) the residual falls below *tolerance*, 2) the ratio of current to initial residuals falls below *reltol*, and 3) the maximum number of iterations *maxIter* is reached. *reltol* is normally set equal to 0 for transient analyses forcing convergence to the solver tolerance in each time step[22].

The pressure loops are the most time-consuming, thus setting the solution tolerance to an appropriate level is critical. If it is set to high, the accuracy of the final solution might not be adequate, but if it is set too low, the time needed to obtain a solution may become very high. The *Geometric-algebraic multi-grid solver (GAMG)* was chosen to solve the pressure equations, as described in section 2.3.5. The *smoothSolver* was used for the velocity fields together with the smoother *GaussSeidel*, which is generally the most reliable option[22].

The tolerances were initially set to 1e-06 for all equations, but were later changed to 1e-04 in order to make the simulations more time efficient. This can affect the accuracy and stability of the simulations, and a comparison between the results with regards to the tolerance level could be beneficial. If this is done, convergence of the pressure equation loops should be monitored as it can sometimes be difficult to achieve with tight convergence criteria[2]. It could be interesting to investigate the result of lowering the pressure equation residuals in a future study.

The settings for the pressure-velocity coupling loops are as follows:

```
PIMPLE
{
    momentumPredictor      yes;
    nCorrectors            4;
    nNonOrthogonalCorrectors 3;
    nAlphaCorr              1;
    nAlphaSubCycles         3;
    cAlpha                  1.5;
    correctPhi              no;
}
```

PIMPLE is a combination of the *PISO (Pressure-Implicit Split-Operator)* and *SIMPLE (Semi-Implicit Method for Pressure Linked Equations)* algorithms, but PISO is implemented when no over-relaxation factor is given[10]. This can be observed in the beginning of the simulation log files. Several parameters should be specified as shown above. *nCorrectors* controls the number of corrections in the PISO-loop as explained in section 2.3.5. The number of non-orthogonal correctors is specified with *nNonOrthogonalCorrectors*, the value depending on the level of non-orthogonality of the mesh used. 0 should be specified for completely orthogonal meshes. Enabling the momentum predictor allows an approximation of the new velocity field to be calculated, described as the first step in the PISO-loop in section 2.3.5. The *nAlphaCorr* and *nAlphaSubCycles* refer to the number of corrections and sub-cycles, respectively, to perform for the *alpha1* phase equation[22][28], and are used to increase the stability of the solution. *cAlpha* refers to compression

ddtSchemes	
default	Euler
gradSchemes	
default	Gauss linear
divSchemes	
div(rho*phi,U)	Gauss limitedLinearV 1
div(phi,alpha)	Gauss vanLeer01
div(phirb,alpha)	Gauss interfaceCompression
div((muEff*dev(T(grad(U)))))	Gauss linear
div((nuEff*dev(T(grad(U)))))	Gauss linear
laplacianSchemes	
default	Gauss linear limited 1.0
interpolationSchemes	
default	linear
snGradSchemes	
default	limited 1.0

Table 3.3: Settings used in *fvSchemes*

of the fluid interface, where a higher number refers to and enhanced compression and possibly a sharper interface, which can improve cases with a diffusive *alpha1*. The enhanced compression, however, gives a higher computational cost[9].

An overview of the discretisation schemes used, specified in *system/fvSchemes*, is found in Table 3.3. The first order, bounded and implicit *Euler Scheme* is used for discretisation in time (*ddtSchemes*). *Gaussian finite volume integration* is used for the derivative terms. For the gradient schemes (*gradSchemes*) the *linear interpolation* scheme is used as advised in the OpenFOAM User Guide[27]. *vanLeer01* is specified for *alpha1* with the aim of strictly bounding *alpha1* between 0 and 1.

3.4.6 Simulation control

The general settings for simulation execution, such as start and stop time, time step, writing of results, Courant number, and run time post-processing, are specified in *system/controlDict*. An overview of the input parameters is found in Table 3.4.

Time control

startFrom	Start time, for instance latest written time or a specific time
startTime	For specifying a start time, <i>startTime</i> must be given above
stopAt	Controls end of simulation
endTime	Specify end time for simulation, <i>endTime</i> must be given above
deltaT	Time step of the simulation

Data writing

writeControl	Controls the timing of output written to file
writeInterval	The write interval is specified here

Data reading

runTimeModifiable	Specify whether dictionaries are re-read at each time step
-------------------	--

Table 3.4: Overview of parameters in *system/controlDict*[22]

Adaptive time stepping has been used for the simulations performed in this thesis by specifying *adjustTimeStep yes*. That means that OpenFOAM automatically adjusts the time step size to the largest possible value, while still fulfilling the *Courant Friedrich Levy (CFL)* criteria. The settings listed below were used.

```
writeControl adjustableRunTime;
writeInterval 1e-3;
runTimeModifiable no;
adjustTimeStep yes;

maxCo 10;
maxAlphaCo 1;
maxDeltaT 1e-4;
```

Because of large computational time, the Courant number was specified as high as possible to increase the simulation speed. Increasing the Courant number further was attempted, but caused the simulations to crash. Therefore the Courant number for the flow at the air-water interface (*maxAlphaCo*) was limited to 1.0. This is normal for explicit solvers, as explained in section 2.3.3.

runTimeModifiable was set to *no* to avoid OpenFOAM checking the case files between each time step, potentially slowing the simulation. Also, no unnecessary time steps were written as this would have resulted in a lot of files being created, especially in the parallel cases subdivided into separate folders. *adjustableRunTime* was used for *writeControl* when using adaptive time stepping, making the time steps adjusted to coincide with the specified *writeInterval*.

3.5 Post-processing

There are various tools available for run-time post-processing in OpenFOAM, among others the built-in *forces* function. *forces* calculates forces and torques by integrating the pressure and skin-friction forces over a given list of patches[23]. The density of the flow has to be specified in *rhoInf*, and the centre of rotation *CofR* is needed for torque calculations. The output is found in the file *forces/<timedir>/forces.dat*. Note that the term *moment* is used for torque in the output file.

The *forces* function is not designed to be used with multiphase flows, and will not run without modifying the code or using a workaround. It will simply give the error message of not recognising the kinematic viscosity *nu*. If only the forces from one phase are needed, a workaround of defining *nu* outside of the phase-dictionaries in *transportProperties* will be possible. This method was verified in the project work of the author and others in a CFD course lectured at NTNU[17].

The output is given in columns containing time, pressure and viscous forces, and torque due to pressure and viscous forces. All of the preceding are decomposed in the x-, y- and z-directions. As the desired value in this thesis is the torque about the y-axis, the torque values in the y-direction are summed.

The torque is only measured on one half bucket, thus the value achieved has to be further post-processed to achieve the total torque of the turbine. That is done by post-processing the results in *Matlab* using the script described by Barstad in his master's thesis[5] to generate the total torque matrix.

Further post-processing, such as visual inspection, is done in *Paraview*. Creating an *IsoVolume* for certain values of *alpha1* (for instance $0.3 < \alpha_1 < 1.001$) will give a good visual view of how the water interacts with the buckets. Note that *alpha1* is unfortunately not strictly bounded below 1.0, as was the aim with the solver settings discussed in section 3.4.5. It was therefore necessary to use an *IsoVolume* with a max *alpha1* slightly larger than 1.0, as shown above.

Mesh	No. of cells	Face size mb/tb [mm]	Body size BoI [mm]
SHM1	172323	N.A.	N.A.
AM1	817914	default	4
AM2	2384935	1.2	2.6
AM3	4713956	0.6	1.8

Table 3.5: Simulation and mesh details

3.6 Simulations

An overview of the different meshes that were used in the simulations is found in Table 3.5. *SHM* refers to *snappyHexMesh*, used for mesh generation, while *AM* refers to ANSYS Meshing. Several other meshes were created and tested, both with *snappyHexMesh* and with ANSYS Meshing, but did not give successful simulations. As a general observation, meshes created with *snappyHexMesh* had mesh skewness present in the bucket area, causing the simulations to crash when the flow reached those areas. With regards to the meshes created with ANSYS Meshing, cell numbers lower than *AM1* resulted in unsuccessful simulations, while at the same time the large number of cells that were used gave very large simulation times.

The face and body sizing values are not available for *SHM1* since they are not specified in *snappyHexMesh*. Refinements are rather defined in a *level* relative to the background mesh (see section 3.3.2). Face sizing is not defined for *AM1*, and a default face size is therefore set automatically by ANSYS Meshing.

Chapter 4

Results

A combination of stability errors due to unsatisfactory mesh generation in *snappyHexMesh*, that was first used, and a high computational cost has put a limit on the number of simulations that were completed. Table 4.1 contains an overview of simulations that were run successfully. Please note that not all were run until end time due to limited computational resources. The end time is specified for the unfinished simulations.

Most simulations based on a mesh generated with *snappyHexMesh* failed to converge, and are therefore not included in this thesis. Simulation *SHM1*, however, ran until the end time despite having a very coarse mesh. The results, both visually and with regards to torque measurement were not realistic, as could be expected. However, some effects, such as the water flow out of the buckets, did look promising despite the mesh coarseness. A uniform refinement of this mesh unfortunately did not give satisfactory results, causing the simulations to crash.

Mesh	No. of cells	Run time [core hours]	Comment [Nm]
SHM1	172323	10	Stopped at 0.023
AM1	817914	1142	
AM2	2384935	7200	Stopped at 0.019
AM3	4713956	18945	

Table 4.1: Overview of simulation runs

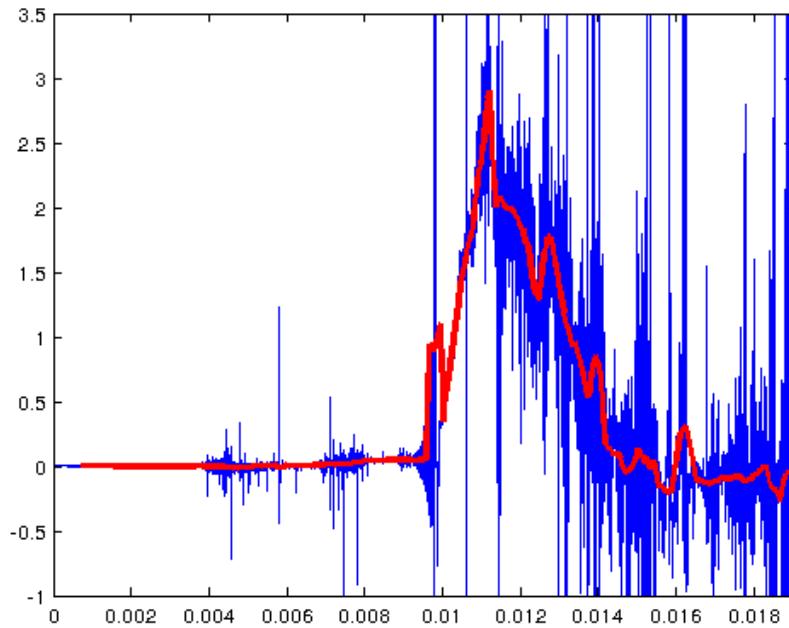


Figure 4.1: Curves for torque applied to the middle bucket in *AM2* calculated by OpenFOAM (blue), filtered and smoothed in Matlab (red)

4.1 Torque measurement

The torque applied to the middle bucket for *AM2* and *AM3* is shown in Figure 4.1 and 4.2. The blue line represents the original torque output of every timestep from OpenFOAM. There are large fluctuations in the torque curve. A simple filter was made in *Matlab*, based on a new matrix that was created with the median values of the original output and applying the built-in *smooth* function. The red line illustrates the values after filtering, which seems to represent the development of the original output fairly well. The same filter was applied to *AM1*.

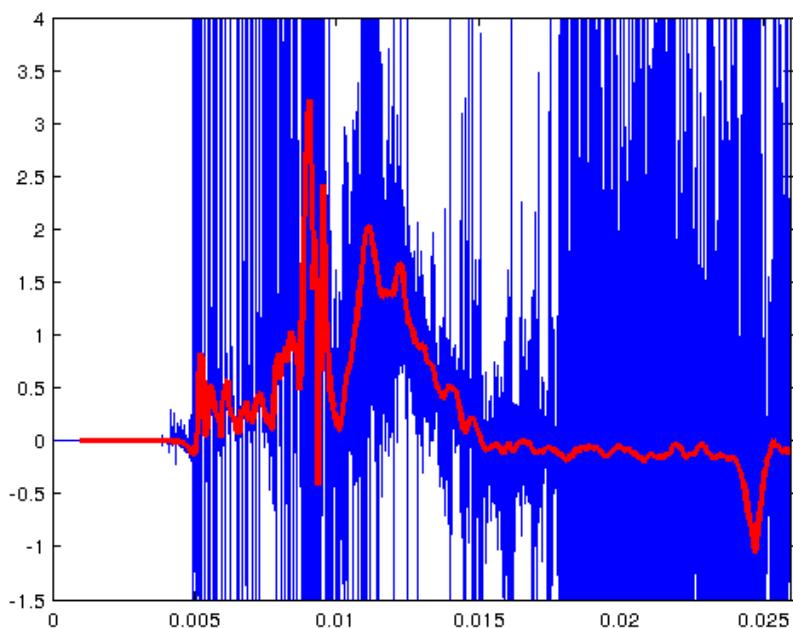


Figure 4.2: Curves for torque applied to the middle bucket in *AM3* calculated by OpenFOAM (blue), filtered and smoothed in Matlab (red)

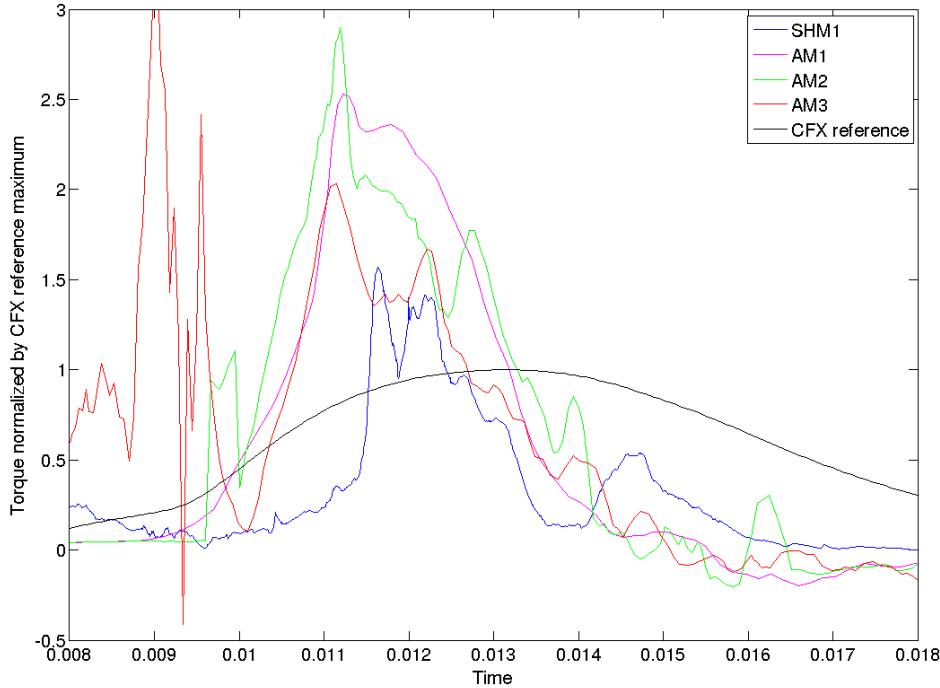
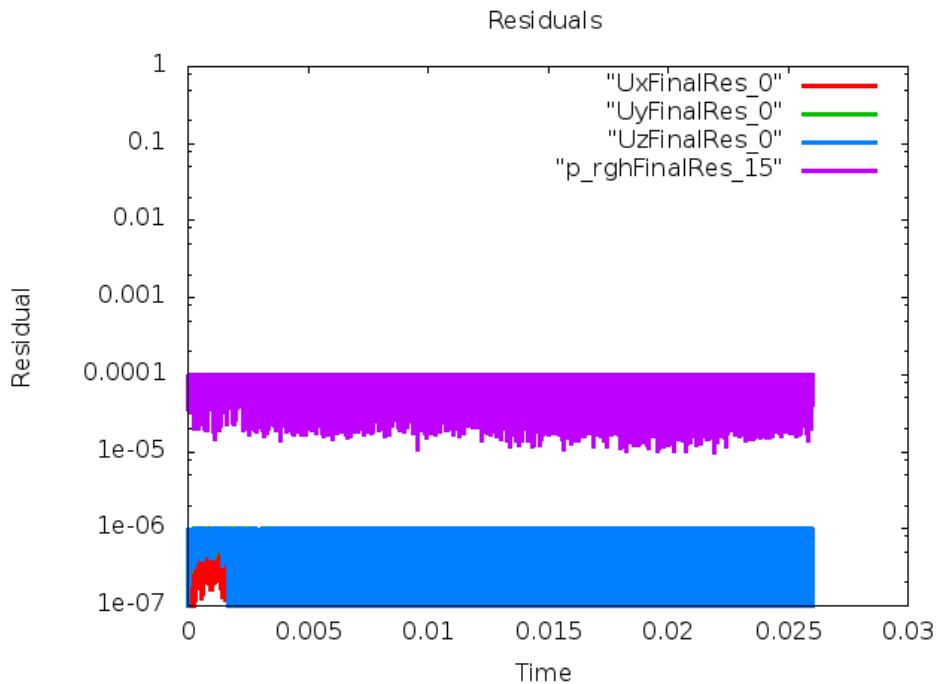
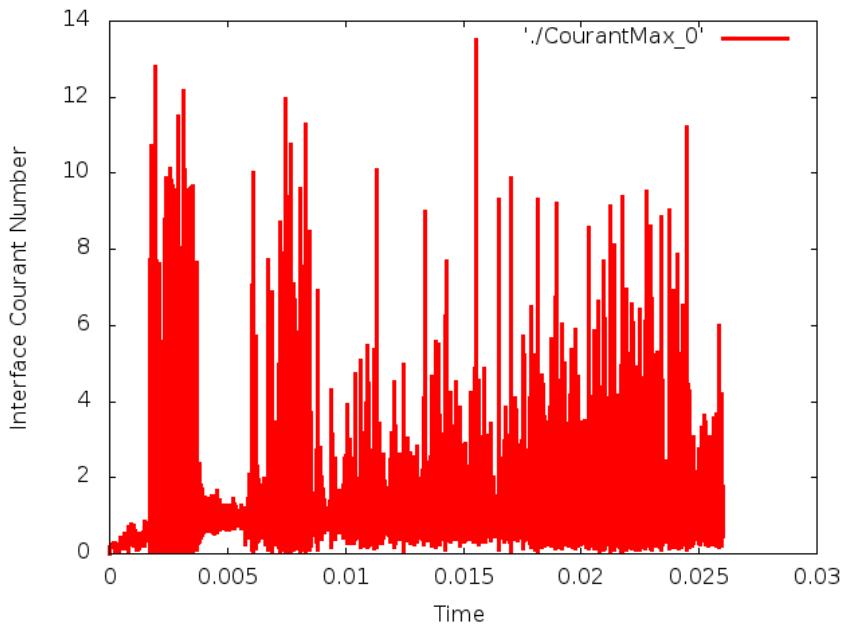


Figure 4.3: Comparison of the normalised torque curves generated in OpenFOAM and ANSYS CFX

The calculated torque curves are compared with a reference torque curve in Figure 4.3. The reference torque curve was calculated in *ANSYS CFX* for the author's project thesis the autumn of 2012[25] based on the method of Barstad[5]. It is difficult to identify the fluctuations in the torque curves that are real fluctuations due to the flow, and which that are related to numerical errors and can be considered as noise. When analysing the torque curves produced by *AM1*, *AM2* and *AM3*, similar patterns in the curves should be emphasised as these are more likely to represent the actual simulated flow. One such pattern is seen in the time interval *0.010* to *0.013*, just before the maximum value of the reference torque curve. The torque value of *AM3* in this interval is lower than that of *AM2*, which is lower than that of *AM1*, and at the same time closer to the maximum reference torque curve.

4.2 Convergence and stability

The residuals for *AM3* are plotted in Figure 4.4 and do not exceed the specified residual target. The Courant number for the interface between the air and water phases, in contrast, is not always below the specified value of 1.0, as shown in Figure 4.5. It exceeds the limit set for *maxAlphaCo* (explained in section 3.4.6) for 3.27 % of the time steps.

Figure 4.4: Plot of residuals for *AM3*Figure 4.5: *AM3* Interface Courant Number

4.3 Flow visualisation

A visualisation of the flow in the turbine buckets from the *AM3* simulation can be seen in Figure 4.6. When inspecting the calculated flow, several problematic phenomena can be identified. These are described below.

- As seen in Figure 4.6a, the jet is split at the location where it crosses the AMI-patches close to the lower bucket. A small amount of "leaked" water is present next to the top bucket.
- A large amount of water does not leave the buckets, causing severe backwash on the backside of the next bucket.
- The water is unevenly distributed across the buckets and there are several local accumulations of cells with a high volume fraction of water, most visible in the lower bucket in Figure 4.6d.
- The jet assumes an unnatural shape from Figure 4.6d after it starts interacting with the top bucket, and parts of it seem to disappear.

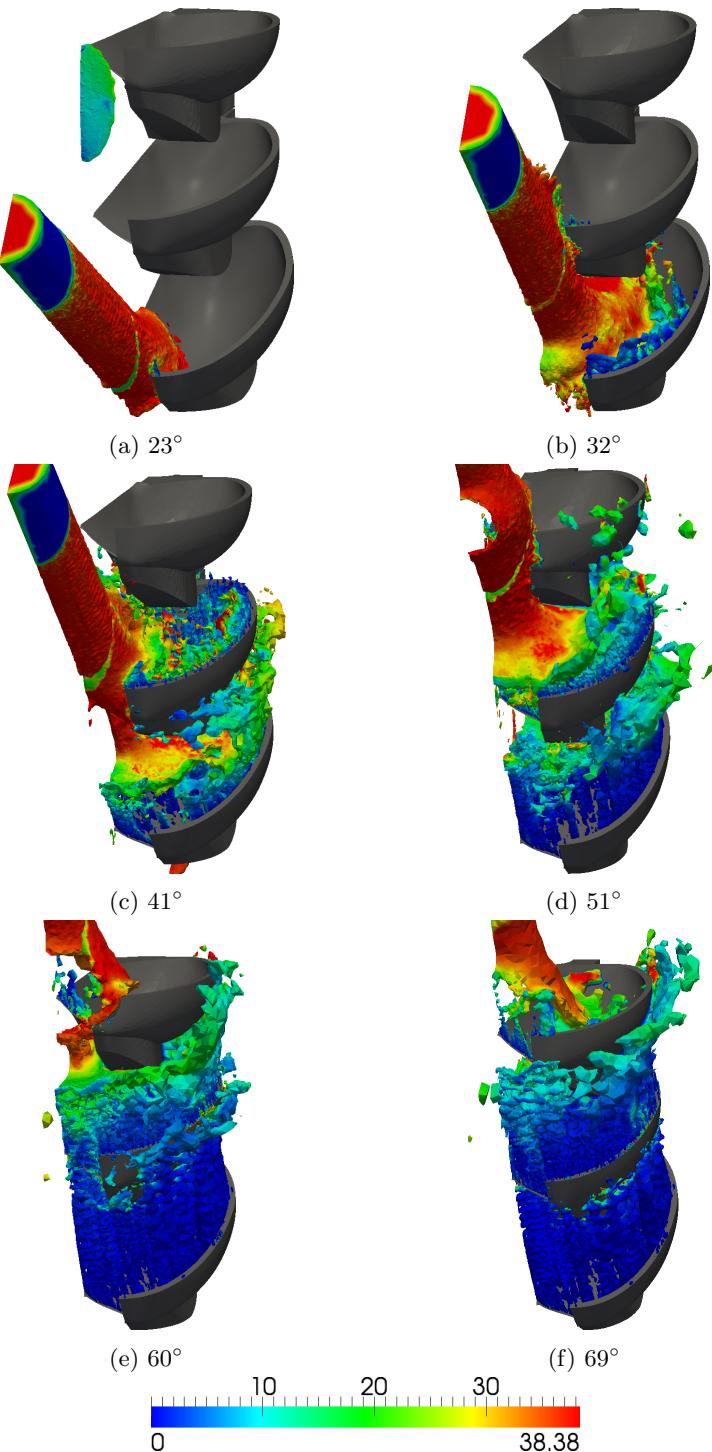


Figure 4.6: Flow visualisation of the *AM3* simulation displaying all cells with a volume fraction of water (*alpha1*) larger than 0.5, colored by the water velocity [m/s]. The angular position is specified below the separate sub-figures

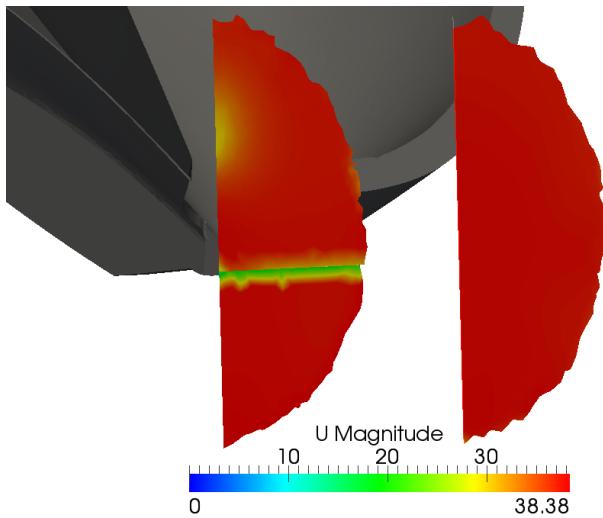


Figure 4.7: Comparison of the jet velocity before and after it is affected by the jet split. The green part of the left contour represents the jet split, with the lower part of the contour being outside the AMI-patches, and the top part inside.

The results from a further investigation of the jet split can be seen in Figure 4.7 and 4.8. The short distance between the AMI-patches and the buckets affects the velocity in the jet area closest to the bucket, causing the velocity to decrease as shown in Figure 4.7 (yellow half-circular area) and in Figure 4.8a (dip in the curve around 0.030 m). Taking the effect of the closeness to the turbine bucket into account, the velocity and the volume fraction of the jet seems to be unaffected by the jet split.

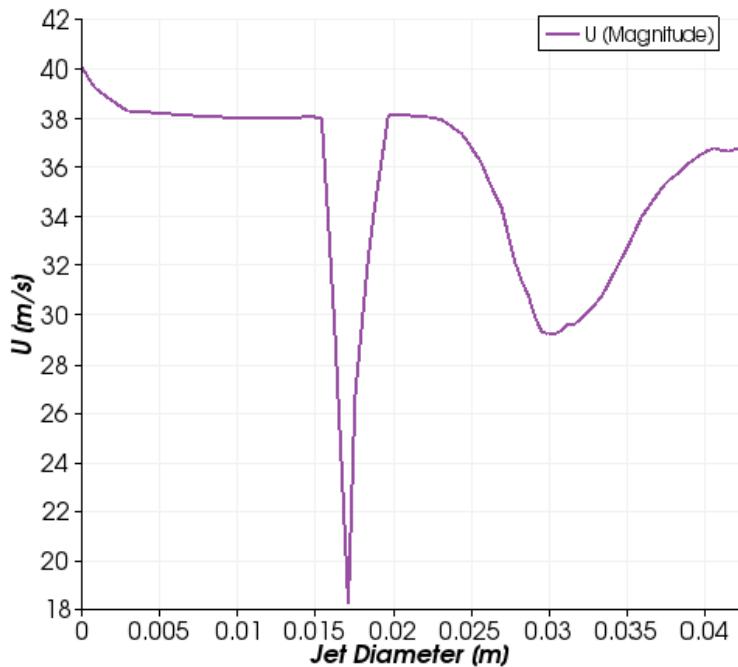
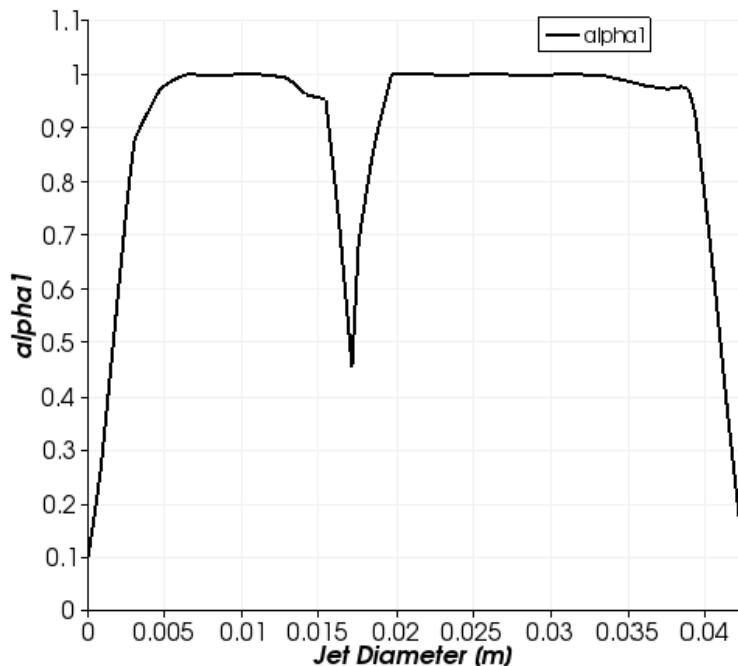
(a) Plot of U across the jet split(b) Plot of α_1 across the jet split

Figure 4.8: Comparison of the jet velocity and volume fraction across the jet split

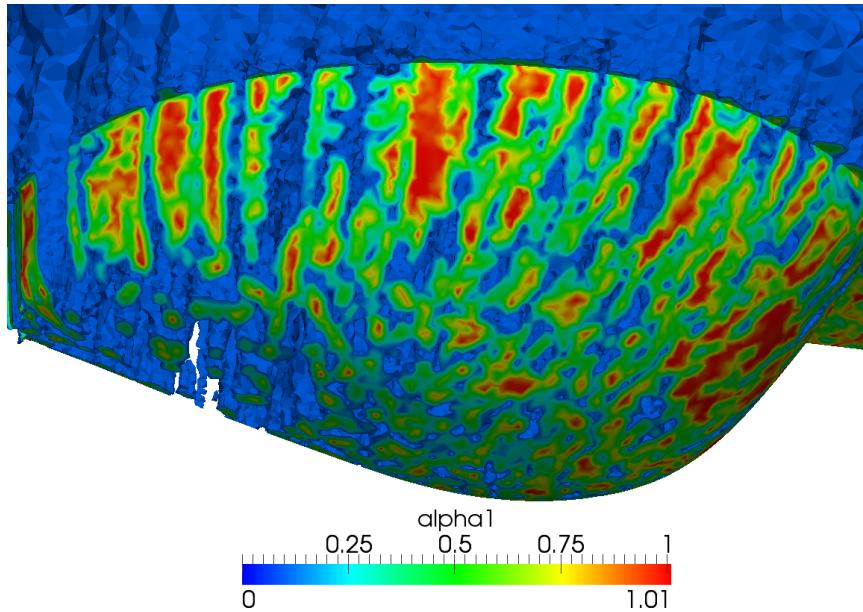
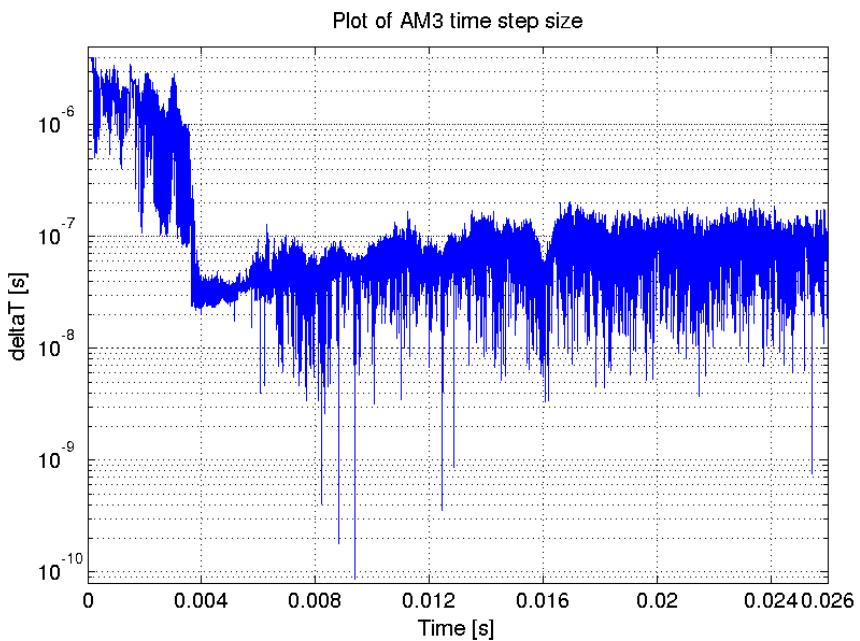


Figure 4.9: Isovolum for $\alpha_1 > 0.1$ in one of the turbine buckets

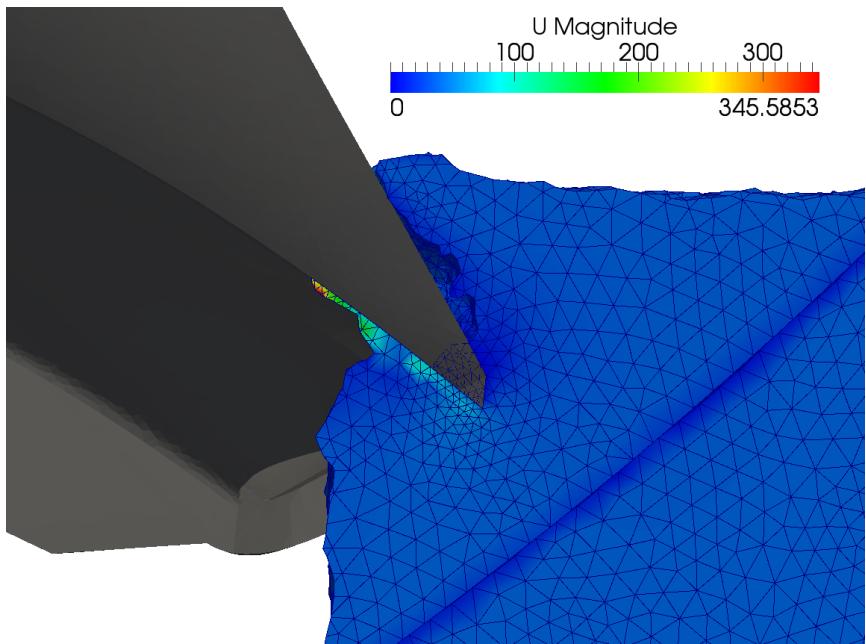
The uneven distribution of water in the buckets is illustrated in detail in Figure 4.9. It displays all cells with a volume fraction of water higher than 0.1, colored by the variation in the volume fraction (α_1). This is the same bucket and time step as in Figure 4.6d.

4.4 Computational time

The *AM3* simulation needed as many as 379 318 iterations to complete because of the small time step size. A plot of the time step size for *AM3* is found in Figure 4.10a. For most of the time steps, artificially high velocities were present for a small number of cells, directly affecting the Courant number and the time step size. The first written time step with an artificially high velocity is at the time 0.004, visualised in Figure 4.10b. The effect the high velocities have on the time step size is clearly seen in Figure 4.10a. The velocities on the back of the middle bucket at the time 0.016 is included in Figure 4.11, showing velocities at magnitudes of 10^4 .



(a) Time step δT for AM3.



(b) High velocity induced on the back of turbine bucket in AM3 at time 0.004.

Figure 4.10: Effect of high velocity on the back of bucket on the time step size.

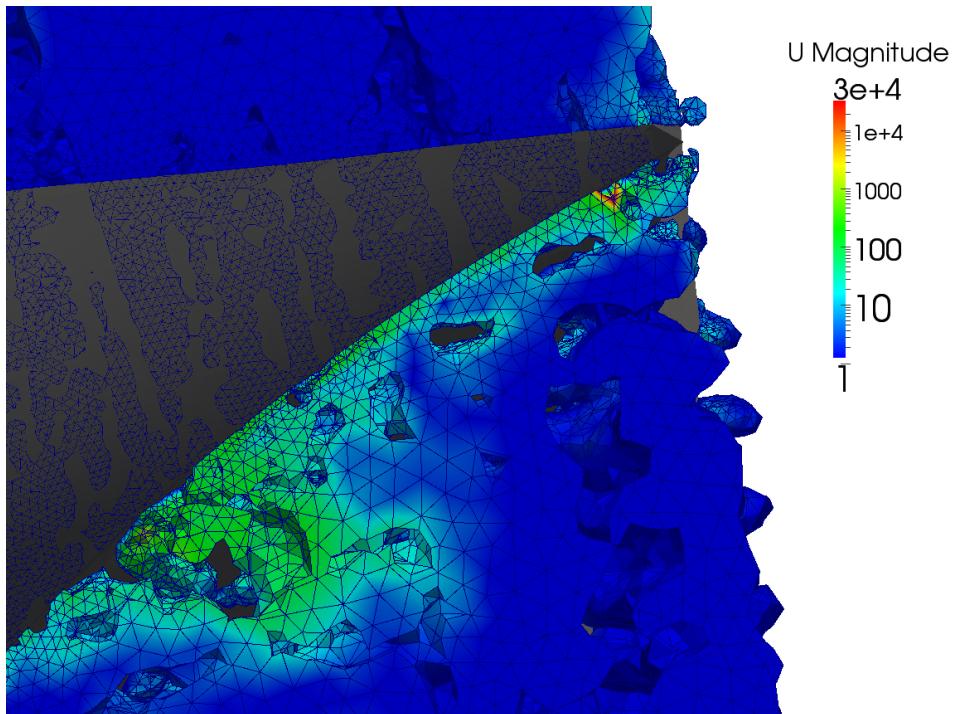


Figure 4.11: *AM3* high velocities at time 0.016. Note that a logarithmic scale is used.

Chapter 5

Discussion

The four simulations that produced a torque measurement will be discussed further in this chapter. The computational time will also be discussed as it has been a limitation during the work with this thesis.

5.1 Torque prediction

The reference curve gives a fairly realistic representation of the measured experimental torque in the model turbine, as validated by Barstad. The torque measurement curves from OpenFOAM are far from the reference curve generated with ANSYS CFX both in shape and magnitude, as can be seen in Figure 4.3.

The meshes for the simulations were made with two different approaches: with the OpenFOAM built-in tool *snappyHexMesh*, and with ANSYS Meshing. The torque curves generated by OpenFOAM illustrate a clear difference between the results of these two approaches. Firstly, *AM1*, *AM2*, and *AM3* gave unstable torque curves with large fluctuations, making it necessary to filter the data. Secondly, the *SHM1* torque curve did not need filtering. Additionally, it can be seen in Figure 4.1 and 4.2 that the *AM3* simulation, with twice as many cells as *AM2*, gave more oscillations around the smoothed torque curve than *AM2*. *AM1* produced an even cleaner torque measurement curve. Consequently, it seems that the noise in the torque measurement curve increases with mesh density.

The torque value of *AM3* in Figure 4.3, after time 0.010, is lower than that of *AM2*, which is again lower than that of *AM1*, and at the same time closer to the maximum reference torque curve. This could mean that the flow is better resolved in the denser mesh, even though the torque curve output is more unstable. The mesh generated with *snappyHexMesh*, *SHM1*, seems to perform quite well despite its mesh coarseness, giving an output torque measurement curve that is closer to the reference curve than any of the ANSYS meshes.

The time period that torque is produced in OpenFOAM seems to be both shorter and at an earlier time than in the reference simulation. This was first thought to be caused by a wrong rotation being applied, either because of an incorrectly specified rotational speed, or because of an error in the way the rotation was performed by the solver. However, the rotation was later verified to be correct, and there must for that reason be another cause. One possibility is that the backwash taking place in the turbine causes the steep decline in produced torque. To verify this possibility further investigation is necessary and it should be considered in a further study.

5.2 Convergence and stability

The residual target for the pressure equations is specified to be 1e-4. This is considered a relatively loose convergence, but is often sufficient. Using a residual target of 1e-5 or even 1e-6 might perhaps give better results, but would calculate slower. 1e-6 is used as the residual target for the velocity equations in this thesis as these are less time-consuming than the pressure loops. It could be interesting to see if lowering the residual target for the pressure equations would have any effect on the torque measurement, either with regards to the actual size or the stability of the results.

The residuals for *AM3* are plotted in Figure 4.4. The residuals do not exceed the specified tolerances as described in section 3.4.5, and are therefore converged to the desired levels. The *Interface Courant number*, in contrast, was not always below the specified value of 1.0, as shown in Figure 4.5. In *AM3*, the Interface Courant number is larger than 1.0 for 3.27 % of the time steps, possibly affecting the accuracy of the simulation results. This is believed to be caused by a limitation in how much the time step size can change for each iteration, a limitation possibly included to avoid large changes in the time step size between time steps. It seems that the Courant number has to be set lower than 1.0 to ensure that it is strictly bounded below this value, with the result of an even slower simulation.

5.3 Flow visualisation

With regards to the jet split and water leak across the AMI-patches, it is uncertain if this is a visual problem due to the post-processing interpolation across the AMI-patches in Paraview, or if the actual calculated flow is affected. However, taking into account Figure 4.7 and 4.8, the jet velocity and the volume fraction are not changed by the jet split. The shape of the jet seems to be constant across the jet AMI-patches. Hence, its influence on the results is most likely insignificant.

Figure 4.6e and 4.6f illustrate severe backwash caused by water accumulated between the buckets. In reality, most of the water should leave the turbine bucket during the cycle. The accumulation of water between buckets in the simulations

can explain the instabilities in torque measurement of *AM3* in Figure 4.2 from the time around 0.018. The accumulation of water might be caused by a too coarse mesh close to the bucket surfaces, causing the water to be influenced by the no-slip boundary condition of the wall. If this is the case, an inflation layer could be beneficial in the near-bucket area. A simulation implementing the inflation layer has been made, but was unfortunately not finished in time to be included in this thesis.

The unnatural shape of the jet, clearly shown in Figure 4.6d, but also in Figure 4.6e and 4.6f, can be explained by numerical diffusion due to the mesh coarseness above the top bucket. When comparing the above mentioned figures with Figure 3.3, which illustrates the mesh refinement in ANSYS, it can be observed that the mesh refinement due to the Body of Influence is no longer applied above the last bucket subject to the jet. Consequently, the flow is not properly resolved. When inspecting the results in Paraview it is found that the jet does not disappear as Figure 4.6d can give an impression of, but is instead spread to nearby cells in volume fractions lower than 0.5. The spreading of the flow is not visible in Figure 4.6 due to the low volume fraction of water in the cells.

The uneven distribution of water in the buckets, illustrated in Figure 4.9, is highly unnatural and does not resemble the flow in a real Pelton turbine. This effect might possibly be connected with the large amount of water not leaving the buckets in the simulations, resulting in backwash and most likely disturbing the torque measurement. Thus, the cause of this effect should be investigated in future studies.

As mentioned in section 3.3.3, the rotating-stationary mesh interface was moved closer to the turbine buckets than in the simulation by Barstad. Otherwise mostly the same refinement parameters were used. This does not seem to have been beneficial. The reason is that when the inner domain rotates relative to the outer domain, the mesh in the stationary domain just outside the turbine buckets becomes too coarse to resolve any flow that crosses the rotating-stationary mesh interface. Normally, water leaving the buckets should not cross this interface, but as the interface is very close to the flow in the turbine buckets it might cause diffusion and disturb the flow. More importantly, having the rotating-stationary mesh interface too close to the buckets makes it difficult to visually inspect the flow close to the buckets, as the jet split across the AMI-patches gives an unrealistic visualisation.

5.4 Computational time

The computational time has been an important issue while working with this thesis. The simulation *AM3*, with a mesh size just above the mesh independence level defined by Barstad, took almost 19 000 core hours to finish on the supercomputer *Vilje*. For comparison, the reference CFX simulation needed about 667 core hours on a 12 core *Intel Xeon* computer. In an implicit code such as ANSYS CFX, simulations can be run with Courant numbers higher than 1.0, as described in

section 2.3.3. This explains the difference in time step size between OpenFOAM and ANSYS CFX, but at the same time, each time step should calculate faster in OpenFOAM.

Several changes have been made to lower the simulation time, mainly adjusting the numerical schemes and settings for the linear solvers and the pressure-velocity coupling. However, the main parameter limiting the simulation speed is the Courant number. For unknown reasons local velocities of the magnitude 10^4 are induced in the boundary layers around the turbine buckets, thus reducing the time steps to magnitudes down to 10^{-9} to fulfil the *CFL*-criterion.

Chapter 6

Conclusion and Further Work

The aim of this thesis has been to develop a method for predicting the torque applied to a Pelton turbine by a high speed water jet using the Open Source software OpenFOAM. A method has been developed and tested, combining the capabilities of multiphase and dynamic mesh handling of *interDyMFoam* with the Arbitrary Mesh Interface (AMI) boundary condition. The built-in *forces* function is used to measure the torque applied to one of the turbine buckets. The measured torque was significantly larger than both the torque measured with the ANSYS CFX method of Barstad and the experimental torque. Additionally, the torque measurement curve from OpenFOAM contained instabilities and did not coincide well with the one generated in ANSYS CFX. The measured maximum torque of the model seemed to go towards the actual solution when the density of the mesh increased, but at the same time it gave more noise in the output, making smoothing of the results necessary.

A visual inspection of the simulations showed that water accumulates between the buckets, causing severe backwash. This is most likely one of the sources of the instabilities in the torque measurement, and can be caused by insufficient mesh density in close-bucket areas. Furthermore, the water flow in the buckets is unnatural with an uneven distribution of the volume fraction.

The simulations have demanded a large amount of computational resources, thus limiting the development and testing of the model. The reason seems to be artificially large velocities that are induced in several areas close to the buckets, resulting in a drop in the time step size to magnitudes as low as 10^{-9} in order to fulfil the Courant Friedrich Levy (CFL) criteria. With the mesh density level found by Barstad to be mesh independent, the method developed in this thesis needed around 19 000 core hours to finish, almost thirty times that needed by ANSYS CFX.

The computational cost of the simulations has been the largest limitation in the work with this thesis, and illustrates the need for further development of the method. The first priority in further studies of the method should be to make it more efficient with regards to simulation time. Running the simulations, the time step size has been low due to certain artificially high local velocities in the close-bucket areas, resulting in high computational costs. These should be investigated further to see what is causing them. Investigation of the discretisation schemes and linear solvers can also be beneficial to improve both speed and accuracy. Additionally, the accumulation and uneven distribution of water in the turbine buckets will need to be investigated.

Continued development of the method could perhaps benefit from an approach not utilising any sliding-grid interface, thus saving the time used for interpolation between two meshes. By doing so, it may be possible to make a smaller domain in order to reduce the total number of cells, while still maintaining a high mesh density in the area around the buckets.

Bibliography

- [1] American Institute of Aeronautics and Astronautics [1998], *Guide for the verification and validation of computational fluid dynamics simulations*.
- [2] ANSYS, Inc [2009], *ANSYS CFX-Solver Modeling Guide 12*.
- [3] ANSYS, Inc. [2011], *ANSYS CFX-Solver Theory Guide*.
- [4] ANSYS, Inc. [2012], *Introduction to ANSYS CFX, Lecture 5, Solver Settings and Output File*.
- [5] Barstad, L. F. [2012], CFD-analysis of a Pelton turbine, Master's thesis, Norwegian University of Science and Technology.
- [6] Brekke, H. [2000], *Grunnkurs i hydrauliske strømningsmaskiner*, Vannkraftlaboratoriet, NTNU.
- [7] Brekke, H. [2003], *Pumper og turbiner: kompendium*, Tapir akademisk forlag, Kompendieforlaget.
- [8] CFD Online [n.d.], ‘Best practice guidelines for turbomachinery CFD’.
URL: http://www.cfd-online.com/Wiki/Best-practice_guidelines_for_turbomachinery_CFD
- [9] CFD Online Discussion Forums [2011], ‘Numerical schemes for free surface flows (VOF)’.
URL: <http://www.cfd-online.com/Forums/openfoam/85999-numerical-schemes-free-surface-flows-vof.html>
- [10] CFD Online Discussion Forums [2012], ‘Does PISO algorithm work with interfoam in openFOAM 2.1.0?’.
URL: <http://www.cfd-online.com/Forums/openfoam-solving/103183-does-piso-algorithm-work-interfoam-openfoam-2-1-0-a.html>
- [11] Ferziger, J. H. and Peric, M. [2002], *Computational methods for fluid dynamics*, number 3rd, rev. ed., Springer.
- [12] Grein, H., Klicov, D. and Wieser, W. [1988], ‘Efficiency scale effects in Pelton turbines’, *Water Power & Dam Construction* **40**(5), 32–36.

- [13] Håkan Nilsson, *Personal correspondence* [2012].
- [14] Hana, M. [1999], Numerical analysis of non-stationary free surface flow in a Pelton bucket, PhD thesis, Norwegian University of Science and Technology.
- [15] International Electrotechnical Commission [1999], *Hydraulic turbines, storage pumps and pump-turbines: model acceptance tests*, number 2nd ed., International Electrotechnical Commission.
- [16] Klemetsen, L. E. [2010], An experimental and numerical study of the free surface Pelton bucket flow, Master's thesis, Norwegian University of Science and Technology.
- [17] Kristoffersen, R. [2012], 'TEP4545 Course in Computational Fluid Dynamics'.
- [18] Menter, F. R. [1994], 'Two-Equation Eddy-Viscosity Turbulence Models for Engineering Applications', *AIAA Journal* **32**(8), 1598–1605.
- [19] Nielsen, T. K. [2012], 'Lecture notes TEP4200 - Mechanical Design, Operation and Maintenance of Hydraulic Machinery'.
- [20] NTNU HPC Group [2012], *Vilje User Guide*.
URL: <http://www.hpc.ntnu.no/display/hpc/OpenFOAM+-+Performance+on+Vilje>
- [21] Olimstad, G. [2009], 3D Simulering av pumpeturbinkarakteristikker, Master's thesis, NTNU.
- [22] OpenFOAM Foundation [2012], *OpenFOAM User Guide, Version 2.1.1*.
- [23] OpenFOAM Foundation [n.d.], *OpenFOAM C++ Documentation*.
URL: <http://openfoam.org/docs/cpp/>
- [24] Perrig, A. [2007], Hydrodynamics of the Free Surface Flow in Pelton Turbine Buckets, PhD thesis, Ecole Polytechnique Federale de Lausanne (EPFL).
- [25] Rygg, J. R. [2012], 'CFD analysis of a Pelton turbine, Project thesis'.
- [26] The OpenFOAM Extend Project [n.d.], 'The OpenFOAM Extend Project website'.
URL: <http://www.extend-project.de/>
- [27] The OpenFOAM Foundation [2012], 'The OpenFOAM Foundation website'.
URL: <http://www.openfoam.org/>
- [28] *Unofficial OpenFOAM wiki website* [n.d.].
URL: http://openfoamwiki.net/index.php/Main_Page
- [29] Versteeg, H. K. and Malalasekera, W. [2007], *An introduction to computational fluid dynamics*, number 2nd ed., Pearson/Prentice Hall.
- [30] White, F. M. [2006], *Viscous fluid flow*, number 3rd ed., McGraw-Hill Higher Education.

Appendices

Appendix A

snappyHexMesh Workflow

The workflow for creating a mesh using *snappyHexMesh* is described in this appendix. Example dictionaries *blockMeshDict* and *snappyHexMeshDict* are attached in this appendix.

The method for creating a mesh using *snappyHexMesh* can roughly be described as follows:

1. Prepare surface data files in OBJ format
2. Create a background mesh using *blockMesh*
3. Prepare the *snappyHexMeshDict* dictionary
4. Run the meshing script
5. Manipulate the mesh to create the AMI-interface

A.1 Preparing the geometry

The geometries are scaled to meters and converted to the *Wavefront .obj* format using the command:

```
surfaceTransformPoints constant/triSurface/*.stl constant/triSurface/*.obj -scale  
'(0.001 0.001 0.001)'
```

A.2 Creation of the background mesh

A simple mesh is created using the *blockMesh*-tool to define the extent of the computational domain. The surfaces will later be subtracted from this domain,

and it should be somewhat similar to, but not necessarily equal to, the desired domain geometry.

The background mesh is defined in *blockMeshDict* by specifying vertices, blocks and edges. The patches are defined by specifying the number of vertices. When specifying faces of the patches it is essential to number the vertices in the correct order. More information can be found in section 5.3 of the *OpenFOAM User Guide*[22].

The *blockMeshDict* used for creating the background mesh can be seen in appendix A.6.

A.3 Configuring snappyHexMesh

Using *snappyHexMesh* is an iterative process, and it normally takes some time to get a working mesh. One should start with modifying the required parameters, and then tuning it afterwards to improve the mesh. An example *snappyHexMeshDict* can be found in appendix A.7. The most important steps to start with are listed below.

1. Turn on/off *snappyHexMesh* steps. These can be run separate for greater control of the process.
2. Specify surfaces (.obj-files) to be used in the mesh
3. Set the maximum limit of mesh size in *maxGlobalCells*.
4. Specify refinement for cells intersected by geometry edges
5. Specify a minimum and maximum refinement for cells intersected by geometry surfaces
6. Specify *refinementRegions* by specifying a distance from surface and refinement level. Refinement around the buckets is essential. Multiple refinement levels can be specified for a single surface by listing the highest refinement level first, for instance:
mode distance;
levels ((0.001 6) (0.01 5));
7. Specify a point for *locationInMesh* that is inside the desired domain, but not on a cell face.

The refinement level is specified as a number, with a higher number giving more refinement, and 0 being the level of the base mesh specified in *blockMeshDict*.

A.4 Improving the mesh

The generated mesh has to be inspected both by using the built-in *checkMesh* application and visually. The *snappyHexMesh* log file also provides useful info about the mesh generation process. Any error in the *snappyHexMesh* log file or from running *checkMesh* will have to be corrected in order to have a good mesh for simulations. Improving the mesh is a time consuming process that requires tuning the dictionary and inspecting the generated mesh several times.

snappyHexMesh will stop refining the mesh when the cell number reaches the *maxGlobalCells* parameter, thus this will have to be in accordance with the specified refinement levels. If the level of refinement is increased, the number of *maxGlobalCells* should also be increased, or the refinement will not be as specified by the user.

When specifying *refinementRegions* it is important that the specified refinement areas do not overlap, as there will be a conflict that *snappyHexMesh* handles poorly. The user will not receive any error message, but it is likely that the area will not be refined as desired.

The *meshQualityControls* are important for the quality of the generated mesh, and should be adjusted if *checkMesh* results in errors. It is wise to change these carefully while investigating the effect of a stricter or looser tolerance. The mesh generation will most likely fail or give larger errors in *checkMesh* if the parameters are set to a very strict level.

A.5 Define AMI-interface patches

1. Delete empty patches (*nFaces=0*) from the background mesh. Remember to change the number of patches in the start of the file
2. Copy the endFace-ID from *constant/polymesh/boundary*. It is found by adding the values of *nFaces* and *startFace* of the last boundary.
3. Paste this value in the *startFace*-field for the AMI-patches in *system/changeDictionaryDict*
4. Update *system/createAMIFaces.topoSetDict* to correspond with the rotating and stationary domain. *rotif* refers to the rotating domain, *rotifFace* to the sliding interface between the rotating and the stationary domain, and *outerCells* refers to the stationary domain.
5. Update *removeRedundantZones.topoSetDict* to remove the *cellZone rotif* that is no longer needed
6. Update *Allrun2.pre* to generate the AMI-patches from *rotif*
7. Update the fields in the *0*-folder to be in accordance with *constant/polymesh/boundary*
8. Run *Allrun2.pre*

9. Delete the empty *rotif*-patch. Remember to change the number of patches in the start of the file.

A.6 *blockMeshDict* example

```
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    object       blockMeshDict;
}

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

convertToMeters 0.001;

vertices
(
    (-470 139.08 0) //0
    (0 139.08 -470)
    (470 139.08 0)
    (0 139.08 470)
    (-173.85 139.08 0)
    (0 139.08 -173.85)
    (173.85 139.08 0)
    (0 139.08 173.85)
    (-470 0 0) //8
    (0 0 -470)
    (470 0 0)
    (0 0 470)
    (-173.85 0 0)
    (0 0 -173.85)
    (173.85 0 0)
    (0 0 173.85) //15
);

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

```

```

edges
(
arc 1 0 (-332.34 139.08 -332.34)
arc 0 3 (-332.34 139.08 332.34)
arc 3 2 (332.34 139.08 332.34)
arc 2 1 (332.34 139.08 -332.34)
arc 9 8 (-332.34 0 -332.34) //5
arc 8 11 (-332.34 0 332.34)
arc 11 10 (332.34 0 332.34)
arc 10 9 (332.34 0 -332.34)

//Inner circle
arc 5 4 (-122.93 139.08 -122.93)
arc 4 7 (-122.93 139.08 122.93)
arc 7 6 (122.93 139.08 122.93)
arc 6 5 (122.93 139.08 -122.93)
arc 13 12 (-122.93 0 -122.93)
arc 12 15 (-122.93 0 122.93)
arc 15 14 (122.93 0 122.93)
arc 14 13 (122.93 0 -122.93)
);

boundary
(
    outside
    {
        type patch;
        faces
        (
            (0 1 9 8)
            (1 2 10 9)
            (2 3 11 10)
            (3 0 8 11)
        );
    }
    inside
    {
        type patch;
        faces
        (
            (5 4 12 13)
            (4 7 15 12)
            (7 6 14 15)
            (6 5 13 14)
    }
)

```

```

        );
    }
    top
    {
        type patch;
        faces
        (
            (0 4 5 1)
            (1 5 6 2)
            (2 6 7 3)
            (3 7 4 0)
        );
    }
    bottom
    {
        type wall;
        faces
        (
            (8 9 13 12)
            (9 10 14 13)
            (10 11 15 14)
            (11 8 12 15)
        );
    }
};

// ****

```

A.7 Example *snappyHexMeshDict*

```

FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    object       snappyHexMeshDict;
}
// ****

// Which of the steps to run
castellatedMesh true;
snap          true;
addLayers     true;

```

```

// Geometry. Definition of all surfaces. All surfaces are of class
// searchableSurface.
// Surfaces are used
// - to specify refinement for any mesh cell intersecting it
// - to specify refinement for any mesh cell inside/outside/near
// - to 'snap' the mesh boundary to the surface
geometry
{
    jetinlet.obj
    {
        type      triSurfaceMesh;
        name      jetinlet;
        regions
        {
            jetinlet
            {
                name      jetinlet;
            }
        }
    }

    jetwalls.obj
    {
        type      triSurfaceMesh;
        name      jetwalls;
        regions
        {
            jetwalls
            {
                name      jetwalls;
            }
        }
    }

    BBC.obj
    {
        type      triSurfaceMesh;
        name      BBC;
        regions
        {
            BBC
            {
                name      BBC;
            }
        }
    }
}

```

```

        }
    }

MBC.obj
{
    type      triSurfaceMesh;
    name      MBC;
    regions
    {
        MBC
        {
            name      MBC;
        }
    }
}

TBC.obj
{
    type      triSurfaceMesh;
    name      TBC;
    regions
    {
        TBC
        {
            name      TBC;
        }
    }
}

statopening.obj
{
    type      triSurfaceMesh;
    name      statopening;
    regions
    {
        statopening
        {
            name      statopening;
        }
    }
}

rotif.obj
{
    type      triSurfaceMesh;
    name      rotif;
}

```

```

regions
{
    rotif
    {
        name      rotif;
    }
}
rotopen.obj
{
    type      triSurfaceMesh;
    name      rotopen;
    regions
    {
        rotopen
        {
            name      rotopen;
        }
    }
}
rotsym.obj
{
    type      triSurfaceMesh;
    name      rotsym;
    regions
    {
        rotsym
        {
            name      rotsym;
        }
    }
}
statinlet.obj
{
    type      triSurfaceMesh;
    name      statinlet;
    regions
    {
        statinlet
        {
            name      statinlet;
        }
    }
}

```

```

statsym.obj
{
    type      triSurfaceMesh;
    name      statsym;
    regions
    {
        statsym
        {
            name      statsym;
        }
    }
}

//;

// Settings for the castellatedMesh generation.
castellatedMeshControls
{
    // Refinement parameters
    // ~~~~~

    // If local number of cells is >= maxLocalCells on any processor
    // switches from refinement followed by balancing
    // (current method) to (weighted) balancing before refinement.
    maxLocalCells 100000;

    // Overall cell limit (approximately). Refinement will stop immediately
    // upon reaching this number so a refinement level might not complete.
    // Note that this is the number of cells before removing the part which
    // is not 'visible' from the keepPoint. The final number of cells might
    // actually be a lot less.
    maxGlobalCells 500000;

    // The surface refinement loop might spend lots of iterations refining just a
    // few cells. This setting will cause refinement to stop if <= minimumRefine
    // are selected for refinement. Note: it will at least do one iteration
    // (unless the number of cells to refine is 0)
    minRefinementCells 0;

    // Allow a certain level of imbalance during refining
    // (since balancing is quite expensive)
    // Expressed as fraction of perfect balance (= overall number of cells /

```

```

// nProcs). 0=balance always.
maxLoadUnbalance 0.10;

// Number of buffer layers between different levels.
// 1 means normal 2:1 refinement restriction, larger means slower
// refinement.
nCellsBetweenLevels 2;

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

// Specifies a level for any cell intersected by its edges.
// This is a featureEdgeMesh, read from constant/triSurface for now.
features
(
    {
        file      "jetinlet.eMesh";
        level     3;
    }

    {
        file      "jetwalls.eMesh";
        level     3;
    }

    {
        file      "statopening.eMesh";
        level     1;
    }

    {
        file      "rotif.eMesh";
        level     3;
    }

    {
        file      "BBC.eMesh";
        level     3;
    }

    {
        file      "MBC.eMesh";
        level     3;
    }
)

```

```

    {
        file      "TBC.eMesh";
        level     3;
    }
{
    file      "rotopen.eMesh";
    level     1;
}
{
    file      "rotsym.eMesh";
    level     1;
}
{
    file      "statinlet.eMesh";
    level     1;
}
{
    file      "statsym.eMesh";
    level     1;
}

);

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

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

refinementSurfaces
{
    BBC
    {
        level      (2 3);
    }
    MBC
    {
        level      (2 3);
    }
    TBC
}

```

```

{
    level      (2 3);
}
rotif
{
    level      (3 3);
    cellZone   rotif;
    faceZone   rotif;
    cellZoneInside inside;
}
statopening
{
    level      (1 1);
}
jetinlet
{
    level      (2 2);
}
jetwalls
{
    level      (2 2);
}
rotopen
{
    level      (2 2);
}
rotsym
{
    level      (4 4);
}
statinlet
{
    level      (1 1);
}
statsym
{
    level      (4 4);
}
}

// Resolve sharp angles
resolveFeatureAngle 30;

// Region-wise refinement

```

```

// ~~~~~

// Specifies refinement level for cells in relation to a surface. One of
// three modes
// - distance. 'levels' specifies per distance to the surface the
//   wanted refinement level. The distances need to be specified in
//   descending order.
// - inside. 'levels' is only one entry and only the level is used. All
//   cells inside the surface get refined up to the level. The surface
//   needs to be closed for this to be possible.
// - outside. Same but cells outside.

refinementRegions
{

MBC
{
    mode      distance;
    levels   ((0.03 4));
}
TBC
{
    mode      distance;
    levels   ((0.03 4));
}
BBC
{
    mode      distance;
    levels   ((0.03 4));
}

jetwalls
{
    mode      distance;
    levels   ((0.01 4));
}

jetinlet
{
    mode      distance;
    levels   ((0.01 5));
}

}

```

```

// Mesh selection
// ~~~~~

// After refinement patches get added for all refinementSurfaces and
// all cells intersecting the surfaces get put into these patches. The
// section reachable from the locationInMesh is kept.
// NOTE: This point should never be on a face, always inside a cell, even
// after refinement.
// locationInMesh (21 601.1 211.1);
locationInMesh (-0.20545645 0.111378 0.202156456);

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

// Settings for the snapping.
snapControls
{
    // Number of patch smoothing iterations before finding correspondence
    // to surface
    nSmoothPatch 5;

    // Relative distance for points to be attracted by surface feature point
    // or edge. True distance is this factor times local
    // maximum edge length.
    tolerance 4.0;

    // Number of mesh displacement relaxation iterations.
    nSolveIter 20;

    // Maximum number of snapping relaxation iterations. Should stop
    // before upon reaching a correct mesh.
    nRelaxIter 6;

    // Highly experimental and wip: number of feature edge snapping
    // iterations. Leave out altogether to disable.
    // Do not use here since mesh resolution too low and baffles present
    nFeatureSnapIter 20;
}

```

```

// Settings for the layer addition.
addLayersControls
{
    // Are the thickness parameters below relative to the undistorted
    // size of the refined cell outside layer (true) or absolute sizes (false).
    relativeSizes true;

    // Per final patch (so not geometry!) the layer information
    layers
    {
    }

    // Expansion factor for layer mesh
    expansionRatio 1.0;

    //-- Wanted thickness of final added cell layer. If multiple layers
    // is the
    // thickness of the layer furthest away from the wall.
    // Relative to undistorted size of cell outside layer.
    // is the thickness of the layer furthest away from the wall.
    // See relativeSizes parameter.
    finalLayerThickness 0.3;

    //-- Minimum thickness of cell layer. If for any reason layer
    // cannot be above minThickness do not add layer.
    // Relative to undistorted size of cell outside layer.
    minThickness 0.1;

    //-- If points get not extruded do nGrow layers of connected faces that are
    // also not grown. This helps convergence of the layer addition process
    // close to features.
    // Note: changed(corrected) w.r.t 17x! (didn't do anything in 17x)
    nGrow 0;

    // Advanced settings

    //-- When not to extrude surface. 0 is flat surface, 90 is when two faces
    // make straight angle.
    featureAngle 30;

    //-- Maximum number of snapping relaxation iterations. Should stop
    // before upon reaching a correct mesh.
    nRelaxIter 3;

    // Number of smoothing iterations of surface normals

```

```

nSmoothSurfaceNormals 1;

// Number of smoothing iterations of interior mesh movement direction
nSmoothNormals 3;

// Smooth layer thickness over surface patches
nSmoothThickness 10;

// Stop layer growth on highly warped cells
maxFaceThicknessRatio 0.5;

// Reduce layer growth where ratio thickness to medial
// distance is large
maxThicknessToMedialRatio 0.3;

// Angle used to pick up medial axis points
// Note: changed(corrected) w.r.t 17x! 90 degrees corresponds to 130 in 17x
minMedianAxisAngle 90;

// Create buffer region for new layer terminations
nBufferCellsNoExtrude 0;

// Overall max number of layer addition iterations. The mesher will exit
// if it reaches this number of iterations; possibly with an illegal
// mesh.
nLayerIter 50;
}

// Generic mesh quality settings. At any undoable phase these determine
// where to undo.
meshQualityControls
{
    // Maximum non-orthogonality allowed. Set to 180 to disable.
    maxNonOrtho 65;

    // Max skewness allowed. Set to <0 to disable.
    maxBoundarySkewness 20;
    maxInternalSkewness 4;

    // Max concaveness allowed. Is angle (in degrees) below which concavity
}

```

```

// is allowed. 0 is straight face, <0 would be convex face.
// Set to 180 to disable.
maxConcave 80;

// Minimum pyramid volume. Is absolute volume of cell pyramid.
// Set to a sensible fraction of the smallest cell volume expected.
// Set to very negative number (e.g. -1E30) to disable.
minVol 1e-13;

// Minimum quality of the tet formed by the face-centre
// and variable base point minimum decomposition triangles and
// the cell centre. This has to be a positive number for tracking
// to work. Set to very negative number (e.g. -1E30) to
// disable.
//   <0 = inside out tet,
//     0 = flat tet
//     1 = regular tet
minTetQuality -1;

// Minimum face area. Set to <0 to disable.
minArea -1;

// Minimum face twist. Set to <-1 to disable. dot product of face normal
// and face centre triangles normal
minTwist 0.01;

// minimum normalised cell determinant
// 1 = hex, <= 0 = folded or flattened illegal cell
minDeterminant 0.001;

// minFaceWeight (0 -> 0.5)
minFaceWeight 0.05;

// minVolRatio (0 -> 1)
minVolRatio 0.01;

//must be >0 for Fluent compatibility
minTriangleTwist -1;

// Advanced

// Number of error distribution iterations
nSmoothScale 4;
// amount to scale back displacement at error points

```

```

errorReduction 0.75;

// Optional : some meshing phases allow usage of relaxed rules.
// See e.g. addLayersControls::nRelaxedIter.
relaxed
{
    //-- Maximum non-orthogonality allowed. Set to 180 to disable.
    maxNonOrtho 75;
}

// Advanced

// Flags for optional output
// 0 : only write final meshes
// 1 : write intermediate meshes
// 2 : write volScalarField with cellLevel for postprocessing
// 4 : write current intersections as .obj files
debug 0;

// Merge tolerance. Is fraction of overall bounding box of initial mesh.
// Note: the write tolerance needs to be higher than this.
mergeTolerance 1e-6;

// ****

```


Appendix B

OpenFOAM case setup

Some of the case files for the simulations in OpenFOAM are included in this appendix. These are the settings used for one of the simulations with a mesh generated in ANSYS Meshing. The settings for the snappyHexMesh simulations are similar, but with a slightly different naming. Additional case files for both approaches are delivered together with this thesis and should be used for further reference.

B.1 Velocity field U

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0";
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0];
internalField   uniform (0 0 0);

boundaryField
{
    bb
```

```

{
type fixedValue;
value uniform (0 0 0);
}
bbu
{
type fixedValue;
value uniform (0 0 0);
}
mb
{
type fixedValue;
value uniform (0 0 0);
}
mbu
{
type fixedValue;
value uniform (0 0 0);
}
rotopen
{
type      pressureInletOutletVelocity;
value      uniform (0 0 0);
}
rotsym
{
type      symmetryPlane;
}
tb
{
type fixedValue;
value uniform (0 0 0);
}
tbu
{
type fixedValue;
value uniform (0 0 0);
}
jetinlet
{
type fixedValue;
value uniform (-38.38 0 0);
}
jetwalls
{

```

```

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

statopen
{
type      pressureInletOutletVelocity;
value      uniform (0 0 0);
}

AMI1
{
type cyclicAMI;
value uniform (0 0 0);
}
AMI2
{
type cyclicAMI;
value uniform (0 0 0);
}
}

// ****

```

B.2 Dynamic pressure field p_{rgh}

```

FoamFile
{
version      2.0;
format       ascii;
class        volScalarField;
location     "0";
object       p_rgh;
}
// *****

dimensions    [1 -1 -2 0 0 0 0];

internalField  uniform 0;

boundaryField

```

```

{

bb
{
    type      zeroGradient;
}
bbu
{
    type      zeroGradient;
}
mb
{
    type      zeroGradient;
}
mbu
{
    type      zeroGradient;
}
rotopen
{
    type      totalPressure;
    p0        uniform 0;
    U         U;
    phi       phi;
    rho       rho;
    psi       none;
    gamma    1;
    value    uniform 0;
}
rotsym
{
    type      symmetryPlane;
}
tb
{
    type      zeroGradient;
}
tbu
{
    type      zeroGradient;
}
jetinlet
{
    type      zeroGradient;
}

```

```

}

jetwalls
{
    type          zeroGradient;
}

statopen
{
    type          totalPressure;
    p0            uniform 0;
    U             U;
    phi           phi;
    rho           rho;
    psi           none;
    gamma         1;
    value          uniform 0;
}

AMI1
{
type cyclicAMI;
value uniform 0;
}
AMI2
{
type cyclicAMI;
value uniform 0;
}

// ****

```

B.3 Phase fraction field *alpha1*

```

FoamFile
{
    version      2.0;
    format       ascii;
    class        volScalarField;
    location     "0";
    object       alpha1.org;
}
// ****

```

```

dimensions      [0 0 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    bb
    {
        type          zeroGradient;
    }
    bbu
    {
        type          zeroGradient;
    }
    mb
    {
        type          zeroGradient;
    }
    mbu
    {
        type          zeroGradient;
    }
    rotopen
    {
        type          inletOutlet;
        inletValue    uniform 0;
        value         uniform 0;
    }
    rotsym
    {
        type          symmetryPlane;
    }
    tb
    {
        type          zeroGradient;
    }
    tbu
    {
        type          zeroGradient;
    }
    jetinlet
    {
        type          fixedValue;
    }
}

```

```

value uniform 1;
}
jetwalls
{
    type          zeroGradient;
}

statopen
{
    type          inletOutlet;
    inletValue    uniform 0;
    value         uniform 0;
}

AMI1
{
type cyclicAMI;
    value        uniform 0;
}
AMI2
{
type cyclicAMI;
    value        uniform 0;
}

}

// ****

```

B.4 fvSchemes

```

FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;

```

```

}

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

ddtSchemes
{
    default      Euler;
}

gradSchemes
{
    default      Gauss linear;
}

divSchemes
{
    div(rho*phi,U)  Gauss limitedLinearV 1;
    div(phi,alpha)  Gauss vanLeer01;
    div(phirb,alpha) Gauss interfaceCompression;
    div((muEff*dev(T(grad(U))))) Gauss linear;
    div((nuEff*dev(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    default      Gauss linear limited 1.0;
}

interpolationSchemes
{
    default      linear;
}

snGradSchemes
{
    default      limited 1.0;
}

fluxRequired
{
    default      no;
    p_rgh;
    pcorr;
    alpha;
}

```

XXX

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

B.5 fvSolution

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}

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

solvers
{
    pcorr
    {
        solver          GAMG;
        tolerance       1e-04;
        relTol          0;
        smoother        DIC;
        nPreSweeps      0;
        nPostSweeps     2;
        nFinestSweeps   2;
        cacheAgglomeration true;
        nCellsInCoarsestLevel 10;
        agglomerator    faceAreaPair;
        mergeLevels     1;
    }

    p_rgh
    {
        solver          GAMG;
        tolerance       1e-04;
        relTol          0;
        smoother        DIC;
        nPreSweeps      0;
        nPostSweeps     2;
        nFinestSweeps   2;
        cacheAgglomeration true;
        nCellsInCoarsestLevel 10;
        agglomerator    faceAreaPair;
    }
}
```

```

        mergeLevels      1;
    }

p_rghFinal
{
    solver          GAMG;
    tolerance       1e-04;
    relTol          0;
    smoother         DIC;
    nPreSweeps      0;
    nPostSweeps     2;
    nFinestSweeps   2;
    cacheAgglomeration true;
    nCellsInCoarsestLevel 10;
    agglomerator    faceAreaPair;
    mergeLevels      1;
}

U
{
    solver          smoothSolver;
    smoother         GaussSeidel;
    tolerance       1e-06;
    relTol          0;
    nSweeps         1;
}

UFinal
{
    solver          smoothSolver;
    smoother         GaussSeidel;
    tolerance       1e-06;
    relTol          0;
    nSweeps         1;
}

}

PIMPLE
{
    momentumPredictor yes;
    nCorrectors      4;
    nNonOrthogonalCorrectors 3;
    nAlphaCorr       1;
    nAlphaSubCycles 3;
}

```

```

    cAlpha          1.5;
    correctPhi     no;

}

relaxationFactors
{
    fields
    {
    }
    equations
    {
        "U.*"           1;
    }
}

//Initialization with potentialFoam
potentialFlow
{
    nNonOrthogonalCorrectors 10;
}

// ****

```

B.6 transportProperties

```

FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// ****

nu          nu [ 0 2 -1 0 0 0 0 ] 1e-06;

phase1
{
    transportModel Newtonian;
    nu            nu [ 0 2 -1 0 0 0 0 ] 1e-06;
    rho           rho [ 1 -3 0 0 0 0 0 ] 1000;
    CrossPowerLawCoeffs

```

```

{
    nu0          nu0 [ 0 2 -1 0 0 0 0 ] 1e-06;
    nuInf       nuInf [ 0 2 -1 0 0 0 0 ] 1e-06;
    m           m [ 0 0 1 0 0 0 0 ] 1;
    n           n [ 0 0 0 0 0 0 0 ] 0;
}

BirdCarreauCoeffs
{
    nu0          nu0 [ 0 2 -1 0 0 0 0 ] 0.0142515;
    nuInf       nuInf [ 0 2 -1 0 0 0 0 ] 1e-06;
    k            k [ 0 0 1 0 0 0 0 ] 99.6;
    n           n [ 0 0 0 0 0 0 0 ] 0.1003;
}
}

phase2
{
    transportModel Newtonian;
    nu            nu [ 0 2 -1 0 0 0 0 ] 1.48e-05;
    rho          rho [ 1 -3 0 0 0 0 0 ] 1;
    CrossPowerLawCoeffs
    {
        nu0          nu0 [ 0 2 -1 0 0 0 0 ] 1e-06;
        nuInf       nuInf [ 0 2 -1 0 0 0 0 ] 1e-06;
        m           m [ 0 0 1 0 0 0 0 ] 1;
        n           n [ 0 0 0 0 0 0 0 ] 0;
    }
}

BirdCarreauCoeffs
{
    nu0          nu0 [ 0 2 -1 0 0 0 0 ] 0.0142515;
    nuInf       nuInf [ 0 2 -1 0 0 0 0 ] 1e-06;
    k            k [ 0 0 1 0 0 0 0 ] 99.6;
    n           n [ 0 0 0 0 0 0 0 ] 0.1003;
}
}

sigma          sigma [ 1 0 -2 0 0 0 0 ] 0.0728;

// ****

```

B.7 dynamicMeshDict

```
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    location     "constant";
    object       dynamicMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dynamicFvMesh    solidBodyMotionFvMesh;

motionSolverLibs ( "libfvMotionSolvers.so" );

solidBodyMotionFvMeshCoeffs
{
    cellZone      rotif;

    solidBodyMotionFunction  rotatingMotion;
    rotatingMotionCoeffs
    {
        CofG          (0 0 0);
        radialVelocity (0 4595 0 );
    }
}

// **** //
```


Appendix C

Scripts for mesh generation and manipulation

Attached here are the scripts used for mesh generation and manipulation. The first script is used to generate meshes with snappyHexMesh when all the parameters and geometries are prepared, and the second script is used to create the AMI-patches and the cell zones for the dynamic mesh.

C.1 snappyHexMex execution - *Allrun1.pre*

This script controls the mesh generation process of snappyHexMesh, starting with preparing the background mesh, extracting the geometries and executing snappy-HexMesh.

```
#!/bin/sh
cd ${0%/*} || exit 1      # run from this directory

# Source tutorial run functions
. $WM_PROJECT_DIR/bin/tools/RunFunctions

# - meshing
runApplication blockMesh

surfaces="
BBC
MBC
```

```

TBC
rotif
rotopen
rotsym
jetinlet
jetwalls
statinlet
statopening
statsym
"
for s in $surfaces; do
    runApplication surfaceFeatureExtract -includedAngle 150 -minElem 10 \
        constant/triSurface/$s.obj $s
    mv log.surfaceFeatureExtract log.surfaceFeatureExtract.$s
done

runApplication snappyHexMesh -overwrite

```

C.2 Mesh manipulation - *Allrun2.pre*

```

#!/bin/sh
cd ${0%/*} || exit 1      # run from this directory

# Source tutorial run functions
. $WM_PROJECT_DIR/bin/tools/RunFunctions

# - generate face/cell sets and zones

#runApplication setSet -batch removeRedundantZones.setSet
#mv log.setSet log.removeRedundantZones.setSet
runApplication topoSet -dict system/removeRedundantZones.topoSetDict
mv log.topoSet log.removeRedundantZones.topoSet

#runApplication setSet -batch createAMIFaces.setSet
#mv log.setSet log.createAMIFaces.setSet
runApplication topoSet -dict system/createAMIFaces.topoSetDict

mv log.topoSet log.createAMIFaces.topoSet

#
# - create the AMI faces by creating baffles, and then splitting the mesh

```

```
runApplication changeDictionary

# force removal of fields generated by snappy
\rm -rf 0
createBaffles -internalFacesOnly -overwrite rotif '(AMI1 AMI2)' \
> log.createBaffles 2>&1

runApplication mergeOrSplitBaffles -split -overwrite

# - apply the initial fields

cp -rf 0.org 0
```

XL

