POLITECNICO DI MILANO
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Aeronautica

Towards a high-fidelity open-source simulation framework for coupled fluid-structure interaction

Master thesis of:
Alice Zanella
Person code: 928277

Supervisor:
Full Prof. Alberto Guardone
Co-supervisors:
Dr. Myles Morelli
Dr. Neda Taymourtash

Anno Accademico 2020/2021

ὅταν βλέπω σε, προσκυνῶ, καὶ τους λόγους.
τῆς παρθένου τὸν οἶκον ἀστρῷον βλέπων
εἰς οὐρανὸν γάρ ἐστι σοῦ τὰ πράγματα,
Ὑπατία σεμνή, τῶν λόγων εὐμορφία,
ἄχραντον ἄστρον τῆς σοφῆς παιδεύσεως.

Quando ti vedo mi prostro davanti a te e alle tue parole,
vedendo la casa astrale della Vergine,
infatti verso il cielo è rivolto ogni tuo atto
Ipazia sacra, bellezza delle parole,
astro incontaminato della sapiente cultura.
(Pallada, Antologia Palatina, IX, 400)

Dedicata ai miei genitori
e a mio nonno

Ringraziamenti

La fine di questo percorso porta con sé emozioni differenti. Da un lato, profonda nostalgia nel ripensare a quanto passato. Dall'altro, immenso orgoglio dato dal raggiungimento del traguardo tanto desiderato, che mai sarebbe stato possible senza il sostegno di chi mi sta accanto.

Un doveroso e sentito ringraziamento va a chi ha reso possible questa tesi: il Prof. Alberto Guardone, la Dr.ssa Neda Taymourtash e il Dr. Myles Morelli. Mi sento onorata di aver potuto lavorare ad un progetto con così tante potenzialità, che senza dubbio mi ha messo alla prova e mi ha spinto verso traguardi che, fino a qualche mese fa, ritenevo impossibili. Senza il vostro completo sostegno e la vostra guida, non sarebbe stato possibile niente di tutto questo

Il grazie più grande va a mamma e papà, il mio punto di riferimento più grande. Mi avete dato tutto per poter intraprendere il percorso al Politecnico, credendo in me molto più di quanto non abbia mai fatto io.
Mamma, non riesco ad esprimere a parole quanto il tuo sostegno sia per me fondamentale. Sei la donna a cui aspiro a diventare e che, ogni giorno, mi insegna ad essere migliore. Grazie a te ho capito che è più importante vivere una vita piena di affetti, piuttosto che di successi. Che l'altruismo va coltivato e che l'amore va dimostrato sempre, tutti i giorni.
Grazie papà, per essere sempre stato dalla mia parte, anche negli sbagli. Da te ho imparato che l'amore non conosce distanze, né tempo. Il profondo legame che ci lega è e sarà sempre.

Un grazie importante al nonno Valter, il mio più grande fan. Mi riempie d'orgoglio sapere quanto sei fiero di me e quando guardo nei tuoi occhi mi sento sicura di aver intrapreso il percorso giusto

Grazie a Tobia, per essere diventato la persona più importante della mia vita. Grazie a te ho capito che amarsi incondizionatamente è possibile e bellissimo. Con te al mio fianco mi sento completa e so che, ovunque ci porti il lavoro, il legame che ci unisce non farà altro che aumentare.

Grazie a Lucrezia, Caterina, Rebecca, Federica e Laura, amiche da quando alle medie andavamo alle giostre di Mirandola. Possiamo non vederci per mesi, ma tutto rimane sempre uguale. Se ci vediamo tutti i giorni, la voglia di vedersi aumenta.
Un grazie particolare a Caterina, con cui, durante il secondo lock-down, ho condiviso non solo la scrivania, ma risate, pianti e bicchieri di vino.

Grazie a Matilde, Eleonora, Rachele ed Anna. Questi anni mi hanno insegnato che la nostra amicizia non conosce confini, che saremo sempre insieme con l'anima, anche se fisicamente in parti diverse del mondo.

Grazie a Gregorio, che più di un amico è un fratello. Insieme da sempre, da quando eravamo troppo piccoli per capire che sarebbe nato un legame così forte da far paura. Grazie per essere un punto di riferimento fondamentale. Su di te so di poter contare sempre, qualunque cosa accada (e ne sappiamo qualcosa).

Grazie a tutti i miei amici del Poli: Marco, Luca, Sere, Ste, Pozzo, Valeria, Dennis e Marco N.

Avete reso le lunghissime giornate in Bovisa più piacevoli, tra una risata e l'altra.

Grazie a mia cugina Chiara per aver condiviso gioie e dolori ogni giorno per 3 anni. Essere coinquiline oltre che cugine, è stato uno dei più bei regali della mia vita milanese.

Debbo ringraziare, infine, i miei professori del Liceo Pico.
In particolare la Professoressa Mantovani ed il Professor Pederzoli, poiché mi hanno portato ad essere la persona che sono ora, insegnandomi che il duro lavoro, la perseveranza e la passione portano sempre al raggiungimento di un traguardo. E' anche grazie a voi che, durante i periodi di sconforto, ho resistito e ho superato gli innumerevoli ostacoli.

Per citare Albert Einstein:

" C'è una forza motrice più forte del vapore, dell'elettricità e dell'energia atomica: la volontà."

Abstract

Fluid-structure interaction (FSI) represents one main concern for the aerospace industry, as it may apply to different scenarios, from wind turbines to rotorcrafts. FSI simulations are used as a way to predict the deformation of flexible structures subjected to forces exerted by the surrounding fluid. One possible way is to employ two separate solvers, each adept at solving a single physical phenomenon in the multi-physics problem.

The aim of this thesis is to create a open-source comprehensive code which couples SU2, the fluid solver, with MBDyn, the multi-body structural solver, using preCICE, the coupling library. One of the key aspect is the flexibility. The adapter is designed to be modular, general and "easy to use", making it possible to use it for different simulations, without directly changing the code.

Two cases are used as verification of the adapter code. A vertical flap which extends in a channel and deforms under the fluid force and a fixed square bluff body, with a flexible flap attached to it immersed in a laminar flow. Both of them compare well with the reference data, thus confirming the correctness of the present approach.

The comprehensive code has the ultimate goal of being used for rotorcraft FSI simulations, as in this context not many codes are freely available. Therefore, the HART-II rotor multi-body model is presented and a complete structural analysis is introduced, which will allow for the coupled simulation, as a future development.

Sommario

Nel campo dell'ingegneria aerospaziale, l'interazione fluido-struttura (FSI) suscita grande interesse, in quanto è rilevante in molti ambiti, dal design delle turbine eoliche a quello degli elicotteri. Le simulazioni FSI sono usate per prevedere la deformazione di strutture flessibili sottoposte alla forza del fluido che le circonda. Esistono diverse soluzioni per creare un accoppiamento fluido-struttura. Una di queste è l'impiego di due solver separati, ognuno dei quali è specializzato nella risoluzione del singolo problema. Vengono poi accoppiati, in modo da potersi scambiare informazioni, creando in tal modo il coupling.
L'obiettivo della presente tesi è di creare un codice open-source che accoppi SU2, il solver CFD, con MBDyn, solver multi-body strutturale, facendo uso di preCICE, una libreria specifica per l'accoppiamento di software nell'ambito delle simulazioni multi-physics. Uno degli aspetti chiave è la flessibilità. L'adapter è progettato per essere modulare, generale e facile da usare, rendendo possibile l'utilizzo per diverse simulazioni, senza modificare direttamente il codice.
Due casi vengono utilizzati come verifica dell'effettivo funzionamento dell'adapter. Un flap verticale che si estende in un canale e si deforma sotto la forza del fluido e un corpo tozzo cui è attaccato un flap flessibile, immerso in un flusso laminare. Entrambi trovano un buon riscontro con i dati di riferimento, confermando così la correttezza del presente approccio.
L'accoppiamento proposto ha l'obiettivo finale di essere utilizzato per simulazioni FSI di elicotteri, ambito in cui si distinguono molti simili accoppiamenti, ma quasi nessuno completamente open-source.
Viene, dunque, presentato il modello multi-body del rotore HART-II di cui viene fatta un'analisi strutturale completa, che consentirà di effettuare la simulazione accoppiata come sviluppo futuro.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

ALE	Arbitrary Lagrangian Eulerian
API	Application Programming Interface
BVI	Blade-Vortex Interaction
CFD	Computational Fluid Dynamics
CSD	Computational Structure Dynamics
FFT	Fast Fourier Transform
FSI	Fluid-Structure Interaction
$HART$	Higher-Harmonic Control Aeroacoustics Rotor Test
HHC	Higher Armonic Control
$MAVs$	Micro-Air Vehicles
MN	Minimum Noise
MV	Minimum Vibration
NSE	Navier Stokes Equations
$PDEs$	Partial Differential Equations
PIV	Particle Image Velocimetry
$RANS$	Reynolds Average Navier Stokes
RBF	Radial Basis Functions
$VTOL$	Vertical Landing and Take-Off aircraft
$WIPP$	Workshop for Integrated Propeller Prediction

Symbols

$\boldsymbol{\epsilon}$	strain tensor
$\boldsymbol{\gamma}$	angular momentum

$\boldsymbol{\omega}$	angular velocity
$\boldsymbol{\sigma}$	stress tensor
$\boldsymbol{\tau}$	viscous stress tensor
$\boldsymbol{f}$	force vector
$\boldsymbol{g}$	gravity constant
$\boldsymbol{I}$	identity matrix
$\boldsymbol{J}$	second order inertia moment
$\boldsymbol{L}$	strain-rate tensor
$\boldsymbol{M}$	mass matrix
$\boldsymbol{m}$	torque vector
$\boldsymbol{n}$	outward normal vector
$\boldsymbol{q}$	linear momentum
$\boldsymbol{S}$	first order inertia moment
$\boldsymbol{v}$	velocity vector
$\boldsymbol{x}$	position vector
λ, μ	Lamé constants
μ_f	fluid viscosity
ν	Poisson's ratio
ϕ	holomic constraint
ψ	non holomic constraint
ρ	density
C_p	specific heat
E	Young modulus
e_0	energy
F^c	convective fluxes
F^v	viscous fluxes
H	enthalpy
p	pressure
Q	generic source term

T	temperature
t	time
U	conservative variables vector

Chapter 1

Introduction

The interaction between fluid and structure has been a primary concern for the aerospace industry since the beginning and it poses a challenging problem both numerically and computationally. Early research concerning Fluid-Structure Interaction (FSI) primarily focused on the development of techniques to stabilise the coupled simulation and to achieve accurate results in a most efficient way.
FSI has a wide range of applications in various engineering fields. From the design of wind turbines in the energy department, to the investigation of blood flowing through the veins in the bio-engineering field. In a more aeronautical context, FSI finds natural application in a wide range of problems, like the design of largely-deformable wings, Vertical Take-Off and Landing (VTOL) aircrafts, flapping wings and, last but not least, rotorcrafts.
Computational FSI analysis has reached a significant level of development since the beginning in the late 1970s and early 1980s, thanks to the increasing capacity of computers and the advances in numerical techniques. As a result, the aeronautical industry relies more and more on FSI and several established CFD solvers have been already used to investigate the aeroelastic response of flexible wings and rotor blades.
Performing an accurate simulation of the flow field surrounding a helicopter is a complex task, requiring the consideration of aerodynamics and blade elasticity. There are several flight conditions for which accurate rotor loads prediction is not feasible without taking into account the aeroelastic effects, such as high speed forward flight or high trust condition. In these cases, there is a strong coupling between fluid and structure, hence the necessity of a multi-physics analysis.

1.1 Motivation

Many structural solvers come with an in-house low-fidelity aerodynamic model, which is able to reproduce the inflow and consequently trim the rotor. With the increasing available computational power and the possibility to parallelize the simulation in always bigger clusters, high-fidelity solvers are preferable, because of their capability to capture complex phenomena, such as the blade-vortex interaction (BVI) and the vortical wake.
The common way is to employ two separate solvers, one for the structural part and one for the fluid one, and have information exchanged between the two solvers. This data exchange is known as coupling with there being many different coupling approaches. A structural solver is used to model the blade dynamics and a fluid solver is used to model the aerodynamics of

the rotor. Subsequently, the aerodynamic predictions include the blade aeroelasticity and the predicted blade deflections include possible non-linear aerodynamics effects.
The goal of this thesis is to create a completely open-source environment which can solve fluid-structure interaction problems with a very generalised structure. The future aim is to apply this general environment to investigate the aeroelastic response of helicopter blades. In this way, the same case structure can be used for a variety of different simulations, stemming from simply hover cases to more complex flight conditions, such as high advancing ratio or unsteady manoeuvring.
Last but not least, one of the key aspects of this work is providing a flexible and user-friendly scheme. Since it is completely open-source, it can be tested, modified and updated to minimise computational errors and to improve the fidelity of predictive aeroelastic models.

1.2 Literature Review

This section provides an overview of the advances in the coupling between Computational Structural Dynamics (CSD) and Computational Fluid Dynamics (CFD) solvers, starting from the first aeronautical applications and moving on to rotorcraft simulations.
Early researches on fluid-structure interaction problems can be dated back to the mid 1970s and early 1980s, with the works of Belytschko [8]-[9] and of Bathe and Hahn [10]. Emphasis was placed on computational algorithms, especially finite element method, time integration and mesh deformation, with various examples, such as pipe flow. In 1982, Donea et al. [11] presented an Arbitrary Lagrangian-Eulerian (ALE) finite element method for the prediction of the non-linear response of fluid-structure systems exposed to transient dynamic loading.
The advances in the capacity of computers and the developement of new numerical tools, especially in the CFD area, have made it feasible to address many complex problems concerning FSI. The interaction between flexible wings and the surrounding fluid is a central topic for micro-air vehicles (MAVs): structural flexibility enhances the leading-edge suction via increasing the effective angle of attack, resulting in higher thrust generation [12].
Another area in which computational FSI plays an important role is renewable energy production. Wind turbines are largely dependant on the interaction between the rotating blades and the fluid that surrounds them. Since the blades are long, thin and flexible structures, aeroelasticity has a huge impact on lift production and it is capable of causing flutter, a destructive condition [13], [14].

The accurate modelling of FSI problems is an essential element to correctly predict the states of a complex system such as rotorcraft. Consequently, the application of FSI in rotorcraft studies became more dominant in the 1980's. In 1986 Tung et al. coupled full-potential aerodynamic methods with comprehensive codes in a loosely-coupled transonic simulation using CAMRAD and Full Potential Rotor (FPR) solver [15]. Then again, in 1989, Bridgeman et al. used a finite element multi-body solver coupled with simplified models based on lifting line theory and vortex wake models [16].
In the 1990's, CFD methods were coupled with advanced CSD methods. Bauchau et al. developed a tightly-coupled solver coupling CAMRAD/JA and OVERFLOW to simulate the four-bladed UH-60A Blackhawk helicopter rotor [17].
As another example, Euler/Navier-Stokes methodology for rotary wing flows was implemented to perform a coupled analysis with an elastic rotor blade beam structural model in hovering flight [18]. The dissertation showed the great impact of the vortex wake on the blades and in-

vestigated the differences in the aeroelastic predictions using tightly/loosely-coupled analyses. In the early 2000's, finite element multi-body solvers were used to model rotor blade structural dynamics and control during rotorcraft operations [19]. These CSD codes provided simplified aerodynamic models in order to reduce computational costs. Another example of this comes from [20], whose purpose was to perform an aeroelastic analysis of a flexible rotorcraft with a multi-body solver.

Full-scale rotor systems experience strong variations across several Mach regimes during a single revolution. The wake system, which usually is in the vicinity of the rotor, can influence the flow field and produce highly non-linear phenomena, as well as the dynamic stall dominated by viscous effects. The accuracy of aerodynamic loads obtained from the lower-fidelity simplified aerodynamic models is therefore questioned. Benefitting from the increasing computational power available, the use of CFD has dramatically changed the game. Instead of using a comprehensive code with lower fidelity, CSD codes have been coupled with CFD tools in different types of coupling.
Several approaches are based on simultaneously solving the fluid and structural equations, which are called monolithic methods. Although these methods show a very good convergence, they require rewriting the existing code, which is developed to solve either the structure or the fluid domain and potentially is not open-source, hence not accessible.
Hübner et al. [21] proposed a monolithic approach to solve fluid-structure interaction problems using space-time finite elements and successfully applied it to bi-dimensional test cases where strong interactions were observed. In case of strong interactions, simultaneous solution procedures may be preferable in order to ensure stability and to accelerate convergence of the coupled solution. The concept is simple: simultaneous procedure solve the coupled system in a single iteration loop with consistent time integration schemes for all physical fields, leading to time accurate solutions. See, for example, [22], where Alonso and Jameson presented a fully-implicit approach applied to a transonic aeroelastic simulation of a swept wing. The system (fluid flow and structural model) is fully coupled and a fully converged solution is achieved at every time step of the calculation.
Several CFD solvers were coupled with CSD codes in a loosely-coupled methodology, where the information between CFD and CSD is transferred on a per revolution, periodic basis. This kind of coupling doesn't work in case of strong interaction, but can be time-saving when the airloads are periodic, such as such as the loads generated by a hovering rotor.
NASA FUN3D was coupled with CAMRAD II to compute the rotor airloads on the UH-60A rotorcraft at several flight conditions and was compared to OVERFLOW, another CFD code which uses structural grids, as opposed to FUN3D which uses unstructured grids [23]. The same helicopter was used by Potsdam et. al to predict airloads using CAMRAD II and OVERFLOW-D with a loosely-coupling method [24]. The tested flight conditions are:

1. low speed ($\mu = 0.15$) with blade-vortex interaction,
2. high speed ($\mu = 0.37$) with advancing blade negative lift,
3. high thrust with dynamic stall ($\mu = 0.24$),
4. hover.

The comparison with experimental data showed good agreement and for all cases the loose coupling methodology is shown to be stable and convergent. One of the key aspect of loose-coupling is the possibility to trim the rotor, while in the tight-coupling it seems to be a difficult

task to perform. On the other hand, tight-coupling can achieve higher accuracy and it requires less rewriting on the existing codes. It is a very general procedure that can be applied to various flight conditions. Pahlke et al. [25] showed improved correlation on the 7A 7AD model rotors in high speed forward flight with a loose coupling of FLOWer and DLR CSD code, including viscous effect.
These studies also demonstrate how easy it can be, with a partitioned approach, to change the coupled programs without changing the structure of the coupling, something which could never be achieved with a monolithic approach.
Altmikus et al. provided a comprehensive direct comparison of loosely- and tightly-coupled rotor blade simulations [26], the result being that the first method naturally yields trimmed solutions and takes less time, while the second provides more accurate solutions especially at high advance-ratios, but a trimmed solution is a prerequisite. Because the exchange of data between the CFD and the CSD codes at each time step is required by the tight coupling, and because the time step must be sufficiently small to minimise the phase differences in the two sets of data, the tight coupling process is computationally costly. If the two time steps are different, as it typically happens, there has to be also a way of sub-cycling in order for the smaller time step to reach the other and exchange data.
Chunhua Sheng et al. [27] demonstrated how tight coupling can be efficient in simulating rotors undergoing a manoeuvre, associated with a user prescribed drive which automatically changes the collective and cyclic controls. Table 1.1 gives a review of the principal rotorcraft FSI simulations in chronological order.

The Higher harmonic control Aeroacoustics Rotor Test II (HART-II) is an extensive rotor experimental data set developed through an international collaboration between the German Aerospace Center (DLR), the German-Dutch Wind Tunnels (DNW), the French Office National D'Etudes et de Recherches Aerospatiales (ONERA), NASA Langley Research Center, and the U.S. Army Aeroflightdynamics Directorate (AFDD) research organisations. The program started in October 2001. After the wind tunnel test in 2001 and since the establishment of the HART-II International Workshop in 2005 numerous publications were based on the released data.
The test used an aeroelastically scaled model of the BO-105 main rotor that was tested at DNW. Measured data for the HART-II includes detailed acoustic, aerodynamic, rotor wake, blade deformation, and rotor airloads data on a pressure instrumented blade [7], [28].
The analyses range from low-order computational structural dynamics (CSD) methods [29], [30], [31], Navier Stokes/Free wake hybrid methods [32], and CFD-CSD couplings using both loose and tight methods [33], [34], [24]. A complete insight into HART-II simulations is given by Smith et al. [35], with an assessment of CFD/CSD state-of-the-art using the HART-II International Workshop data [36]. The paper establishes simulation and modelling guidelines, provides a summary of state-of-the-art BVI CFD/CSD predictions, and explores the use of higher harmonic control (HHC) to reduce or eliminate BVI. Blade–vortex interaction (BVI) noise is generated from unsteady pressure fluctuations on a blade due to interactions with previously generated tip vortices during descent or manoeuvring flight. A schematic representation is given in Figure 1.1.
What emerges is that the CFD code has the most impact on the accuracy of results and that CFD/CSD coupling is capturing better the BVI than comprehensive codes.

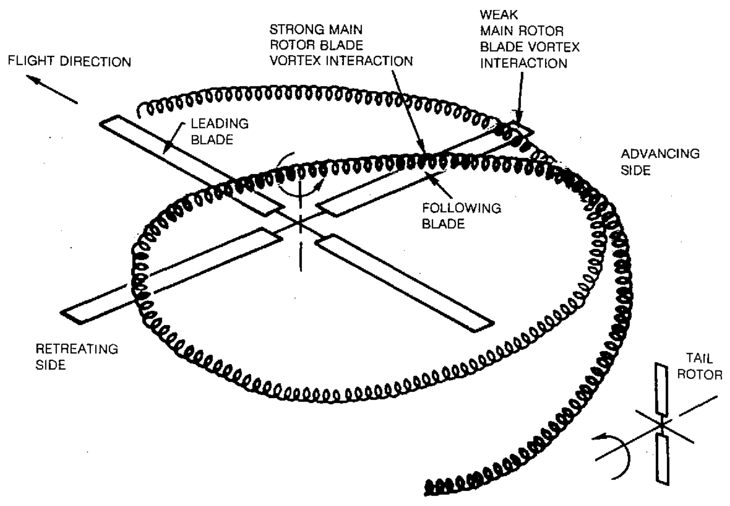


Figure 1.1: Schematic picture of the blade-vortex interaction. Taken from [1].

Table 1.1: Review of achievements in CFD/CSD coupling applied to rotorcrafts.

Author (Year)	Coupling Approach	Fluid Modelling	Structural Modelling	Strengths	Weaknesses
Pahlke et al. (2001)	loose	Euler (FLOWer)	FEM (S4)	trimmed solutions	phase shift, elastic torsion not accurate
Pomin et al. (2004)	tight	RANS (INROT)	FEM (DYNROT)	good correlations	time consuming
Potsdam et al. (2006)	loose	RANS (OVERFLOW -D)	FEM (CAMRAD II)	good correlations	not accurate for high speed and dynamic stall
Ananthan et al. (2008)	tight	RANS	CSD	flow phenomena captured correctly	bending moments not fully resolved
Sheng et al. (2013)	tight	RANS (U^2NCLE)	multi-body dynamics (DYMORE)	time consuming	efficient in manoeuvre flight

1.3 Structure of Work

The remaining work is organised into four distinct chapters. Chapter 2 introduces all the physical and computational aspects relevant to simulating FSI problems. It starts with a description of the fluid and structural domains and their respective equations and it continues with an assessment of the mesh deformation strategies and the different coupling approaches. In Chapter 3 the software tools used in this thesis are introduced, namely SU2, MBDyn and preCICE, with a thorough insight into preCICE library and its methods. Chapter 4 describes the implementation of two test cases, used to validate the coupling, with their respective results. Chapter 5 is all dedicated to the HART-II rotor. It starts with a description of the model and the reference data, then the structural model is presented. The results of the frequency analysis, hover and descent flight are then discussed, in anticipation of a CSD/CFD coupling. Chapter 6 suggests possible future improvements.
The structure of SU2 adapter is given in Appendix B, while the MBDyn adapter is presented in Appendix A. Appendix C gives an overview of the required steps to setup a coupled simulation.

Chapter 2

Physical and Computational Aspects

This chapter introduces the physical and computational principles upon which fluid-structure interaction problems are based. Firstly, a brief insight into the domains that compose FSI simulations is proposed in Section 2.1. The division between monolithic and partitioned approach is addressed in Section 2.2.1. Tightly coupled and loosely coupled methods are compared in Section 2.2.2. Then the mesh deformation techniques are described in Section 2.3, with special regard to *Radial Basis Functions* as a grid movement method applied to rotors.

2.1 Domains

In a FSI model, the fluid deforms the solid because of the forces exerted on the structure. In Section 2.1.1, the fluid domain and its equations of motion are presented. Then, the same goes for the solid domain in Section 2.1.2. The interface domain is outlined in Section 2.1.3. The specific discretization of the equations is presented in Section 3.1 and 3.2, where the two coupled solvers, namely SU2 and MBDyn, are introduced.

2.1.1 Fluid Domain

In this work, the Navier-Stokes (NS) equations are used to describe the fluid flow. A complete explanation of the NS equations can be found in references [37] and [38].
A brief summary of the NS equations will now be provided. The following mass (Equation 2.1), momentum (Equation 2.2) and energy (Equation 2.3) equations can be written as:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{v}) = 0 \tag{2.1}$$

$$\frac{\partial}{\partial t}(\rho \boldsymbol{v}) + \nabla \cdot (\rho \boldsymbol{v} \otimes \boldsymbol{v}) + \nabla p - \nabla \cdot \boldsymbol{\tau} - \rho \boldsymbol{g} = 0 \tag{2.2}$$

$$\frac{\partial}{\partial t}(\rho e_0) + \nabla \cdot (\rho e_0 \boldsymbol{v}) + \nabla \cdot (\boldsymbol{v} p + \boldsymbol{q} - \boldsymbol{v} \cdot \boldsymbol{\tau}) - \boldsymbol{v} \cdot \rho \boldsymbol{g} = 0. \tag{2.3}$$

Where ρ is the fluid density, $\boldsymbol{v}$ is the velocity vector, p is the pressure, $\boldsymbol{\tau}$ is the viscous stress tensor and $\boldsymbol{g}$ is the gravity constant. For a Newtonian fluid, the viscous stress tensor is given by:

$$\boldsymbol{\tau} = -\frac{2}{3}\mu_f (\nabla \cdot \boldsymbol{v}) I + 2\mu \boldsymbol{L} \tag{2.4}$$

with μ being the dynamic viscosity and the subscript f representing the fluid. $\boldsymbol{L}$ is the strain-rate tensor (i.e. the symmetric part of the velocity gradient $\nabla \boldsymbol{v}$):

$$\boldsymbol{L} = \frac{1}{2}\left(\nabla \boldsymbol{v} + \nabla \boldsymbol{v}^T\right) \tag{2.5}$$

In order to form a closed set of these equations, some other models have to be introduced, such as the conductive heat flow model, the caloric and the thermodynamic equations of state and the turbulence models.
SU2 solves various PDEs on a domain $\Omega \subset \mathbb{R}^3$ as long as they can be formulated as:

$$\frac{\partial \boldsymbol{U}}{\partial t} + \nabla \cdot \boldsymbol{F}^c(\boldsymbol{U}) - \nabla \cdot \boldsymbol{F}^v(\boldsymbol{U}) = Q \quad \text{in } \Omega_f \times [0, t] \tag{2.6}$$

where $\boldsymbol{U}$ is the vector of conservative variables, where $\boldsymbol{F}^c(\boldsymbol{U})$ are the convective fluxes, $\boldsymbol{F}^v(\boldsymbol{U})$ are the viscous fluxes and Q is a generic source term.
From this general form a wide variety of PDEs can be derived, including the classical Euler, Navier-Stokes and Reynolds Averaged Navier Stokes (RANS). For example, the NSE can be formed using as conservative variables:

$$\boldsymbol{U} = \begin{pmatrix} \rho \\ \rho \boldsymbol{v} \\ \rho e_0 \end{pmatrix}. \tag{2.7}$$

The convective and viscous fluxes are given as

$$\boldsymbol{F}^c(\boldsymbol{U}) = \begin{pmatrix} \rho \boldsymbol{v} \\ \rho \boldsymbol{v} \otimes \boldsymbol{v} + \boldsymbol{I} p \\ \rho \boldsymbol{v} H + p \boldsymbol{v} \end{pmatrix}, \qquad \boldsymbol{F}^v(\boldsymbol{U}) = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{\tau} \\ \boldsymbol{\tau} \cdot \boldsymbol{v} + \mu_f C_p \nabla T \end{pmatrix} \tag{2.8}$$

where p denotes the static pressure, $\boldsymbol{I} \in \mathbb{R}^{3\times3}$ is the identity matrix, H is the fluid enthalpy and $\boldsymbol{\tau}$ the viscous stress tensor. The heat conduction has been discretized with the Fourier law, with μ being the fluid viscosity and the subscript f representing the fluid, C_p the specific heat and T the temperature.
This formulation corresponds to an Eulerian description of the fluid domain, where the fluid properties are written as functions of space and time. The Eulerian perspective fixates on a particular point in space, and records the properties of the fluid elements passing through that point.

2.1.1.1 ALE RANS

In FSI problems, the deformation of the structure makes it necessary to account for the displacements of the mesh. The Eulerian point of view must be combined with the Lagrangian one (see for example [39]), where the particles of fluid are followed as they move through the flow. These algorithms are widely used in structural mechanics.
Figure 2.1 shows the three different perspectives: Lagrangian, Eulerian and ALE. The first picture refers to the Lagrangian method: the mesh nodes follow their respective particles through their movement in time and coincide with them. In the second picture the Eulerian mesh nodes remain at the same spatial points as time passes, while particles may change their positions. The last one shows the ALE perspective, where the nodes of the mesh can move independently

of the particles. Their only constraint is that the nodes displacements should not distort the mesh too much, in order to maintain accuracy during the computation. Mesh deformation techniques will be addressed in Section 2.3.

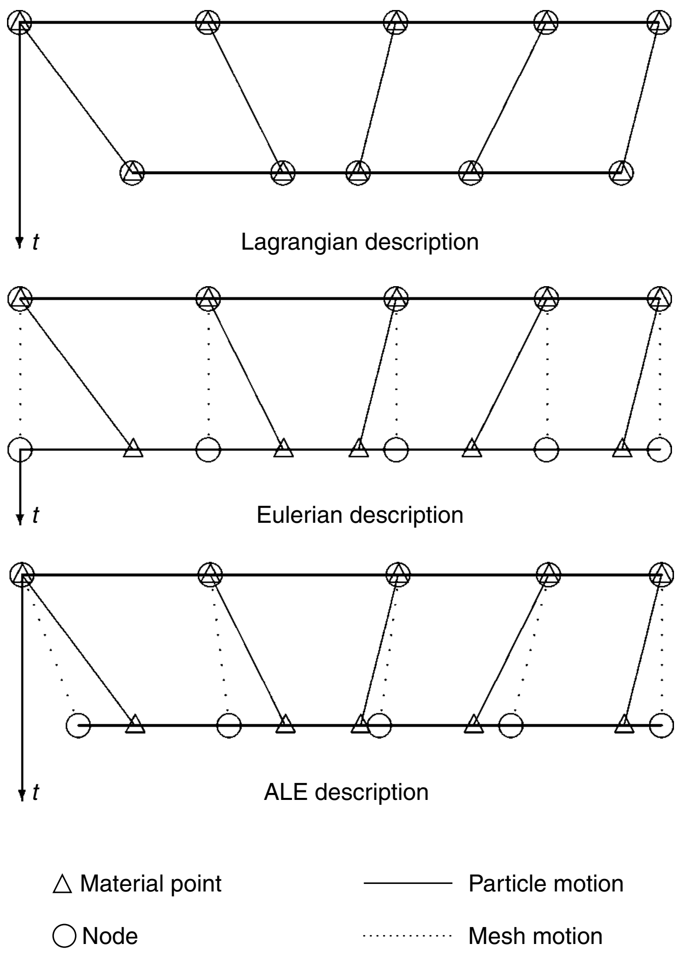
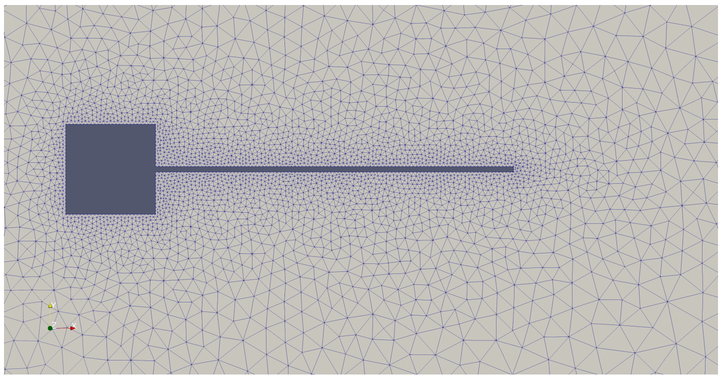


Figure 2.1: 1D example of Lagrangian, Eulerian and ALE mesh. Taken from [2].

Therefore, the Navier-Stokes equations have to be reformulated in an Arbitrary Lagrangian-Eulerian (ALE) framework [11], [40], [2], [41], an attempt to combine the advantages of the classical kinematic description, while minimising their respective drawbacks.
This method allows more freedom in moving the computational mesh, while preserving the FSI interface and this makes it suitable to deal with large deformations. The ALE observer moves arbitrarily with respect to the material particle. In this manner, the fluid mesh nodes at the interface do not detach from it and the rest of the mesh can move without displaying to much distortion.
In Figure 2.2 the ALE method can be appreciated: Figure 2.2a depicts the initial undeformed configuration, while Figure 2.2b shows the deformed mesh when the horizontal flap is at its maximum deflection. The case is discussed further in Section 4.2.
As before, the conservative variable is defined as $\boldsymbol{U} = (\rho, \rho\boldsymbol{v}, \rho e_0)$. The PDE system can be


(a) Undeformed mesh.

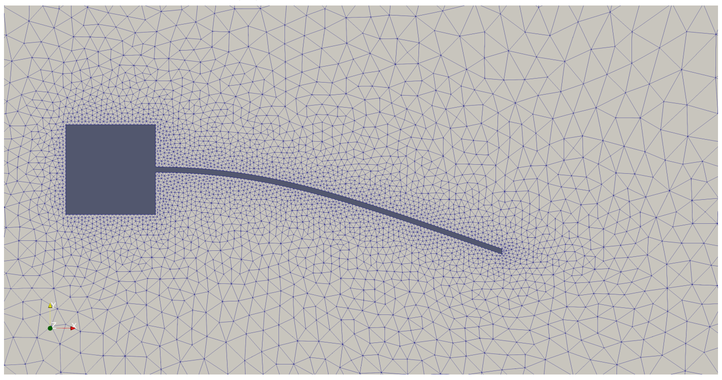

(b) Deformed mesh.

Figure 2.2: ALE approach to mesh movement.

rewritten as:

$$\frac{\partial \boldsymbol{U}}{\partial t} + \nabla \cdot \boldsymbol{F}^c_{ALE}(\boldsymbol{U}) - \nabla \cdot \boldsymbol{F}^v(\boldsymbol{U}) = Q \quad in\ \Omega_f \times [0,t] \tag{2.9}$$

where $\boldsymbol{F}^c_{ALE}(\boldsymbol{U})$ is the vector of convective fluxes and $\boldsymbol{F}^v(\boldsymbol{U})$ is the one for viscous fluxes, defined by

$$\boldsymbol{F}^c_{ALE}(\boldsymbol{U}) = \begin{pmatrix} \rho(\boldsymbol{v} - \dot{\boldsymbol{u}}_\Omega) \\ \rho\boldsymbol{v} \otimes (\boldsymbol{v} - \dot{\boldsymbol{u}}_\Omega) + \boldsymbol{I}p \\ \rho(\boldsymbol{v} - \dot{\boldsymbol{u}}_\Omega)e_0 + p\boldsymbol{v} \end{pmatrix}, \qquad \boldsymbol{F}^v(\boldsymbol{U}) = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{\tau} \\ \boldsymbol{\tau} \cdot \boldsymbol{v} + \mu_f C_p \nabla T \end{pmatrix} \tag{2.10}$$

with $\dot{\boldsymbol{u}}_\Omega$ being the velocity of the nodes of the grid.

2.1.2 Structural Domain

The solid domain is usually described with a Lagrangian perspective, since particles do not travel as much as the fluid ones. The de-Saint Venant-Kirchhoff model [42] is commonly used to describe the solid mechanics, especially when dealing with large deformations.
The material, both in the validation tests (Chapter 4) and in HART-II case (Chapter 5), is considered *homogeneous*, *linear elastic* and *isotropic.*
The design of complex mechanical system requires sophisticated models, with reliable approximations. With that in mind, a multi-body dynamics methodology seems to be the most performing and reliable way to model systems of rigid and deformable bodies, as a helicopter

or a wind turbine. It gives the right amount of complexity to every detail, thanks to its own modular approach and it saves a lot of computational power, compared to more traditional structural solvers.
The motion of a system of rigid/deformable bodies can be described using the *Newton-Euler equations*, which define linear momentum $\boldsymbol{q}$ and angular momentum γ for a set of rigid bodies

$$\boldsymbol{q} = \boldsymbol{M} \cdot \dot{\boldsymbol{x}} \tag{2.11}$$

$$\gamma = \boldsymbol{J}\boldsymbol{\omega} + S \times \dot{\boldsymbol{x}} \tag{2.12}$$

where $\boldsymbol{M}$ is the mass matrix, $\dot{\boldsymbol{x}}$ the velocity vector, S the first order inertia moment, $\boldsymbol{J}$ the second order inertia moment and $\boldsymbol{\omega}$ the angular velocity vector. Each structural node (Section 3.2.1) instantiates a set of $\boldsymbol{q}$ and γ equations. The equilibrium of force and moment is given by

$$\dot{\boldsymbol{q}} = \boldsymbol{M} \cdot \ddot{\boldsymbol{x}} = \boldsymbol{f} \tag{2.13}$$

$$\dot{\gamma} + \dot{\boldsymbol{x}} \times \boldsymbol{q} = \boldsymbol{m} \tag{2.14}$$

with acceleration vector $\ddot{\boldsymbol{x}}$, force vector $\boldsymbol{f}$ and torque vector $\boldsymbol{m}$ [43].
The system then has to be constrained, in order to reduce its degrees of freedom. In MBDyn this means adding joints (Section 3.2.2). Algebraic constraints in the form $\boldsymbol{\phi}(\boldsymbol{x}, \dot{\boldsymbol{x}}, t) = 0$ are added to the previous equations by means of the Lagrange multipliers.
The problem is formulated as a system of *algebraic differential equations* (DAE):

$$\boldsymbol{M} \cdot \dot{\boldsymbol{x}} - \boldsymbol{q} = 0 \tag{2.15}$$

$$\dot{\boldsymbol{q}} - \boldsymbol{\phi}^{\boldsymbol{T}}_{,\boldsymbol{x}} \boldsymbol{\lambda}_{\boldsymbol{\phi}} + \boldsymbol{\psi}^{\boldsymbol{T}}_{,\dot{\boldsymbol{x}}} - \boldsymbol{f}(\boldsymbol{x}, \dot{\boldsymbol{x}}, t) = 0 \tag{2.16}$$

$$\boldsymbol{\phi}(\boldsymbol{x}, t) = 0 \tag{2.17}$$

$$\boldsymbol{\psi}(\boldsymbol{x}, \dot{\boldsymbol{x}}, t) = 0 \tag{2.18}$$

where $\boldsymbol{\phi}^{\boldsymbol{T}}_{,\boldsymbol{x}}$ is the transpose of the partial derivative of the holonomic constraint $\boldsymbol{\phi}$ with respect to $\boldsymbol{x}$, $\boldsymbol{\psi}^{\boldsymbol{T}}_{,\dot{\boldsymbol{x}}}$ is the transpose of the partial derivative of the non-holonomic constraint with respect to $\dot{\boldsymbol{x}}$ and $\boldsymbol{f}$ contains both the forces and the moments.
An implicit multistep integration scheme is applied to solve the equation system.
Models with reduced dimensionality are used to discretize flexible and slender bodies. In particular, the *beam* model is considered when dealing with rotor blades or wind turbines. A beam is defined as a reference line and a reference orientation stemming from the reference line, which respectively represent the position in space of a reference point in the section and the orientation of the section itself (see Figure 2.3) [3].
The beam cross section movement depends on the reference line and all stresses are aggregated here (axial, shear, torsion, bending stiffness).

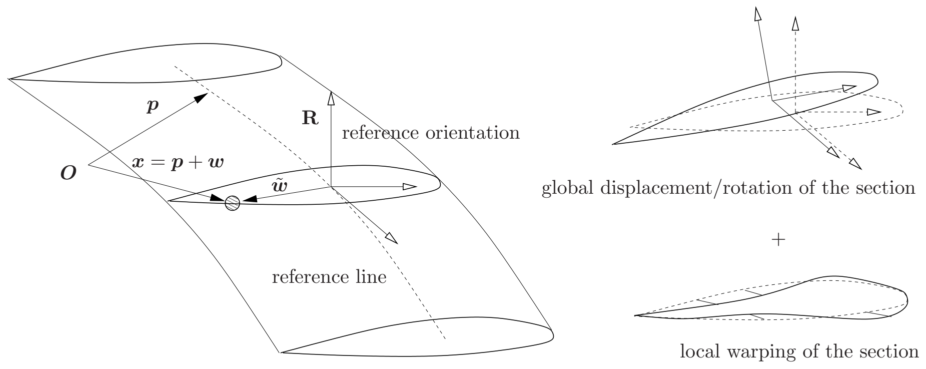

$\boldsymbol{p}$
$\mathbf{R}$
reference orientation
O
$\boldsymbol{x} = \boldsymbol{p} + \boldsymbol{w}$
$\tilde{\boldsymbol{w}}$
reference line
global displacement/rotation of the section
+
local warping of the section

Figure 2.3: Characterisation of the beam section. Taken from [3].

2.1.3 FSI Interface

When talking about fluid-structure interaction problems, the most important part is the common interface. Here, all the quantities are exchanged and some physical conditions must be met. Mass, momentum, and energy conservation must be respected and this is not automatically the case when distinct solvers are used to compute the fluid and the structure parts. These criteria provide the main basis for checking the quality of FSI calculations. A schematic picture of a sample FSI interface is shown in Figure 2.4.
For the sake of clarity, note that all the quantities related to the solid and the fluid domain, as well as the interface, are sub-scripted with S, F and SF, respectively.
First of all, according to the physics of FSI, the solid and fluid domains should never overlap nor separate. Then, for a viscous fluid, the flow velocity at the interface must be equal to the boundary velocity (*no-slip* condition). This means that the displacements of solid and fluid domain, as well as their velocities, must be equal at the interface (*kinematic requirement*):

$$\boldsymbol{x}_F = \boldsymbol{u}_S \; on \; \Gamma_{FS}, \tag{2.19}$$

$$\boldsymbol{v}_F = \frac{\partial \boldsymbol{u}_S}{\partial t} \; on \; \Gamma_{FS}. \tag{2.20}$$

Additionally, an equilibrium of the forces has to be imposed, such that the structure is not torn apart by the resultant forces. This leads to another condition:

$$\boldsymbol{\sigma}_F \cdot \boldsymbol{n}_F = \boldsymbol{\sigma}_S \cdot \boldsymbol{n}_S \; on \; \Gamma_{FS} \tag{2.21}$$

where $\boldsymbol{\sigma} \in \mathbb{R}^{3\times3}$ is the stress tensor and $\boldsymbol{n} \in \mathbb{R}^3$ is the outward normal vector of the fluid and solid domains.

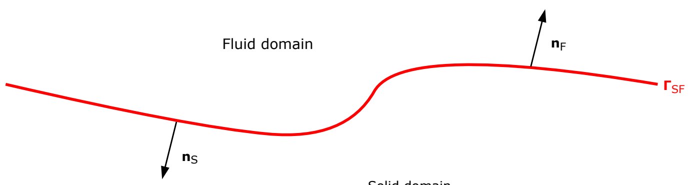


Figure 2.4: Example of a fluid-structure interface.

2.2 Coupling Approaches

In fluid-structure interaction problems, two physical worlds have to interact and communicate at the interface. The numerical methods used to solve FSI problems may be divided into two classes: the *monolithic approach* and the *partitioned approach*. While in the monolithic approach, the goal is to solve one global system of equations, in the partitioned approach separate sets of equations are set up for the fluid and the structure and the coupling is solved externally. Another distinction has to be made to denote the strength of the interactions between fluid and structure. If it happens at every timestep, it is called *tight-coupling*. If the exchange of information happens once every n iterations, it is called *loose-coupling*. In Figure 2.5 a very general example of coupling is given, without making any distinction of approach.

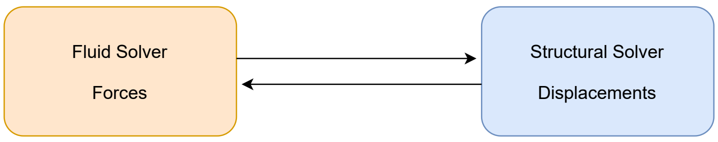


Figure 2.5: FSI coupling: exchange of information between fluid and structure.

2.2.1 Monolithic vs. Partitioned Approach

In the *monolithic approach* the interaction between fluid and structure at the mutual interface is treated simultaneously (see Figure 2.6) and the conservation of quantities at the interface is straightforward; see [44], [45], [46]. With such approach, only one solver would be responsible for both the flow fluid dynamics and the structural dynamics and, if well-implemented, this could result in a very efficient and robust method. The main drawback is that, usually, this type of approach is very case-dependent and specialised. Moreover, this kind of solvers are usually commercial and not open-source, so it could be very cumbersome to maintain.

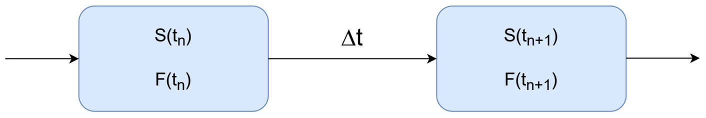


Figure 2.6: Monolithic approach. S and F are the solid and the fluid operators. The advancing of the solution from t to $t+1$ happens simultaneously.

On the other hand, in *partitioned approach*, the fluid and the solid domain are treated as separate and solved by their respective solver. They do collide at the interface, where the coupling between the two sets of equations takes place. In order to exchange data, at the interface some conditions must be enforced: the flow solution stalls until the solid one is updated, then the exchange takes place and the solution updates (Figure 2.7). The loop, as well as the coupling, requires an additional module (or software) to take care of the interaction and the data exchange.
One major drawback is the lack of unconditional stability: partitioned schemes are commonly energy-increasing, therefore the time-step must be reduced accordingly [47].
The great advantage, though, is the freedom of coupling different solvers, of managing the fluid and the solid domain discretization independently and by two highly specialized solvers. Existing solvers can be coupled, from commercial to academic to open-source and the modifications

to the codes are minimal.

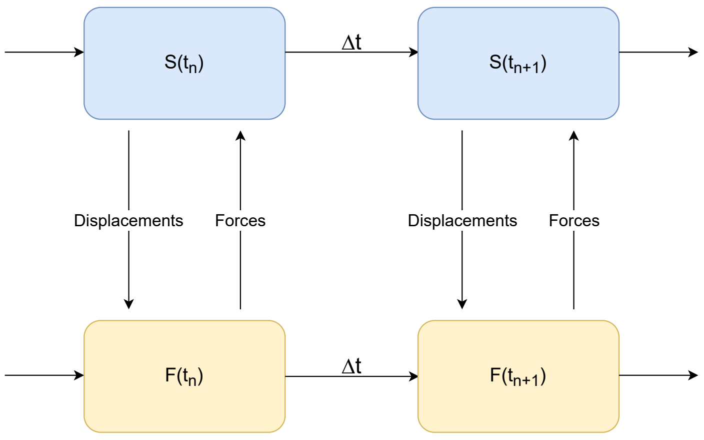


Figure 2.7: Partitioned approach. The fluid solver waits to receive the displacements from the structural solver, updates its solution and sends the forces to the structural solver. The time can advance and the loop repeats.

2.2.2 Tightly Coupled vs. Loosely Coupled

In rotor blade simulations, other than monolithic and partitioned approach, there is an additional choice to make: whether to do a *tightly-coupled* or a *loosely-coupled* simulation.
A loosely-coupled simulation only exchanges data periodically, typically once per full blade revolution [48]. It decreases the computational cost and can produce *trimmed* solutions. It is also used to predict the rotor loads [24] and it is very efficient when dealing with hover,free flight, steady ascent or descent. A first remarkable example of a loose approach was made by Tung and Cardonna in [15].
In tight coupling, instead, the data is exchanged at every time step [49], [50]. This means that non-periodic behaviour, which in loose coupling is impossible to capture, can be observed by the tight procedure. The exchange of displacements and forces is much more reliable and accurate but it is not advisable to use it to trim the rotor. For the purpose of this work, a tightly-coupled approach has been developed.

2.3 Mesh Deformation

In this Section, the mesh deformation techniques are addressed, as they are extremely important in FSI problems and in rotorcrafts simulations, where the nodes of the grid are moving through the domain and their position has to be adjourned.
The major problem that all mesh deformation techniques have to face is the robustness in preserving high mesh quality, particularly when dealing with large deformations.
One way is to treat the mesh as an elastic solid using the equations of linear elasticity [51]. The other is to use an interpolation based on radial basis functions.
In SU2, two kinds of mesh deformation are implemented: Linear Elasticity and Radial Basis Function (RBF) interpolation.

2.3.1 Linear Elasticity

The Linear Elasticity approach treats the mesh as an elastic solid and and models each edge of the mesh as a linear spring connected together at corresponding nodes. This helps prevent poor mesh quality however large displacements remain problematic [52]. Stein et al. [53] have applied it with an elastic stiffness varying in inverse proportion to the cell volume, aiming at preserving quality near the bodies, where there could be boundary layers or high resolution zones. In [52], a Galerkin finite element method is used to discretize the equations, in order to account for a much more robust solution to large deformations.
The equations of linear elasticity act on an elastic solid subjected body forces and surface tractions. Defining $\boldsymbol{u}(x) = (u, v, w)$ the small displacements vector, it can be written

$$\nabla \cdot \boldsymbol{\sigma} = \boldsymbol{f} \quad on\ \Omega, \tag{2.22}$$

where $\boldsymbol{f}$ is a body force, Ω is the computational domain and $\boldsymbol{\sigma}$ is the stress tensor given by the constitutive relation

$$\boldsymbol{\sigma} = \lambda Tr(\boldsymbol{\epsilon})\boldsymbol{I} + 2\mu\boldsymbol{\epsilon}, \tag{2.23}$$

where $\boldsymbol{\epsilon}$ is the strain tensor and Tr is its trace. λ and μ are the Lamé constants, properties of the elastic material, usually given as function of the Young's modulus E and the Poisson's ratio ν:

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}, \qquad \mu = \frac{E}{2(1+\nu)}. \tag{2.24}$$

E indicates the stiffness of the material, so a large E means rigidity. The Poisson's ratio measures how much the material shrinks in the lateral direction as it extends in the axial one ($-1 < \nu < \frac{1}{2}$ for physical materials).
The kinematic law, Equation 2.25, is used to quantify the deformation of a material fibre in an elastic body:

$$\boldsymbol{\epsilon} = \frac{1}{2}\left(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T\right) \tag{2.25}$$

Then, the system is closed by applying Dirichlet boundary conditions, $u = g$ on $\partial\Omega$.
In order to discretize this set of equations, a Galerkian method based on test and trial spaces is applied. Since a complete dissertation about this approach exceed the purpose of the present thesis, the reader is should refer to [52].
In order to achieve robustness and accuracy, E and ν are the most important parameters to set (see, for example, the work of Cavagna, Quaranta and Mantegazza [54]).
Different options are available in SU2 when setting E and ν. The first method is called *inverse volume* and sets $E = \frac{1}{V_i}$, where V_i is the element volume. This means that the lower is the volume, the highest is the stiffness. Another method is to set E as a constant, while varying the value of ν in order to obtain an acceptable stretching of the mesh elements. The third way is to set the mesh stiffness proportional to the distance from the wall [55].
It is important to consider that all these methods are subjected to the quality of the initial mesh, especially the progression from the smallest to the largest elements.

2.3.2 Radial Basis Functions

Radial Basis Functions [56] were efficiently used for dealing with rotor blade motions by Rendall and Allen [57] and they are introduced in SU2 by Morelli and Bellosta [58]. RBF are widely

used also in the context of FSI problems, see [59] and [60] just to name two. This technique is also used by preCICE while dealing with the data mapping between the two meshes (Section 3.3.3).
RBF mesh deformation methods are robust and preserve high-quality mesh even during large deformations. They do not require any information about mesh connectivity, so the implementation is very straightforward both for 2D and 3D domains. The main drawback is the computational cost for large meshes. This concern has been addressed with the introduction of multi-level greedy surface point selection algorithms [61] and volume point reduction methods [62] for large scale problems. These methods were implemented in SU2 by [58] in the context of ice formation on wings and the efficiency has been demonstrated on rotor blades [63] .
The radial basis function interpolation originated in the 1970's [64] and then the method has been firstly used to deal with scattered data, but it can be applied to a great variety of situations [65], [66]. In the context of mesh deformation, RBF generates high quality meshes with well preserved orthogonality and it can be used on all sorts of grids (structured and unstructured) because it doesn't require connectivity.
The main idea is based on a series of functions whose value is related to the distance between the selected position and a supporting point. A function $g : \mathbb{R}^d \rightarrow \mathbb{R}$ for which its function value only depends on the magnitude of its argument, is called radial. For example, $g(\boldsymbol{x}) = \phi(||\boldsymbol{x}||) = \phi(r)$, where $\phi : [0, \infty) \rightarrow \mathbb{R}$ and r is the length of $\boldsymbol{x}$. This means that ϕ is constant for input vectors of the same length and we call it *radial basis function.* Suppose we have a set of data $(\boldsymbol{x_i}, f_i)$ for $i = 1, 2, ..., n$. The goal is to find an interpolant $g(\boldsymbol{x}), \boldsymbol{x} \in \mathbb{R}^d$, that satisfies:

$$g(\boldsymbol{x_i}) = f_i, \quad i = 1, 2, ..., n. \tag{2.26}$$

Since we want the interpolant to be a radial basis function, $g(\boldsymbol{x})$ has to be a linear combination of translates of $\phi(\boldsymbol{x})$:

$$g(\boldsymbol{x_i}) = \sum_{i=1}^{n} \lambda_i \phi(||\boldsymbol{x} - \boldsymbol{x_i}||), \quad \boldsymbol{x} \in \mathbb{R}^d. \tag{2.27}$$

Putting this condition into Eq. 2.26 yields

$$\sum_{i=1}^{n} \lambda_i \phi(||\boldsymbol{x_j} - \boldsymbol{x_i}||) = f_j, \quad j = 1, 2, ..., n. \tag{2.28}$$

In a more compact way, it can be written as

$$\boldsymbol{\Phi\lambda} = \boldsymbol{f}, \tag{2.29}$$

where $\boldsymbol{\Phi}$ is a symmetric matrix. In order to have a unique solution, $\boldsymbol{\Phi}$ must be non-singular.
Many choices for radial basis functions exist and some of them are listed in Table 2.1. The parameter c, present in most of them, is used to adjust the shape of the function.
In SU2 the default option is the Wendland C2 (see Table 2.2):

$$\phi(\eta) = (1 - \eta)^4 (4\eta + 1) \tag{2.30}$$

where $\eta = \frac{||r - r_i||}{d}$, with d being the supporting radius.

Table 2.1: Examples of radial basis functions with global support.

RBF Name	$\phi(r)$	Parameters	Order
Gaussian	$e^{-(cr)^2}$	$c > 0$	0
Multiquadratics	$\sqrt{r^2 + c^2}$	$c > 0$	1
Inverse Multiquadratics	$\frac{1}{\sqrt{r^2+c^2}}$	$c > 0$	0
Inverse Quadratics	$\frac{1}{r^2+c^2}$	$c > 0$	0

Table 2.2: Compactly-supported radial basis functions [Wendland 1995]

RBF Name	$\phi(\eta)$
Wendland (C^0)	$(1-\eta)^2$
Wendland (C^2)	$(1-\eta)^4(4\eta+1)$
Wendland (C^4)	$(1-\eta)^6(\frac{35}{3}\eta^2+6\eta+1)$

Chapter 3

Numerical Tools

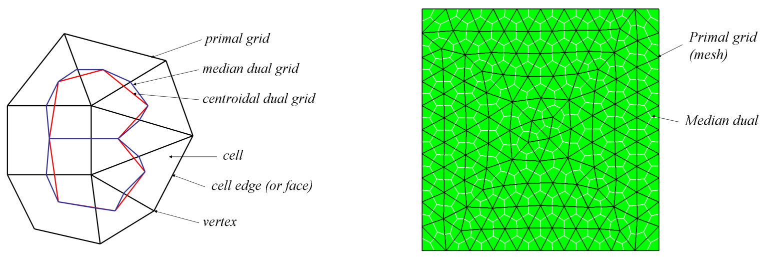
This Chapter introduces the individual solvers used for computing the aerodyanmics and aeroelasticity of the problem as well as the library used for coupling the two solvers.
First, in Section 3.1, SU2, the open-source CFD solver, is presented. Then, MBDyn is introduced in Section 3.2. Finally, preCICE library is illustrated in Section 3.3.

3.1 SU2

SU2, Stanford University Unstructured [1], is an open-source software suite initiated at the Aerospace Design Laboratory of Stanford University, freely available and licensed under the GNU Lesser General Public License, version 2.1 [67].
The choice to couple SU2 with preCICE comes from different observations. First of all, preCICE permits to couple any solver, ranging from commercial solid solvers to open-source codes like SU2. Secondly, preCICE offers a greater, more sophisticated procedure for the mapping of data between non-matching meshes. It also offers very elaborate acceleration schemes for implicit coupling, which outperform the constant and Aitken relaxation techniques implemented in SU2. Finally, the goal of this thesis is to create a completely open-source suite made of two specialised and sophisticated codes, coupled through an efficient library, in order to address the very time consuming task of solving fluid-structure interaction of a helicopter, during various flight conditions. This goal is exactly what preCICE can offer.
The SU2 suite deals with partial differential equations (PDE) using a finite volume method, on arbitrarily unstructured meshes. It uses a vertex-based approach, rather than cell-based, such that the variables are determined and stored at the vertices (nodes). This approach makes use of a median-dual grid, as shown in Figure 3.1.
The Navier-Stokes equations are discretized into RANS equations, which is the most used method in the aerospace field. Different turbulent models are implemented in SU2, such as the one-equation Spalart-Allmaras [68] and the two-equations k-ω SST [69], introduced in 1994. The convective fluxes can be discretized using central or upwind methods. Several methods are implemented in SU2, such as JST, ROE, AUSM, Lax-Friedrich, HLLC and Roe-Turkel. Second-order accuracy is achieved using a Monotone Upstream-centered Schemes of Conservation Laws (MUSCL) approach with gradient limitation [70].
In order to evaluate viscous fluxes using a finite volume method, flow quantities and their first

[1]SU2 website: `https://su2code.github.io/`

derivatives are required at the faces of the control volumes. The gradients of the flow variables are calculated using either a Green-Gauss or least-squares method at all grid nodes and then averaged to obtain the gradients at the cell faces. Recently, SU2 has been extended with a structural solver based on the finite element method (FEM) for the purpose of solving FSI problems [40], although it does not permit the same flexibility as using an external coupling library.
SU2 has also been extensively used for rotorcraft simulations [63], [58]. Many other codes before have been developed for rotorcrafts, showing with very good predictive capabilities. What SU2 can add is it being open-source code, with an active and growing community of developers. The toolkit is continuously updated so to identify new techniques, minimise errors and perform better Validation and Verfication (VnV).
It is also important to mention that SU2 has been used to simulate the turbulent flow field around a wing-tip mounted propeller configuration in the context of the Workshop for Integrated Propeller Prediction (WIPP). One of the key aspect of WIPP is the propeller-wing interaction, which has to be properly captured and resolved. An important feature within SU2 is the discrete adjoint framework, which allows for sensitivity analysis and optimal design of the full propeller-wing assembly [71].



(a) (b)

Figure 3.1: Examples of the (a) primal, median dual, and centroidal dual grids on a mixed mesh of quadrilateral and triangular elements, and (b) the primal grid (black) and median dual grid (white) on a square domain meshed with triangular elements. Taken from [4].

.

3.2 MBDyn

MBDyn [43] is free and open-source general purpose Multibody Dynamics analysis software developed at the *Dipartimento di Scienze e Tecnologie Aerospaziali* of Politecnico di Milano.
If not stated otherwise, all the information concerning MBDyn is taken from the input manual and the official documentation[2].
MBDyn offers a multibody dynamics approach to analyze complex, multidisciplinary dynamics problems. Multibody simulations are suited to study the dynamic behaviour of interconnected rigid and/or flexible bodies undergoing rotations and displacements (see, for example, Figure 3.2). The calculation time is comparably short, as opposed to more traditional structural solvers and it makes multibody analysis a very efficient simulation tool for rotorcraft systems. For a complete and detailed dissertation of multibody dynamics analysis, the interested reader should consult [72].

[2]MBDyn website: `https://www.mbdyn.org/`

MBDyn can simulate linear and non-linear dynamics of rigid and flexible bodies (lumped elements, beams or shells) subjected to kinematic constraints, external forces and control subsystems. Another point of strength of MBDyn is the multidisciplinarity and efficiency: it integrates aerodynamic, electric, thermal and hydraulic domains with the structural domain. It also offers inflow models, which are used to simulate fixed-wing and rotorcraft aerodynamics and enable reliable analysis of various flight conditions.
As explained in more detail later, MBDyn can be connected to other software tools to perform multi-physics simulations. This is the feature used in this thesis, to pass externally computed forces (from SU2) as an input to MBDyn in order to steer the simulation. This is done with an adapter, which is explained in detail in Appendix A.
In this section, some relevant features of MBDyn are outlined, with a focus on nodes, beam elements, externals forces, joints, reference systems and inflow models. The mathematical model upon which MBDyn relies has already been explained in Section 2.

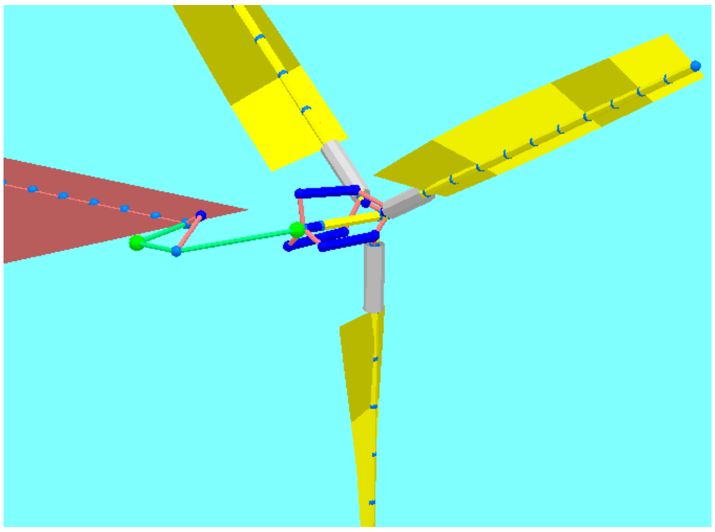


Figure 3.2: MBDyn tiltrotor model. Taken from [5].

3.2.1 Nodes

Nodes are the most fundamental MBDyn entities: they instantiate kinematic degrees of freedom and the corresponding equilibrium equations. Different types of nodes exist in MBDyn, we focus on structural nodes, that can be:

- `static` nodes, no inertia related to the node.
- `dynamic` nodes, with inertia. They can have up to 6 DoF (3 positions and 3 orientations) and they store all the information regarding position and inertia.
- `dummy` nodes, without inertia. Usually used to directly output the the motion of a point with an offset from a structural node.

3.2.2 Elements

Elements are used to connect nodes and they can access nodal properties and write contributions to the equations. Many types of elements exist in MBDyn, we focus on the ones used in the context of this work: beams, bodies, joints and forces.

3.2.2.1 Beams

Beams are particularly useful when dealing with slender bodies, as rotor blades [5] or wind turbines [14], in order to reduce the order of complexity to a 1D finite element model.
Deformable, slender beams are implemented in MBDyn by means of a Finite Volume approach [73]. They are defined by a reference line and its nodes. Currently, `beam3` (with 3 nodes) and `beam2` (with 2 nodes) are implemented in MBDyn, but only three-node beam elements are used in this context, see Figure 3.3.
Each of the three points the beam element connects is referred to a structural node but can have an arbitrary offset to allow more freedom in defining the reference line. The FV method presented in [73] is applied, therefore the forces and moments are evaluated at two points (point I and point II in Figure 3.3) which are midpoints between the three nodes: at $\xi = -1/\sqrt{3}$ and $\xi = 1/\sqrt{3}$ of a non-dimensional abscissa $-1 \leq \xi \leq 1$ running from node 1 to node 3.
At each evaluation point, a 6D constitutive law must be defined: it relates the strains and the curvatures of the beam to the internal forces and moments at the evaluation points. Various constitutive laws are implemented, from an isotropic beam section to a fully anisotropic one. In the context of this thesis, the `linear viscoelastic generic` law is used, both for the validation cases in Chapter 4 and for HART-II case in Chapter 5.
Other than `3beam` elements, aerodynamic beams are also implemented in MBDyn and will be used for the structural validation of the HART-II model, in order to account for the blade aerodynamics without SU2 input.

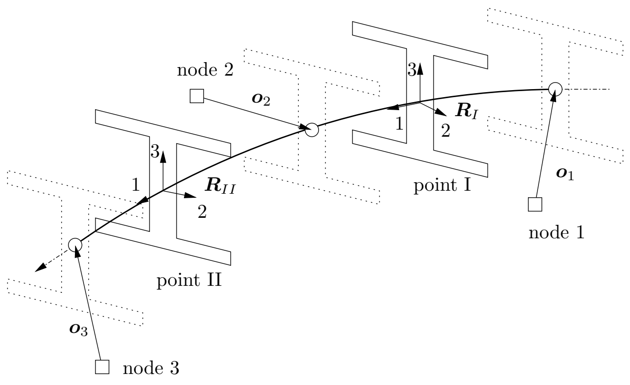


Figure 3.3: Geometry of the three-node beam element. Taken from MBDyn Input Manual.

3.2.2.2 Bodies

The `body` element describes a lumped rigid body when connected to a 6 DoF structural node or a point mass when connected to a rotationless, 3 DoF structural node. Within the `body` element, inertia properties must be specified (mass, center of mass and inertia).

3.2.2.3 Joints

Joints are used to constrain the kinematics and the degrees of freedom of structural dynamic nodes. MBDyn offers a great variety of different joints that can impose the lateral and rotational motion of single nodes or the relative position and orientation between two discrete nodes.
For the validation test cases, `clamp` and `total joint` are used, respectively, to ground all 6 degrees of freedom of a node and to allow to arbitrarily constrain specific components of the relative position and orientation of two nodes (see Section 4.1.2).
The joints used to build the HART-II structural model are the following:

1. `revolute hinge` to set the relative rotation of fixed and rotating parts of the swashplate
2. `axial rotation` to set the rotation of the hub
3. `total joint` to constrain the fixed swashplate to define collective and cyclic motions.

3.2.2.4 Forces

Once again, MBDyn offers a great range of force and torque elements. In the present work, the `external structural` element will be considered, which is the only one needed for FSI simulations.
An external structural force is used to prescribe forces that are calculated by an external software (in the present case SU2). The communication is set via socket, a system that establishes the bidirectional communication between two processes. An exemplary syntax is shown in Figure 3.4.

```
force: 2, external structural
    socket,
    create, yes,
    path, "\$MBSOCK",
    no signal,
    coupling, tight,
    orientation, orientation vector,
    accelerations, yes,
    4,
        1,
        2,
        3,
        4;
```

Figure 3.4: External structural element syntax.

3.2.3 Inflow Models

The `induced velocity` element is used by MBDyn to associate the aerodynamic elements when the inflow related computations are required. The goal of the present work is to use SU2, a high-fidelity CFD software, to compute the aerodynamic loads that act on the structure, so there is no need for a mid-fidelity aerodynamic model [74].
Different inflow models for rotors are implemented:

- `uniform`, where induced velocity is equal to its reference value everywhere

- `glauert`, used to approximate the inflow over the rotor disk in forward flight
- `mangler`, developed under the high speed assumption and to be used only for advance ratio grater than 0.1
- `dynamic inflow`, based on Pitt and Peters [75].

3.3 preCICE

preCICE is a coupling library developed by the Technical University of Munich (TUM), the University of Stuttgart and the University of Erlangen. Its name stands for *Precise Code Interaction Coupling Environment* and it is a free/open-source software.
The preCICE library offers the possibility to couple single-physics specialised software in a multi-physics simulation, without requiring knowledge about the inner processes of either solver. This approach is called partitioned as opposed to the monolithic approach, which would require a multi-physics solver working on the whole problem by itself.
The "black-box" approach seems more feasible and allows the user to do simulations in a plug-and-play manner with well known and reliable single-physics solvers.
Currently, preCICE can be used for partitioned multi-physics simulations including, but not restricted to fluid-structure interaction and conjugate heat transfer.
Flexibility is one of the main advantages of preCICE library: it provides technical communication, data mapping between non-matching grids, and coupling iteration numerics in an easy-to-use way that speeds up the development process. The user has to provide only the adapter for the involved solvers, which ensures that the data is formatted in the correct way and provides the time-step.
preCICE offers official adapters for well-known solvers such as OpenFOAM, deal.II, FEniCS, Nutils, CalculiX, or SU2, but every user can build their own adapter in a very easy and efficient way. The adapter can be inserted into the existing code with few calls to the preCICE library during the simulation loop. In Figure 3.5, all these aspects of preCICE are shown and will be addressed specifically in the next paragraphs. For the purposes of this work, preCICE library is chosen for several reasons.
Firstly, because of the possibility to create a general multi-physics environment as opposed to using *ad hoc* solutions for each type of problem or each type of solver. At any time, the adapter can be modified, upgraded or even replaced with a different one.
Secondly, because of the data mapping utility that preCICE offers, which will be thoroughly explained in Section 3.3.3. Within fluid-structure interaction problems, the fluid mesh and the structural one does not necessarily coincide: the fluid grid, in fact, is nearly always more refined. Hence, the need to map data between non-conforming meshes.
Lastly, preCICE can be useful when dealing with different time steps between coupled solvers, allowing one participant to sub-cycle while waiting for the other.

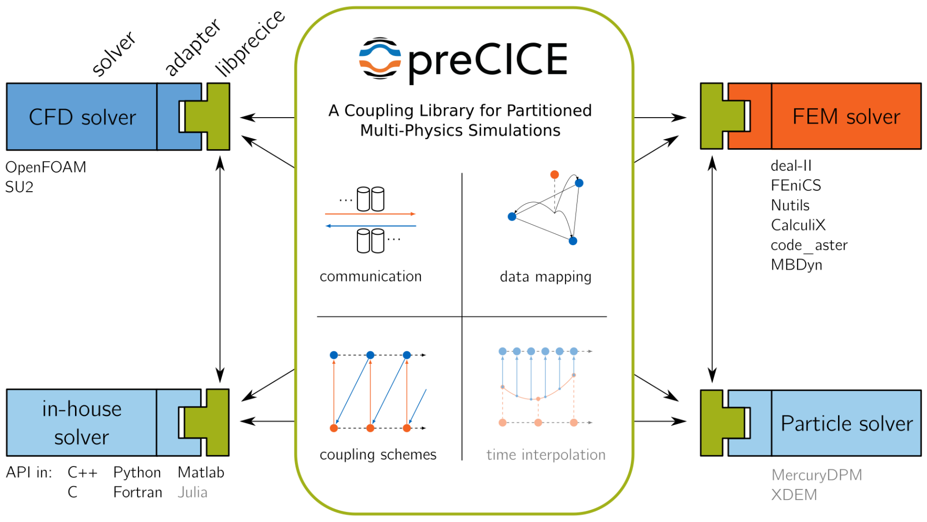


Figure 3.5: preCICE library overview. Taken from preCICE website.

3.3.1 Coupling Strategies

preCICE offers different type of coupling schemes, depending on the fluid-structure interaction at hand. There are four variables which control the interaction between solvers: the coupling step can be run either *parallel* or *serial* and in an *explicit* or *implicit* manner.
When the fluid and the solid solutions are computed iteratively until some convergence criteria is reached within the same time step, the scheme is called *implicit*. Whereas, if we are executing a fixed number of iterations (typically one per time step) without any convergence check, then the scheme is called *explicit*. The choice between them is crucial, not only because of the time/cost saving, but also because of the instabilities that could be present and not removed with an explicit coupling ([76], [77], [78]).
Serial coupling refers to the staggered execution of one participant after the other. *Parallel*, on the other hand, refers to the simultaneous execution of both participants and it is usually preferable for performance reasons.
Concerning notations, $\mathbf{S}$ and $\mathbf{F}$ represent the operators of the structural and fluid solvers, their variable being respectively vectors $\mathbf{s}$ and $\mathbf{f}$ at the FSI interface. n denotes the current time step of the computation.

3.3.1.1 Explicit Serial Coupling

The first algorithm introduced here is the *explicit serial coupling*, which is typically called conventional serial staggered. Here, the fluid solver uses the solid solution at the last time step to compute its current solution:

$$\mathbf{f}^{n+1} = \mathbf{F}^{n}(\mathbf{s}^{n}). \tag{3.1}$$

Then, $\mathbf{f}^{n+1}$ is transmitted to the structure solver and used as

$$\mathbf{s}^{n+1} = \mathbf{S}^{n}(\mathbf{f}^{n+1}). \tag{3.2}$$

This algorithm is represented graphically in Figure 3.6.

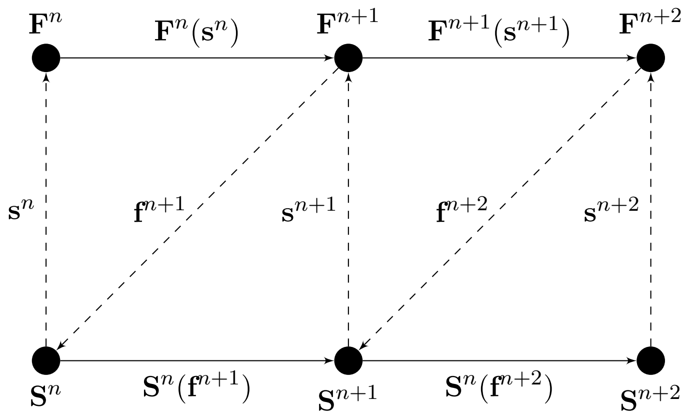


Figure 3.6: Conventional serial staggered (CSS) procedure.

3.3.1.2 Explicit Parallel Coupling

The CSS scheme can be improved by the parallel version, called *conventional parallel staggered* (CPS). The main difference is that now both fluid and structure solvers use coupling values from time step n.

$$\mathbf{f}^{n+1} = \mathbf{F}^n(\mathbf{s}^n) \tag{3.3}$$

$$\mathbf{s}^{n+1} = \mathbf{S}^n(\mathbf{f}^n) \tag{3.4}$$

The run time for the parallel scheme is usually smaller than that of the CSS one, but it could cause loss of accuracy and stability if used with a non-proper time step [79].
In Figure 3.7 the explicit parallel coupling procedure is shown. Note that now the solvers run simultaneously.

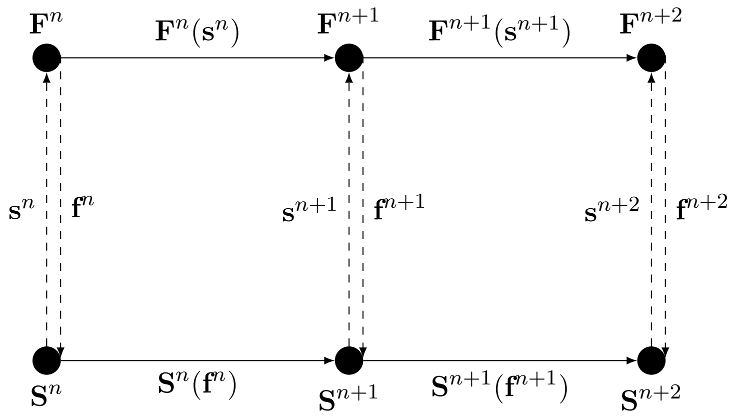


Figure 3.7: Conventional parallel staggered (CPS) procedure.

3.3.1.3 Implicit Coupling

We now discuss *implicit* schemes, which are preferable when instabilities forbid the use of explicit coupling schemes. The *implicit* coupling comes with the necessity of using particular techniques to stabilise the iterations, which will be briefly described in Section 3.3.2.

In the following, the letter k will be used to express the k-th sub-iteration of the coupling in the time step n, e.g. $\mathbf{s}_k^n$.
In the *implicit* scheme, the coupling conditions at the FSI interface are enforced in each time step up to a convergence criterion. If the criterion is not met, another sub-iteration within the same time instance is computed. For this reason, the solution can be approximated with an increased accuracy.
In preCICE, all *implicit* coupling methods are based on fixed-point iterations using the conventional staggered or the parallel scheme (as for the *explicit* case).

$$\mathbf{f}_{k+1}^{n+1} = \mathbf{F}^n(\mathbf{s}_k^{n+1}) \tag{3.5}$$

$$\mathbf{s}_{k+1}^{n+1} = \mathbf{S}^{n+1}(\mathbf{f}_k^{n+1}) \tag{3.6}$$

A general procedure for the *implicit serial* scheme is illustrated in Algorithm 1. Note that $\tilde{\mathbf{s}}$ indicates the solution obtained by the solver without any modification, while $\mathbf{s}$ is the post-processed solution where relaxation has been applied.
The *parallel* version is similar to the *explicit* one, as both solvers are run simultaneously and use the interface value at current time step $n + 1$ and at previous iteration k:

$$\mathbf{f}_{k+1}^{n+1} = \mathbf{F}^{n+1}(\mathbf{s}_k^{n+1}) \tag{3.7}$$

$$\mathbf{s}_{k+1}^{n+1} = \mathbf{S}^{n+1}(\mathbf{f}_k^{n+1}) \tag{3.8}$$

Algorithm 1 Implicit Serial Coupling Algorithm

```
1: $\mathbf{s}_0 = \mathbf{s}_p$
2: $k = 0$
3: while convergence criterion not met do
4:     $\mathbf{F}(\mathbf{s}_k) = \mathbf{f}_{k+1}$
5:     $\mathbf{S}(\mathbf{f}_{k+1}) = \tilde{\mathbf{s}}_{k+1}$
6:     compute $\mathbf{s}_{k+1}$ by relaxation
7:     $k = k + 1$
8: end while
```

As said before, *implicit* coupling in preCICE employs a fixed-point iteration of the form:

$$\tilde{\mathbf{s}}_{k+1} = \mathbf{S} \circ \mathbf{F}(\mathbf{s}_k). \tag{3.9}$$

Acceleration techniques are necessary to bring fixed-point equation to convergence. Those techniques are described in Section 3.3.2.

3.3.2 Acceleration Techniques

A complete coverage of the acceleration techniques is beyond the scope of this thesis, therefore only a brief insight into the methods offered by preCICE is given.
As explained before, *implicit coupling* requires some post-processing to make the solution of the FSI problem converge. In Algorithm 1, $\tilde{\mathbf{s}}_{k+1}$ is used to indicate the structural solution solely obtained by its own solver, without any modification, while $\mathbf{s}_{k+1}$ indicates that such modification (e.g. relaxation) has taken place.

We can then define a residual (Equation 3.10), which can be used to obtain a convergence criterion (Equation 3.11):

$$\mathbf{r}_{k+1} = \mathbf{S} \circ \mathbf{F}(\mathbf{s}_k) - \mathbf{s}_k = \tilde{\mathbf{s}}_{k+1} - \mathbf{s}_k, \tag{3.10}$$

$$\| \mathbf{r}_{k+1} \| < \epsilon_{abs}. \tag{3.11}$$

It is also useful to define a relative convergence criterion, to evaluate the difference between two subsequent sub-iterations:

$$\frac{\| \mathbf{r}_{k+1} \|}{\| \tilde{\mathbf{s}}_{k+1} \|} < \epsilon_{rel}. \tag{3.12}$$

The acceleration techniques which are introduced here are:

- constant under-relaxation
- dynamic Aitken under-relaxation
- quasi-Newton schemes

For a more thorough discussion see [80] and [78].
The simplest approach to stabilise the iterations and to enforce convergence is to use a suitable under-relaxation leading to a fixed-point iteration with under-relaxation:

$$\mathbf{s}_{k+1} = (1 - \omega)\mathbf{s}_k + \omega\tilde{\mathbf{s}}_{k+1}, \tag{3.13}$$

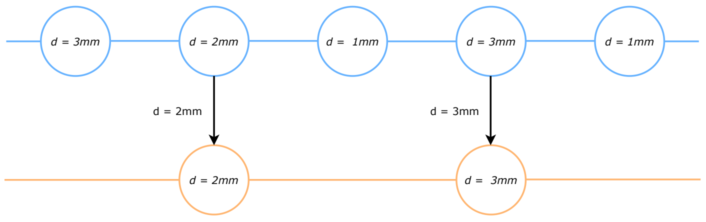
with $0 < \omega < 1$. If we use a value of ω close to 1, the convergence is faster, but the stabilising effect is lower, whereas for values close to 0 the stabilisation is strong, but the convergence is slow. Choosing ω is not a simple task, particularly because of the various nature of FSI problems.
The convergence can be speed up by using a dynamic under-relaxation factor, as done in the Aitken method, which basically adapts the factor at each iteration with the following relation:

$$\omega_k = -\omega_{k-1} \frac{(r_{k-1})^T (r_k - r_{k-1})}{\| r_k - r_{k-1} \|^2} \tag{3.14}$$

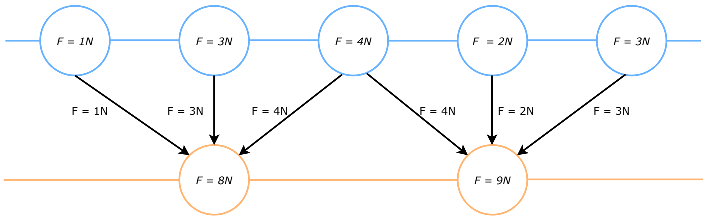
Under-relaxation is suitable for easy, stable problems, but issues arise when dealing with more involved case. These methods are then outperformed by quasi-Newton coupling schemes, which deal with the unavailability of derivative information (Jacobian) at the interface. This problem derives from the fact that preCICE treats *black box* solvers, hence no inner information is available.
preCICE offers different quasi-Newton methods, for serial and parallel usage. The first algorithm is the IQN-ILS, which stands for Interface-Quasi-Newton with Approximation of the Inverse of the Interface Matrix by Least-Squares. When the two solvers are executed in a sequential manner (first the fluid and then the structural), IQN-ILS modifies the structural solution such that the underlying fixed-point iteration converges, then this solution is fed back to the fluid solver and the loop restarts, as long as the convergence criterion is not met.
When parallel procedure is called, firstly the two solvers are executed simultaneously, then IQN-ILS modifies both vectors of kinematic variables. If the convergence criterion is not met, these modified values are used as input for the next iteration and the cycle is repeated.
There exist other algorithms, like generalised Broyden (IQN-IMVJ), however these methods are beyond the scope of this thesis. For a further description of these methods, please refer to [78], [77], [81] and [80].

3.3.3 Data Mapping

When coupling two participants at a common interface, in general, the two surface meshes do not match. In FSI simulations, typically, the fluid mesh is finer than the solid one, meaning that at the interface more fluid nodes than structural appear. Moreover, in our specific case, MBDyn provides a series of nodes (the beam reference line) while SU2 uses grids made of elements (tetrahedrons, triangles and so on).
The other, significant, challenge is that the mapper has also to not disrupt the mass and energy balances. Therefore, preCICE offers different methods to correctly map data between the solid and the fluid participants, either in a *consistent* or in a *conservative* form. An example is shown in Figure 3.8.
In the consistent mapping, the value of a node in one grid is the same as the value of the corresponding node in the other. This is the case of temperatures or displacements, as shown in Figure 3.8a: when displacements are mapped from a single solid node to the fluid nodes, it is not useful to distribute the single displacement value among the fluid nodes such that the displacements of the fluid nodes sum up to the displacement of the solid node. Rather, all fluid nodes assigned to that single solid node experience the same displacement.
The conservative form, on the other hand, makes sure the integral value are preserved. Forces need to be mapped in a conservative fashion, since the sum of forces on both sides of an interface needs to be the same (Figure 3.8b).



(a) Consistent mapping of displacements.



(b) Conservative mapping of forces.

Figure 3.8: Examples of mapping data between non-coincident meshes: consistent (a) and conservative (b) schemes.

Different mapping strategies are implemented in preCICE [81]. In the discussion the methods are presented in the consistent version, but the conservative one is also available:

- ***Nearest Neighbour***: it only requires vertex position information. The value at one node on the source mesh is assigned to the node of the target mesh that is closest to its position (Figure 3.9). Note that "closest" is intended in the sense of the shortest

Euclidean distance. This results in first order accuracy. It is the computationally easiest method and works well when the two meshes are almost coincident.

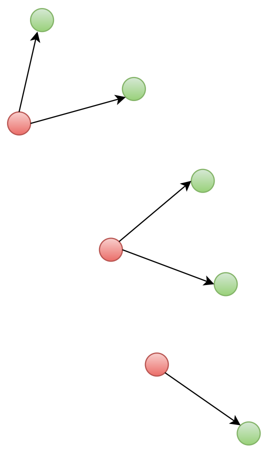


Figure 3.9: Nearest neighbour method: shortest Euclidean distance.

- ***Nearest Projection***: it uses three different sources of information and it requires the connectivity of the source mesh. The target mesh points are projected on the mesh elements of the source mesh, then the method performs linear interpolation on them and assigns the interpolated values back on the target mesh. Normally, this method is second order accurate, due to the small distance between two meshes, in relation to the elements size. A 3D representation is given in Figure 3.10.

- ***Radial Basis Functions***: it does not require any topological information and works well on most meshes. A variety of basis functions is implemented in preCICE through the PETSc library[3], but the most used ones are Gaussian and thin plate splines. The computational complexity of the data mapping can be reduced using a local support for basis functions. This means that the spatial influence of nodes, from which data is to be mapped, is limited to a certain range, called *support radius*. Generally speaking, the wider the support, the better is the approximation, however the complexity is increasing and the matrix becomes sparse.

[3]PETSc: `https://www.mcs.anl.gov/petsc/index.html`

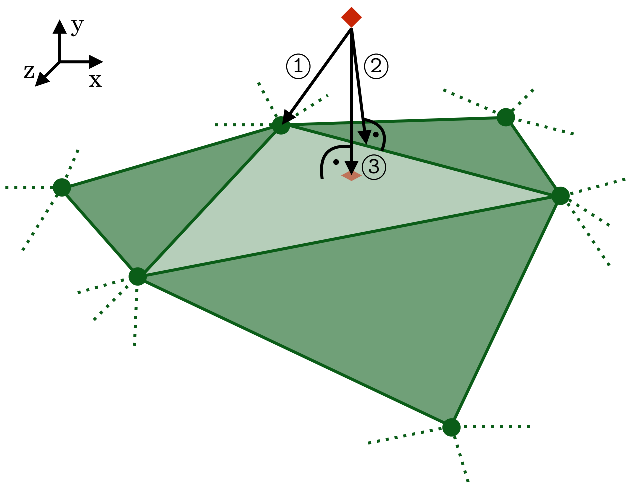


Figure 3.10: Nearest projection method for a 3D case: determining the shortest distance. The fluid mesh is the green one (unstructured). The structural node is the red diamond shape. 1) Distance as computed by nearest neighbor. 2) Orthogonal distance to the nearest edge of the fluid mesh. 3) Orthogonal distance to the nearest surface. Figure taken from [6].

3.3.4 preCICE API

The preCICE library is written in C++. However, different languages can be used when building an adapter, such as C, C++, Python, Fortran90/95 and Fortran2003. Particularly, the MBDyn adapter, which is thoroughly explained in Appendix A, is written in Python language. This choice is dictated by the MBDyn Python bindings for external communication, which makes it easier to work with the same language. The SU2 adapter which is explained in detail in Appendix B, instead, is written in C++ due to the core of the SU2 code being written in this language.

3.3.5 preCICE configuration file

The preCICE configuration file is essential to start up a coupled simulation. An example of it is shown below, in Listing 3.1. Through the configuration file, the user sets all the information used by the solvers:

- type and name of the solvers
- meshes used to exchange data
- how solvers communicate with each other
- method used for the mapping
- coupling scheme and timestep related information

```xml
<?xml version="1.0"?>

<precice-configuration>
<solver-interface dimensions="2">

  <data:vector name="Forces0" />
  <data:vector name="DisplacementDeltas0" />

  <mesh name="SU2_Mesh0">
      <use-data name="Forces0"/>
      <use-data name="DisplacementDeltas0"/>
  </mesh>

  <mesh name="MBDynNodes">
      <use-data name="DisplacementDeltas0" />
      <use-data name="Forces0" />
  </mesh>

  <participant name="SU2_CFD">
    <use-mesh   name="MBDynNodes" from="MBDyn" />
    <use-mesh   name="SU2_Mesh0"  provide="yes" />
    <write-data name="Forces0"             mesh="SU2_Mesh0" />
    <read-data  name="DisplacementDeltas0" mesh="SU2_Mesh0" />
    <mapping:rbf-thin-plate-splines direction="write" from="
   SU2_Mesh0"  to="MBDynNodes"
     constraint="conservative" />
    <mapping:rbf-thin-plate-splines direction="read"   from="
   MBDynNodes" to="SU2_Mesh0"
     constraint="consistent" />
  </participant>

  <participant name="MBDyn">
    <use-mesh   name="MBDynNodes" provide="yes"/>
    <write-data name="DisplacementDeltas0"   mesh="MBDynNodes" />
    <read-data  name="Forces0"               mesh="MBDynNodes" />
    <watch-point mesh="MBDynNodes" name="tip" coordinate="
   0.045;0.0" />
  </participant>

  <m2n:sockets exchange-directory="./../" from="MBDyn" to="
   SU2_CFD"/>

  <coupling-scheme:serial-implicit>
    <participants first="MBDyn" second="SU2_CFD" />
    <max-time value="10.0" />
    <time-window-size value="0.001" />
    <exchange data="Forces0"   mesh="MBDynNodes"         from="
```

```
SU2_CFD" to="MBDyn" />
    <exchange data="DisplacementDeltas0"    mesh="MBDynNodes"
  from="MBDyn"  to="SU2_CFD" />
    <max-iterations value="20"/>
    <relative-convergence-measure limit="1e-4" data="
  DisplacementDeltas0" mesh="MBDynNodes" />
    <relative-convergence-measure limit="1e-3" data="Forces0"
  mesh="MBDynNodes"/>
    <acceleration:aitken>
    <data name="Forces0" mesh="MBDynNodes"/>
    <initial-relaxation value="0.1"/>
    </acceleration:aitken>
  </coupling-scheme:serial-implicit>
</solver-interface>
</precice-configuration>
```

Listing 3.1: Example of preCICE configuration file.

3.4 Socket Communication

In order for the participants to connect and exchange data, a connection has to be set up. preCICE offers different methods: MPI, file communication and sockets.
The file communication mechanism is the most basic form of communication implemented in preCICE. It is based on writing and reading from files located in the computer hard disk. Its performance is comparatively poor and it is mainly due to the slow data transfer speed.
Message Passing Interface (MPI) and Transmission Control Protocol/Internet Protocol (TCP/IP), i.e. socket communication, are the most efficient techniques in terms of transfer speed.
The major drawback of MPI is that it requires a specific MPI version and there may be incompatibilities between solvers. Socket communication is quite as fast and it is free of incompatibilities.
On supercomputers, each node involved in the computation might have a different network address, not known a priori. This requires checking the available network address and specifying it in the preCICE configuration file, as it is shown in Listing 3.2. If one does not specify the network name, only one node can be used.

```
 <m2n:sockets exchange-directory="./../" from="MBDyn" network="
  bond0" to="SU2_CFD"/>
```

Listing 3.2: Socket communication in preCICE configuration file. The network name "bond0" refer to the cluster used at Politecnico di Milano.

Chapter 4

Verification Test Cases

Two bi-dimensional test cases have been performed to verify the reliability of the coupling on simple flows. The first one is a simple 2D vertical beam that interacts with a fluid and experiences large deformations (Section 4.1). The aim is to compare the coupling of MBDyn and SU2 with other known solvers, found on the preCICE repository[1].
The second case, presented in Section 4.2, is a well known FSI benchmark described in [82]: a square bluff body with a trailing flap is crossed by a viscous, laminar flow. The computational aspect is more involved, since fast oscillations of the flap are experienced, along with complex vortex structures. This test case is well suited to be an intermediate step towards a rotorcraft simulation.
Each section starts with a short description of the test case, including structural and fluid parameters, along with a detailed mesh description, then it proceeds presenting the obtained results.

4.1 Vertical Flap

This first test case is a 2D simulation of a deformable flap which extends in a channel. At the lower end the flap is clamped to the floor, while the upper end can move freely. Figure 4.1 shows the geometry of the test case. All the properties and the measures are taken from the preCICE tutorial repository, in order to do an accurate comparison between different software tools. The original simulation was itself not designed to produce physical results. Thus, the behaviour of the MBDyn-SU2 coupling is not intended to fully replicate the results from the original tutorial and the results from this simple coupling case are not used to validate the thesis work. Instead, this test case is just used in the initial development of the MBDyn adapter before implementing more complex models.
The results have been compared in terms of tip displacement with CalculiX[2], an open-source structural solver which uses finite element method, coupled with the same fluid solver, SU2. The aim is to validate the MBDyn adapter, which has been developed as part of this thesis.

[1]preCICE test cases: `https://github.com/precice/tutorials`
[2]CalculiX website: `http://www.calculix.de/`

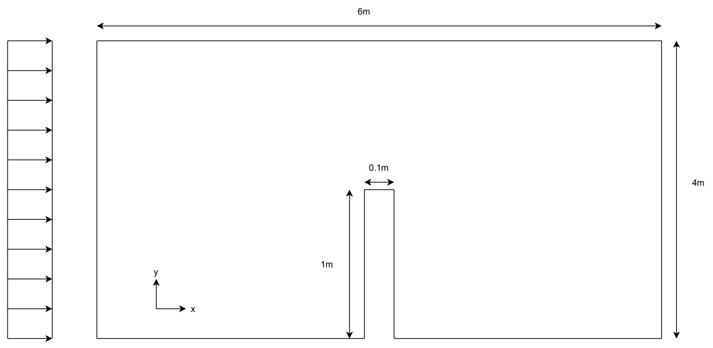


Figure 4.1: Domain of the 2D flap test case.

4.1.1 Fluid domain

The fluid domain is represented in Figure 4.1. The inlet is on the left with uniform flow velocity, the outlet is on the far right, while all other boundaries are *no-slip* walls.
The flow is compressible and inviscid, therefore governed by the Euler equations. The flow domain is discretized using an unstructured, triangular mesh generated by Gmsh [83], an open source 3D-2D mesh generator[3]. The undeformed mesh is shown in Figure 4.2 and its parameters are listed in Table 4.1.

Table 4.1: Vertical flap: mesh properties.

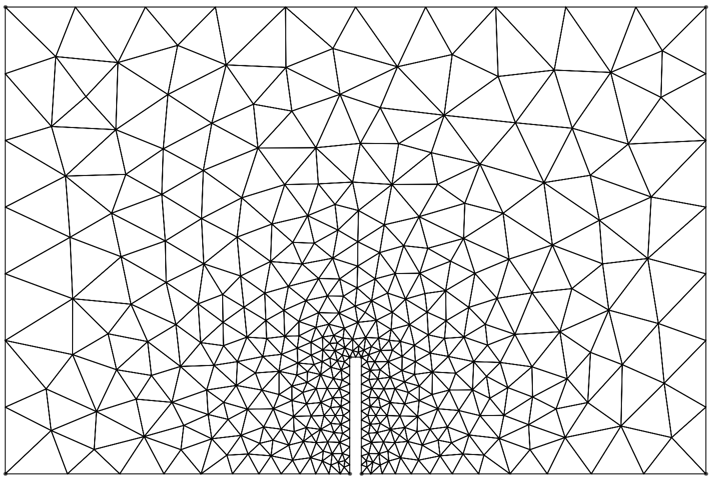
Parameter		Value
number of mesh points	n_p	394
number of mesh elements	n_{el}	710

The fluid properties are listed in Table 4.2. It should be stressed that this test case is no benchmark and serves just as a comparison between the coupling method proposed in this thesis and the other official adapters offered on preCICE website. For the same reason, no mesh convergence has been made and the same grid as for the CalculiX-SU2 coupling has been used.

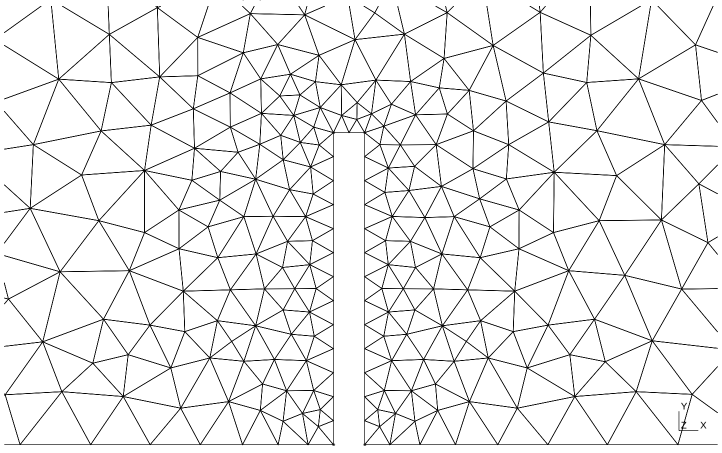
Table 4.2: Vertical flap: fluid properties.

Parameter			Value
fluid density	ρ	kg m^{-3}	1
kinematic viscosity	ν	m^2 s^{-1}	10^{-3}
Mach			0.1
flow type			Euler

[3]Gmsh website: `https://gmsh.info/`



(a) Vertical flap mesh



(b) Detail of vertical flap mesh

Figure 4.2: Unstructured triangular mesh generated by Gmsh.

4.1.2 Solid domain

The MBDyn model uses the solid properties of Table 4.3 and it is composed of 10 `beam3` elements. This requires 11 `node` elements to define the structure, as it is shown in Figure 4.3. The beam is clamped at one end and free at the other.

Table 4.3: Vertical flap: solid properties.

Parameter			Value
solid density	ρ	kg m^{-3}	3000
Elastic modulus	E	Pa	$4 \cdot 10^6$
Poisson coefficient	ν		0.3
damping factor			0.0001
width (z-dir)	h	m	0.14

The beam section is uniform and rectangular and the physical proprieties (ρ, E, ν) are constant throughout the beam length. The chosen constitutive law is `linear viscoelastic generic`, which adds a structural damping to the beam proportional to the stiffness matrix with a coefficient of $1 \cdot 10^{-4}$. The inertia of the structure is provided by two `body` elements per beam:

$$m = \rho w h \frac{l}{2} \tag{4.1}$$

$$I = \frac{m}{12} \begin{bmatrix} (l^2 + w^2) & 0 & 0 \\ 0 & (h^2 + w^2) & 0 \\ 0 & 0 & (l^2 + h^2) \end{bmatrix} \tag{4.2}$$

Since MBDyn treats 3D movements, all the nodes are constrained with `total joint` elements to move only in the xy plane.

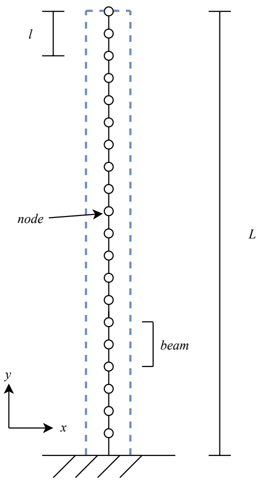


Figure 4.3: Vertical flap made of 10 beam elements.

4.1.3 Coupling

Table 4.4 summarises the coupling configurations. The FSI simulation is started from a single-physics fluid solution, in order to allow the flow field to fully develop and to speed up the convergence. This simulation is done with a fully rigid flap and the solely participation of SU2. Then, the real coupled simulation can start.
MBDyn provides displacements to SU2 which sends back forces. These forces are applied to the structure nodes through the `external structural` element (see Section 3.2) at every iteration. This last information defines the type of coupling as tight, meaning that kinematic variables are passed at every timestep, as opposed to once per *n-timesteps* (loose coupling), as it is normally done when a strong interaction between fluid and structure takes place.

Table 4.4: Vertical flap: coupling parameters.

Parameter			Value
total time		s	3
time step	Δt	s	10^{-2}
coupling scheme			serial implicit
acceleration algorithm			IQN-ILS
displacement rel. convergence limit			10^{-4}
forces rel. convergence limit			10^{-3}
data mapping			RBF: thin-plate-splines

As Table 4.4 reports, the simulation is run with an implicit coupling (see Section 3.3.1): MBDyn starts and feeds the displacements to SU2, which gives back forces. IQN-ILS has been chosen as the acceleration algorithm, with a QR2 filter (see 3.3.2) and a limit of $1 \cdot 10^{-2}$. For this simple test case a `RBF` mapping method (see Section 3.3.3) has been used for the interface mesh. The coupling scheme is reported in Listing 4.1, which shows part of the preCICE configuration file.

```xml
<!-- === Coupling scheme ================================ -->
<coupling-scheme:serial-implicit>
  <participants first="MBDyn" second="SU2_CFD" />
  <max-time value="5.0" />
  <time-window-size value="1e-2" />
  <exchange data="Forces0"   mesh="MBDynNodes"         from="
   SU2_CFD" to="MBDyn" />
  <exchange data="DisplacementDeltas0"    mesh="MBDynNodes"  from
   ="MBDyn"  to="SU2_CFD" />
  <max-iterations value="50"/>
  <relative-convergence-measure limit="1e-3" data="
   DisplacementDeltas0" mesh="MBDynNodes" />
  <relative-convergence-measure limit="1e-3" data="Forces0" mesh=
   "MBDynNodes"/>
  <extrapolation-order value="2"/>

 <acceleration:IQN-ILS>
    <data name="Forces0" mesh="MBDynNodes"/>
    <preconditioner type="residual-sum"/>
    <filter type="QR2" limit="1e-2"/>
    <initial-relaxation value="0.5"/>
    <max-used-iterations value="100"/>
    <time-windows-reused value="15"/>
 </acceleration:IQN-ILS>
</coupling-scheme:serial-implicit>
```

Listing 4.1: Coupling scheme as written in precice-config.xml

4.1.4 Results

The simulation is performed on 4 cores and takes approximately 2 minutes. In total, 1074 iterations are needed for the simulation, thus on average 2.15 iterations are necessary for the convergence of the FSI system per time step. The number of iterations required during the implicit coupling is shown in Figure 4.4.
The comparison with CalculiX has been made in terms of tip displacement in x-direction (flow-wise). As said before, the goal of this case is to test the passing of data in our coupling, not to reproduce exactly the SU2-CalculiX simulation, especially given that the case has non-physical meaning. The tip displacement is shown in Figure 4.5.

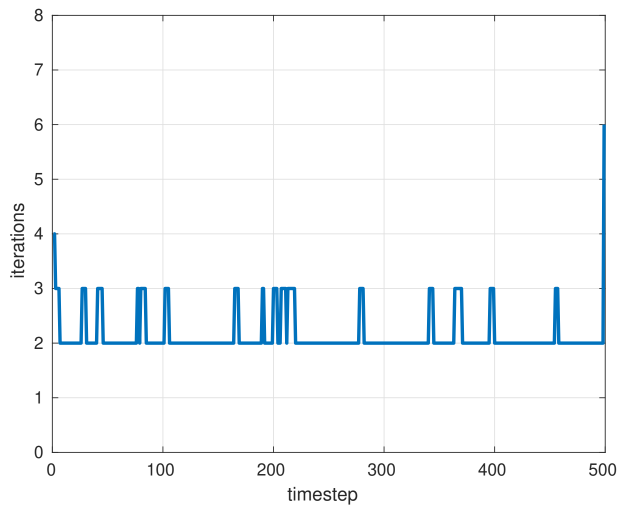


Figure 4.4: Vertical flap: number of iterations to achieve convergence of force and displacements at every timestep.

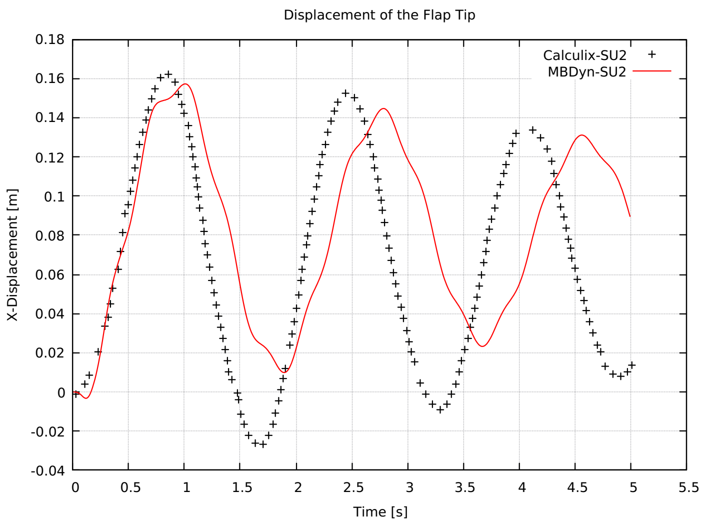
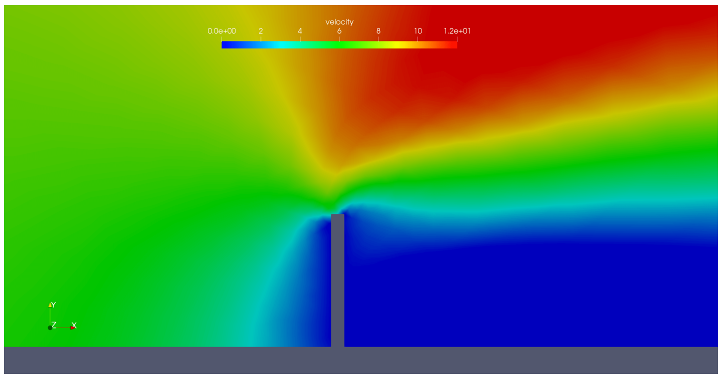
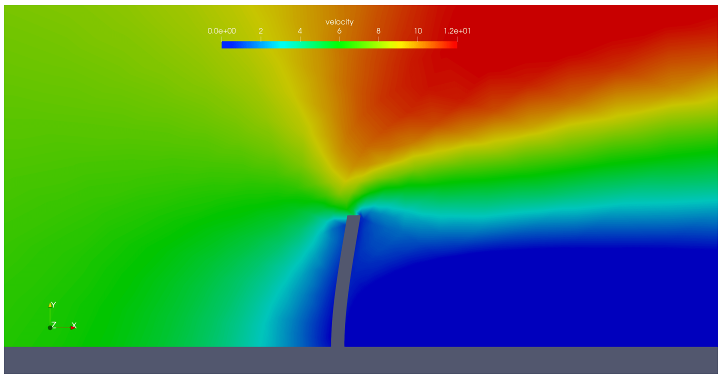


Figure 4.5: Vertical flap: comparison of the tip displacement in x-direction between MBDyn (red line) and CalculiX (black cross).

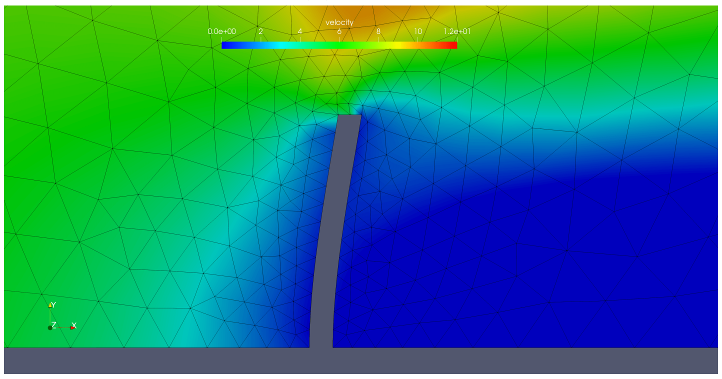
The flow field and the displacement of the flap tip is reported in Figure 4.6. Here it is observed the expected behaviour: the flap bends in the direction of the flow and then it oscillates back and forth 3 times. In Figure 4.6c, the mesh deformation can be appreciated.



(a) $t = 0$ s



(b) $t = 1$ s



(c) grid movement at $t = 1$ s

Figure 4.6: Vertical flap: initial configuration (a) and maximum tip displacement (b). The flow is represented as velocity magnitude. (c) shows the mesh deformation at maximum flap deflection.

4.2 Flexible Cantilever in Vortical Flow

The first test case aimed at verifying the correct exchanging of displacements and forces between the two solvers. The second test case, instead, is used to validate the performance of the system, mapping and coupling schemes included. The chosen example was first proposed in 1998 by Wall and Ramm [82], then studied by many [84], [44], [85], [86] and has also been used as a benchmark for fluid-structure interaction.
We consider a fixed square bluff body, with a flexible flap attached to it, immersed in a compressible flow (as opposed to almost all studies which consider an incompressible flow). The problem is computed in 2D, but presents various complexities. The square cylinder sheds vortices, which generate an area of low-pressure on its wake perceived by the flexible appendix. This alternate force generates vortex-induced vibrations on the structure which result in a structure driven vortex shedding. The domain is shown in Figure 4.7. Two parameters are extensively used in literature as a validation criterion: the frequency of the flap oscillation and the maximum amplitude of the vertical displacement at the cantilever tip. The reported frequencies range from 3.0 to 3.2 Hz, while the tip displacements range from 0.95 to 1.31 cm (See Section 4.2.4).

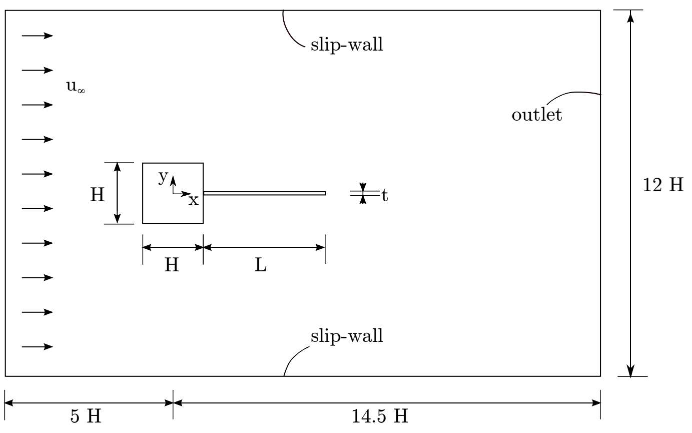


Figure 4.7: Bluff body with flexible cantilever: fluid domain.

4.2.1 Fluid Domain

As previously said, the domain is computed in 2D and consists of a square cylinder immersed in a low-Reynolds number compressible flow. The square and the flap are non-slip walls, while the upper and the lower boundaries are slip walls.
The geometrical and physical properties of the case are shown in Figure 4.7 and in Table 4.5.
Two unstructured meshes have been used for the validation of the fluid sub-problem, called respectively *coarse* and *fine*. They are generated by Gmsh [83] and their properties are listed in Table 4.6. Both of them have been run using two different time discretizations, with $\Delta t = 0.0005$ s and $\Delta t = 0.001$ s, with this last being the chosen one, considering a trade-off between time-saving and accuracy. The meshes are shown in Figure 4.8.

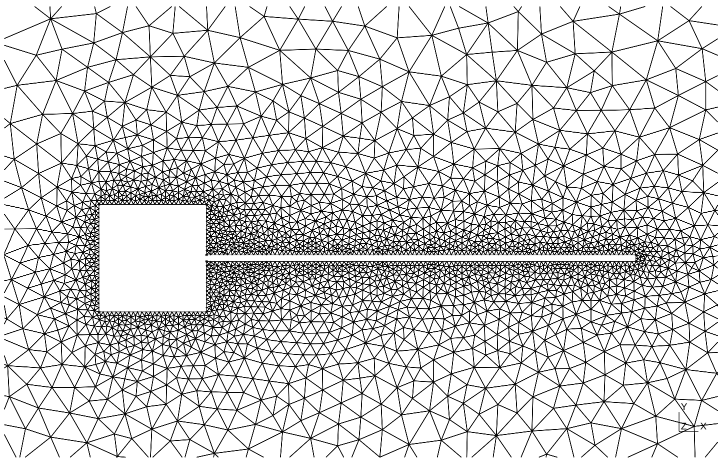
Table 4.5: Bluff body with flexible cantilever: geometric and fluid properties.

Parameter			Value
inlet velocity	u_∞	m/s	0.513
fluid density	ρ_f	kg/m^3	1.18
kinematic viscosity	μ_f	kg/m·s	$1.82 \cdot 10^{-5}$
Re			332
Ma			0.2
temperature	T	K	0.0164
flow type			NS laminar
H		cm	1
L		cm	4
t		cm	0.06

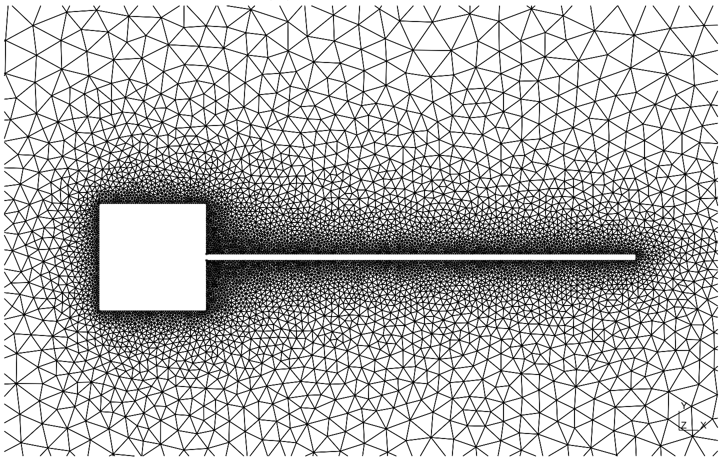
Table 4.6: Bluff body with flexible cantilever: mesh properties.

	Parameter		Value
Coarse mesh	number of mesh points	n_p	7622
	number of mesh elements	n_{el}	14563
Fine mesh	number of mesh points	n_p	3246
	number of mesh elements	n_{el}	6127

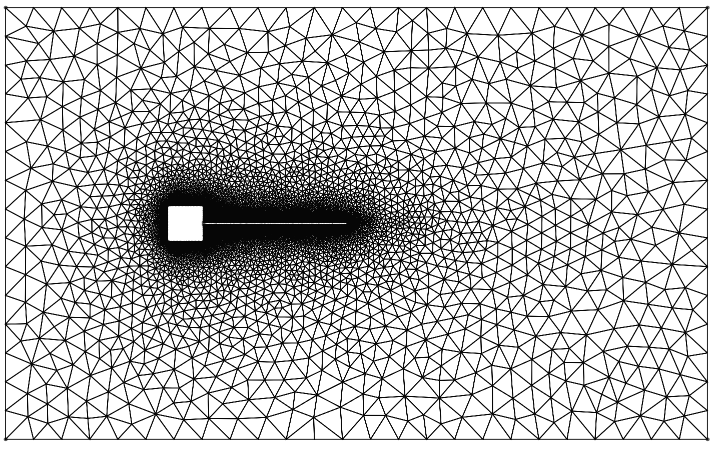
As for the previous test case, the coupled simulation is started from a single-physics one where the flap is maintained rigid. This permits the rapid development of the vortex shedding and improves the efficiency of the coupling.
The key configuration parameters used in the SU2 configuration file are briefly discussed here. A backward Euler with dual time stepping is used as time discretization, with an average number of internal iterations of 10-15. Dual time stepping is particularly appropriate for unsteady problems and can increase accuracy, provided that a convergence is reached. A multi-grid technique has also been used, in order to reach convergence in fewer iterations.
As a convective numerical method, Roe (upwind) is chosen instead of JST (centred) because of the nature of the problem, that is low-Re and low-Mach. To achieve second order Roe scheme has been used with MUSCL reconstruction.



(a) *Coarse mesh.*



(b) *Fine mesh.*



(c) View of the entire domain (*fine mesh*).

Figure 4.8: Bluff body with flexible cantilever: convergence study. (a) *Coarse mesh.* (b) *Fine mesh.* (c) Domain view of the *fine mesh.*

4.2.2 Solid Domain

The MBDyn model uses the solid properties of Table 4.7 and it is composed of 10 `beam3` elements, as the vertical flap case in Section 4.1. The beam is clamped at the left end and free at the other. The beam section is uniform and rectangular and the physical properties (ρ, E, ν) are constant throughout the beam length. The chosen constitutive law is `linear viscoelastic generic`, with a damping factor of 0.005, which appeared to be high enough to give a smooth solution without being excessively damping. The thickness of the cantilever beam as well as the material properties are chosen so that its first eigen-frequency is close to the frequency of the vortex shedding.
A representation of the flexible appendix MBDyn model is shown in Figure 4.9.

Table 4.7: Bluff body with flexible cantilever: solid properties.

Parameter			Value
solid density	ρ	kg m^{-3}	100
Elastic modulus	E	Pa	$4 \cdot 2.5 \cdot 10^5$
Poisson coefficient	ν		0.35
damping factor			0.005
width (z-dir)	h	m	0.1

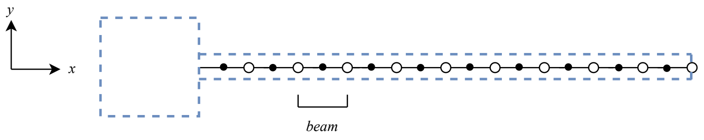


Figure 4.9: Representation of the flexible cantilever structure. Evidence on beams and nodes.

4.2.3 Coupling

Since the problem is strongly coupled, only implicit algorithms have been used, with various acceleration methods in order to evaluate their overall performance.
As already stated, the simulation is started from a previous state, computed with a rigid flap, in order to speed up the transient. The coupling parameters are listed in Table 4.8.

Table 4.8: Bluff body with flexible cantilever: coupling parameters.

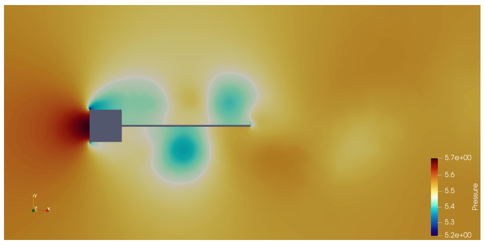
Parameter			Value
total time		s	10
time step	Δt	s	10^{-3}
coupling scheme			serial implicit
acceleration algorithm			IQN-ILS
displacement rel. convergence limit			10^{-4}
forces rel. convergence limit			10^{-3}
data mapping			RBF: thin-plate-splines

4.2.4 Results

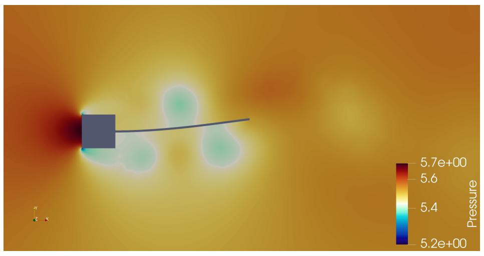
The results of the bluff body with flexible cantilever are hereby presented. Firstly, the complex phenomena involved are showed and explained, then the comparison between the present case and the other studies is assessed and finally, a performance analysis is proposed.

4.2.4.1 Flow Domain

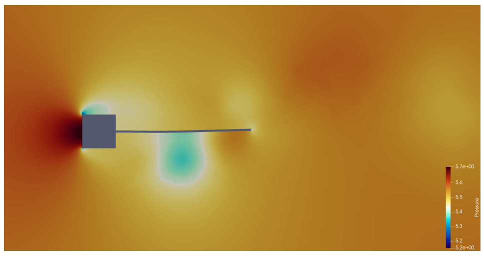
The first simulation to be run is the one with a rigid structure. After a few seconds, the flow exhibits a periodic behaviour with vortices separating from the corner of the square, as in Figure 4.10a. This is coherent with the low Reynolds number and the expected Strouhal number (~ 0.117) which corresponds to a vortex shedding frequency of ~ 3.7 Hz.
Then the actual coupled simulation can start and a transition from flow-driven vibrations to beam-driven vibrations is observed (Figure 4.10b). The aforementioned vortices induce alternative drop and increase in the pressure field behind the square. The vortex shedding causes oscillations of the flexible flap. A vortex lock-in regime is reached, with periodic oscillations of the flap (Figure 4.10c, 4.10d, 4.10e, 4.10f). The rapid motion of the cantilever starts to feed the vortex-pattern and these low and high pressure cores are increased by it. The flow field is represented in terms of pressure field.



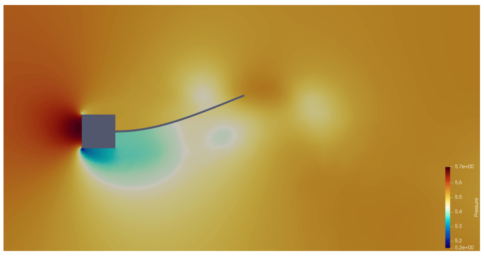
(a) Startup simulation: rigid appendix. Flow-driven vibrations.



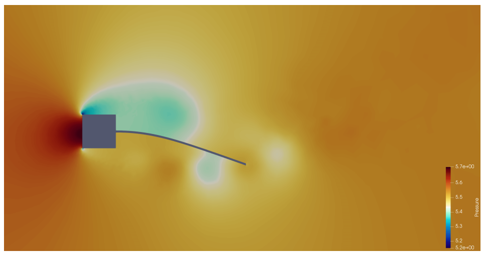
(b) Initial transient of the coupled simulation. Transition.



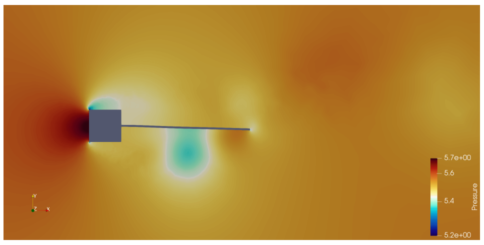
(c) $T = 0$. Beam driven vibrations.



(d) $T = \frac{\pi}{2}$.



(e) $T = \frac{3}{2}\pi$.



(f) $T = 2\pi$.

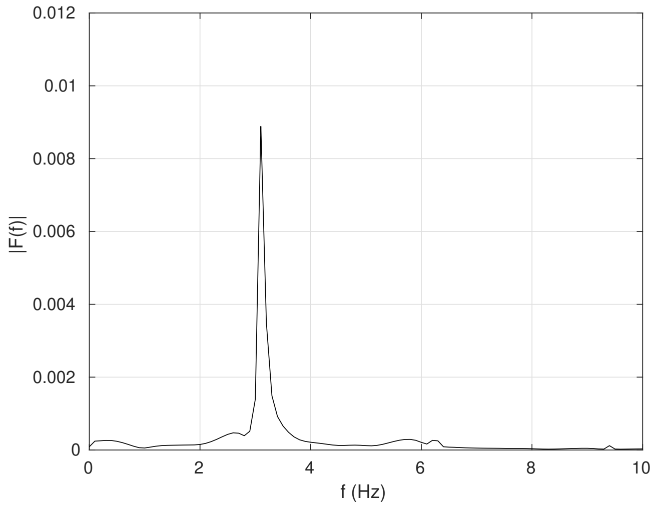
Figure 4.10: Bluff body with flexible cantilever: pressure visualization of the flow field in different phases.

4.2.4.2 Validation

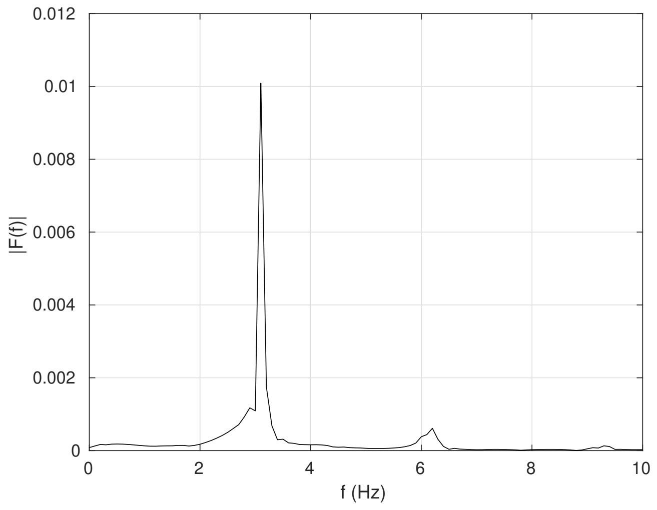
The comparison with the other studies has been made in terms of tip displacement in y-direction and frequency of oscillation and it is reported in Table 4.9. This work presented a tip displacement of 1.04 cm and a frequency of 3.10 Hz ,both of which agree very well with the literature. The frequency peak is clearly visible in the FFT plot in Figure 4.11.
Another small peak is present at a frequency of 6 Hz and has been reported also by other studies as the starting vibration of the beam, which then settles to $\sim$ 3 Hz [40]. Dettmer & Perić [84] discussed about the modulation of this higher frequency with spatial discretization, meaning that with finer meshes the higher frequency has more impact.
This is clearly visible when comparing the two FFT plot: in the *coarse* mesh the peak at 6 Hz has almost disappeared (Figure 4.11a), in contrast with the one in the *fine* mesh (Figure 4.11b).

Table 4.9: Bluff body with flexible cantilever: summary of results and convergence study.

	Average frequency (Hz)	Max tip tip disp. (cm)	Strouhal number
Wall and Ramm [82]	3.08	1.31	0.06
Matthies and Steindorf [87]	2.99	1.33	0.058
Dettmer and Peric [84]	2.96 - 3.31	1.1 - 1.4	0.058 - 0.065
Wood *et al.* [88]	2.78 - 3.125	1.1 - 1.2	0.054 - 0.061
Kassiotis *et al.* [85]	3.17	1.0	0.062
Habchi *et al.* [89]	3.25	1.02	0.063
Froehle and Persson [86]	3.18	1.12	0.062
Hübner *et al.* [44] (rigid cantilever)			0.117
Present study, Coarse	3.11	1.04	0.061
Present study, Fine	3.10	1.08	0.060



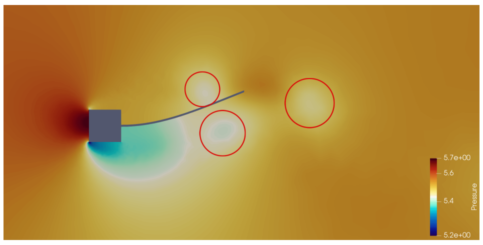
(a) FFT of the *coarse* mesh.



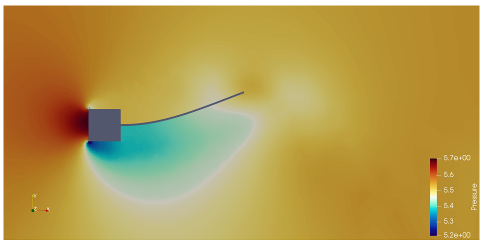
(b) FFT of the *fine* mesh.

Figure 4.11: Bluff body with flexible cantilever: FFT plot representing the oscillation frequency. The larger peak is at $\sim$ 3 Hz, while another small peak is visible at $\sim$ 6 Hz.

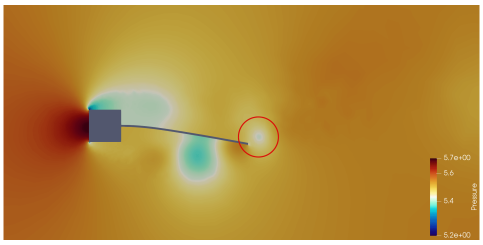
It should also be reported that for the finer discretization, the vortical structures appear more defined (Figure 4.12a and 4.12c) compared to the coarse mesh (Figure 4.12b and 4.12d). Figure 4.13 shows the tip displacement history: the vertical oscillation reaches a periodic regime after 2 seconds in the *coarse mesh*. A slight difference between the two meshes is also visible. The *coarse* mesh (red line) settles to a more evenly distributed oscillation before, while the *fine* one takes a few more seconds to do so. The average amplitude of the displacement is 1.04 cm for the *coarse* mesh and 1.08 cm for the *fine* mesh (refer to Table 4.9).



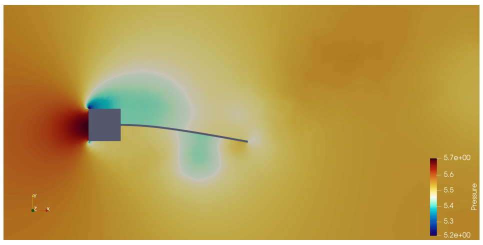
(a) *fine* mesh. 3 vortices highlighted.



(b) *coarse* mesh.

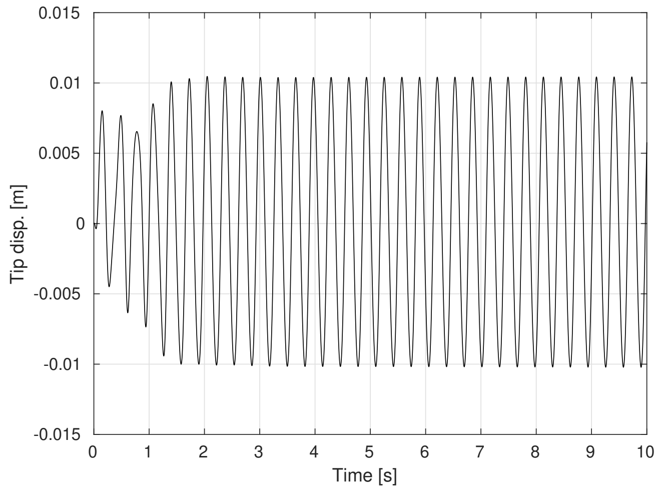


(c) *fine* mesh. 1 vortex highlighted.

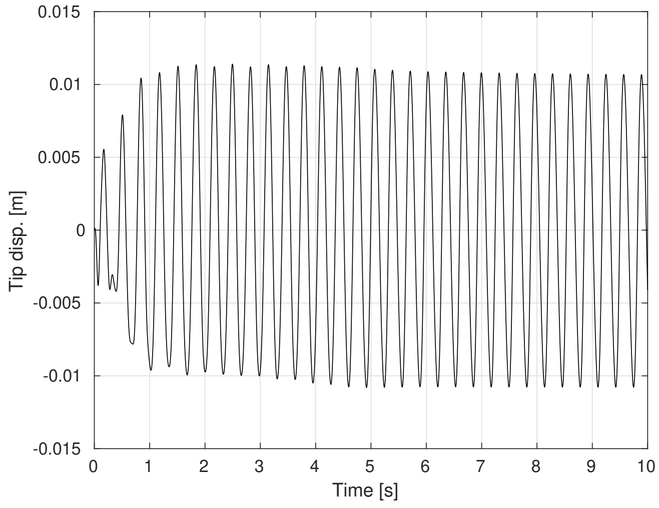


(d) *coarse* mesh.

Figure 4.12: Bluff body with flexible cantilever: pressure contours of the *fine* and *coarse* mesh. The difference between the vortical structures size and refinement is clearly visible.



(a) Plot of the tip displacement in y-direction of the *coarse* mesh.



(b) Plot of the tip displacement in y-direction of the *fine* mesh.

Figure 4.13: Bluff body with flexible cantilever: vertical tip displacement.

4.2.4.3 Performance Analysis

This section is dedicated to an analysis of the various coupling methods. Particularly, with respect to the type of acceleration algorithm and the filter used (refer to Section 3.3.2 for a thorough explanation), in order to see the effect that it has on the performance, especially time. The baseline case, used for the validation with the other studies, used the traditional IQN-ILS quasi-Newton method with a QR2 filter with a threshold of 10^{-2}, which is the suggested one to start with by preCICE documentation. The simulation takes approximately 8 hours using 4 cores.
In Table 4.10 the various acceleration methods with their filter are listed. The mesh used for this scope is the *fine* one. The filter has a great importance during the complicated task of computing the estimated Jacobian matrix. What can be noted is that both QR1 and QR2

filters work similarly, and the best in terms of time managing seems to be QR1 with a threshold of linear Independence of 10^{-4}, which is the suggested one from preCICE website. With QR2, lowering the threshold, reduces significantly the number of iterations, with an average of ~ 6 iterations per time-step. QR1 is more involved and less predictable, as the number of iterations jumps continuously during runtime. Moreover, not every threshold can achieve convergence.
It should be noted that, during this analysis, the relaxation parameter and the number of reused time-steps have been left unchanged to the suggested values of, respectively, 0.1 and 100, however the solver uses only ~ 15 previous time-step during the acceleration process.
Using the *coarse* mesh reduces quite a lot the required time, since with both QR1 and QR2 filters it takes ~ 4 iterations to converge. This could be due to a lesser difference between the solid and the fluid discretization during the grid mapping process.

Table 4.10: Bluff body with flexible cantilever: performance analysis.

IQN-ILS		
Filter	Threshold	Avg. iter
QR2	10^{-1}	6.11
	10^{-2}	9.50
	10^{-3}	diverged
QR1	10^{-4}	8.19
	10^{-5}	9.64
	10^{-6}	diverged

Chapter 5

HART-II

HART-I data analysis revealed the necessity of collecting the wake data, including the vortex formation, aging and its interaction with the blade. In October 2001, the Higher harmonic control Aeroacustics Rotor Test II (HART-II) was performed by a joint effort from US Army AFDD, NASA Langley, German DLR, French ONERA and Dutch DNW. The main objective was to focus on rotor wake measurement using a PIV (Particle Image Velocimetry) technique along with the comprehensive data of blade deflections, airloads, and acoustics. The program focuses on a 40% geometrically and aeroelastically scaled Bo105 hingeless main rotor that was tested in an open-jet anechoic test section of the German-Dutch Wind tunnel (DNW), shown in Figure 5.1.
After the HART-II test in 2001 and since the enstablishment of the HART-II International Workshop in 2005 numerous publications were based on the released data. An overview of these is given in Section 1.2.

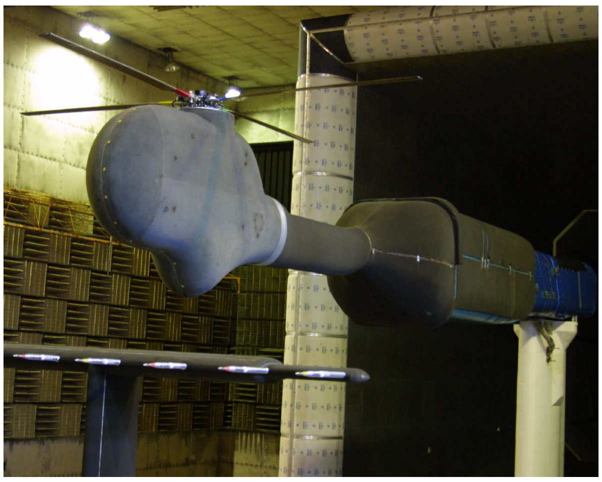


Figure 5.1: HART-II hingeless rotor model in the DNW wind tunnel. Taken from [7].

This Chapter is divided as follows: the HART-II test case description is given in Section 5.1, then in Section 5.2 the structural model is presented. Results are discussed starting from the mode shapes and frequencies analysis, in Section 5.3.1. The hover test case results are presented in Section 5.3.2, while the descent flight condition is analysed in Section 5.3.3.

5.1 Experimental Data

As mentioned before, in the HART-II campaign, a 40% Mach scaled hingeless Bo105 model rotor was used (Figure 5.2). The scaling is done in a way such that the natural frequencies in terms of non-dimensional values (n/rev) are matching the full-scale ones for the first three flapping modes, the first two lead-lag and the first torsion. Since direct scaling doesn't match the Reynolds number at atmospheric pressure, the chord is increased by 10% (and thus the solidity). The rotor is made of 4 rectangular blades with a linear twist, a $5.4mm$ trailing edge tab ($4.46\%c$) of $0.8mm$ thickness ($0.66\%c$) on the NACA23012 airfoil and a precone as in the full-scale rotor. All these parameter can be found listed in Table 5.1. The reference operating condition used throughout the thesis is the HART-II baseline case (BL) as defined in Table 5.1. Two other operational conditions exist: the minimum noise (MN) and the minimum vibration (MV) case. The first, as the name suggests, applies higher harmonic control (HHC) to reduce the BVI noise radiation, while the latter makes use of HHC to reduce rotor vibrations.

Table 5.1: Rotor blade geometry and operating condition

Characteristic	Symbol	Value
Rotor geometry		
Rotor radius	R	2 m
Blade chord	c	0.121 m
Number of blades	N_b	4
Rotor solidity	σ	0.077
Non-dim. root cutout	r_a	0.22
Non-dim. zero twist radius	r_{tw}	0.75
Blade linear twist per span	Θ_{tw}	-8°
Airfoil (trailing edge tab)		NACA23012
Operational Data (BL case)		
Rotational speed	Ω	109.12 rad/s
Hover blade tip Mach no.	M_h	0.639
Rotor shaft angle of attack	α_s	5.3°
Wind tunnel interference angle	Δ_{α_s}	-0.8°
Advance ratio	μ	0.151
Rotor thrust	T	3300 N
Thrust coefficient	C_T	0.00457
Rotor loading coefficient	C_T/σ	0.0594
Rolling moment	M_x	20 Nm
Pitching moment	M_y	-20 Nm
Rotor power	P	18.3 kW
Collective control at r_tw	Θ_{75}	3.8°
Lateral cyclic control	Θ_C	1.92°
Longitudinal cyclic control	Θ_S	-1.34°
Mean steady el. tip twist	Θ_{el}	-1.09°

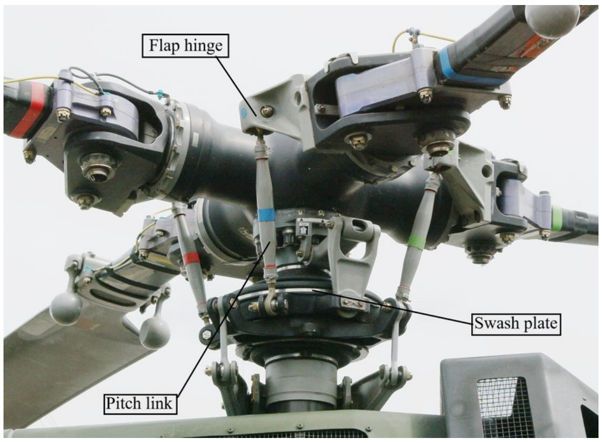
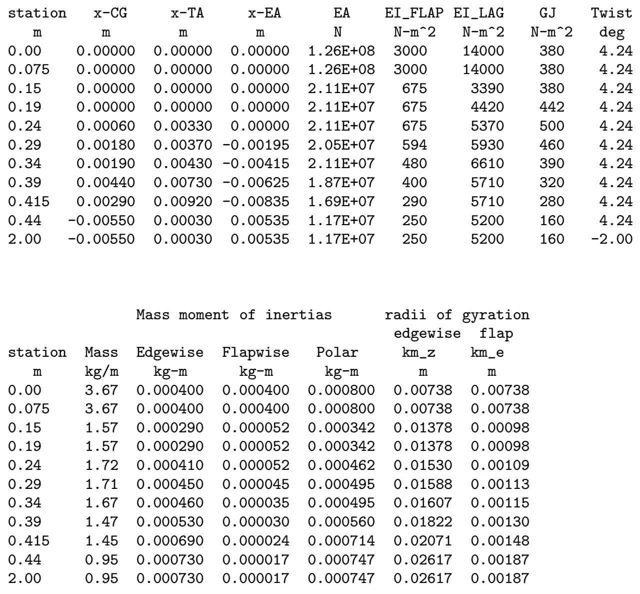


Figure 5.2: Detail of the Bo105 hingeless rotor head.

5.2 Structural Model

In this section the structural model of HART-II rotor is taken into account along with the structural dynamics analysis, required to assess the quality of the discretization.
The Bo105 is a 4 blades hingeless rotor with only one physical hinge for the blade pitch, while the flap and lag main deflections are obtained through the deflection of an *ad hoc* designed flexbeam located at the blade root. In the MBDyn multibody model, the swashplate and the pitch links are represented with rigid bodies, and each blade is modelled using seven geometrically exact finite volume nonlinear beam elements. The beam section stiffness and and mass data of the original Bo105 blades are known and reported in Figure 5.3.



station	x-CG	x-TA	x-EA	EA	EI_FLAP	EI_LAG	GJ	Twist
m	m	m	m	N	N-m^2	N-m^2	N-m^2	deg
0.00	0.00000	0.00000	0.00000	1.26E+08	3000	14000	380	4.24
0.075	0.00000	0.00000	0.00000	1.26E+08	3000	14000	380	4.24
0.15	0.00000	0.00000	0.00000	2.11E+07	675	3390	380	4.24
0.19	0.00000	0.00000	0.00000	2.11E+07	675	4420	442	4.24
0.24	0.00060	0.00330	0.00000	2.11E+07	675	5370	500	4.24
0.29	0.00180	0.00370	-0.00195	2.05E+07	594	5930	460	4.24
0.34	0.00190	0.00430	-0.00415	2.11E+07	480	6610	390	4.24
0.39	0.00440	0.00730	-0.00625	1.87E+07	400	5710	320	4.24
0.415	0.00290	0.00920	-0.00835	1.69E+07	290	5710	280	4.24
0.44	-0.00550	0.00030	0.00535	1.17E+07	250	5200	160	4.24
2.00	-0.00550	0.00030	0.00535	1.17E+07	250	5200	160	-2.00

		Mass moment of inertias			radii of gyration	
					edgewise	flap
station	Mass	Edgewise	Flapwise	Polar	km_z	km_e
m	kg/m	kg-m	kg-m	kg-m	m	m
0.00	3.67	0.000400	0.000400	0.000800	0.00738	0.00738
0.075	3.67	0.000400	0.000400	0.000800	0.00738	0.00738
0.15	1.57	0.000290	0.000052	0.000342	0.01378	0.00098
0.19	1.57	0.000290	0.000052	0.000342	0.01378	0.00098
0.24	1.72	0.000410	0.000052	0.000462	0.01530	0.00109
0.29	1.71	0.000450	0.000045	0.000495	0.01588	0.00113
0.34	1.67	0.000460	0.000035	0.000495	0.01607	0.00115
0.39	1.47	0.000530	0.000030	0.000560	0.01822	0.00130
0.415	1.45	0.000690	0.000024	0.000714	0.02071	0.00148
0.44	0.95	0.000730	0.000017	0.000747	0.02617	0.00187
2.00	0.95	0.000730	0.000017	0.000747	0.02617	0.00187

Figure 5.3: Structural properties of the HART-II blades. Taken from [7].

Figure 5.3 shows that the properties are changing within the first part of the blade, then

from station $x = 0.44$, where the aerodynamic section begins, the structural properties remain constant.
In Figure 5.4, a 2D planar view of the rotor blade is shown and the MBDyn discretization is evinced: seven beams and 13 nodes are used, where four beams are for aerodynamic section, where properties are constant.
Aerodynamic loads of the rotor can be computed using blade-element theory and linear inflow models, while operating in different flight conditions, including hover and forward flight. A C81 table for a NACA20312 airfoil with a tab is also utilised. The HART-II rotor is trimmed to match the target values of the rotor thrust, the hub pitching, and rolling moments, all of which are provided by [36].

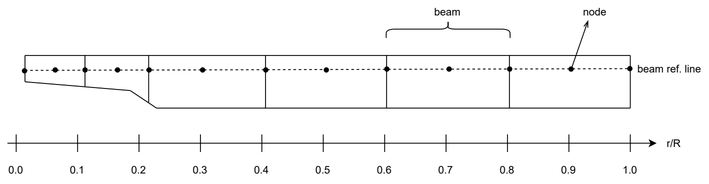


Figure 5.4: MBDyn blade discretization. Beam elements and nodes highlighted.

Regarding the hub modelling, the swashplate is represented with a `total joint` (the fixed part) which also defines collective and cyclic motions and a `revolute hinge` which fixes the relative rotation of the moving part. The Bo105 is a hingeless rotor, hence the only physical hinge is the one responsible for the pitch, modelled as a `revolute hinge` with a `distance` joint element representing the pitch link.

5.3 Results

In this section, the HART-II results obtained from MBDyn are presented.
The blade frequencies analysis is addressed in Section 5.3.1. Then the hover (Section 5.3.2) and the descending flight conditions (Section 5.3.3) are analysed and compared with the literature. The results are in agreement with the experimental data [7] and the other codes [35]. MBDyn offers different inflow models for rotors and three of them have been chosen for a comparison: uniform, glauert and dynamic inflow (explained in Section 3.2.3).

5.3.1 Blade Frequencies Analysis

In order to perform an accurate rotor simulation, the natural oscillation behaviour of the rotor blade *in vacuo* must be assessed. This is performed with the sole use of MBDyn without any type of inflow and provides both the natural frequencies and the coupled mode shapes of the rotor blade. Both depend on the rotational frequency and on the structural properties. Experimental data of natural frequencies are taken from [7], while other reference simulations data are taken from [35].
The frequency diagram and the mode shapes are computed *in vacuo* for a collective pitch setting of 5°, in order to match the reference paper. The nominal rotational speed is $\Omega_{ref} = 109.12$ rad/s. At 100% RPM the sequence of modes (from low frequencies to high) is 1st lag, 1st and 2nd flap, 1st torsion, 2nd lag, 3rd and 4th flap, 2nd torsion, 3rd lag, and 5th flap. These frequencies are listed in Table 5.2, with a comparison between MBDyn model and DLR

experimental data.
The frequency diagram is shown in Figure 5.9 and a comparison with the DLR data from [7] is made. At nominal RPM, most of the flap mode results are very close to the reference. The 1st lag mode is almost overlying the reference for all the speed range, while the 2nd lag of MBDyn shows a higher frequency than the reference. Great differences emerges from higher modes, particularly the 2nd torsion and 3rd lag, especially from lower RPM. This is also observed from ONERA, KU and all the other partners' which are compared in [35]. The variations could be due to different modelling of the blade pith attachment with either a soft-in-torsion element at the blade bold area, or a free end with a torsional spring at this position.

Table 5.2: HART-II: natural frequencies at Ω_{ref}, DLR and MBDyn results. The unit of measure is ω/Ω_{ref}.

Mode	DLR	MBDyn
1L	0.782	0.787
1F	1.125	1.135
2F	2.835	2.840
1T	3.845	3.810
2L	4.592	4.617
3F	5.168	5.080
4F	7.7566	7.874

At 100% RPM the mode shapes are extracted and discussed in the next figures.
The 1st, 2nd and 3d flap modes are almost completely superpositioned (see Figure 5.5). The 4th and 5th flap shows some differences especially near the root (see Figure 5.6).
Good agreement is found also for the lag mode shape, though some differences are observed in the 3rd lag (see Figure 5.7). The torsion mode shapes (Figure 5.8) are quite similar to the reference, with the differences at the root, where the different boundary conditions are applied. These differences are observed also in the other codes compared in [35]. MBDyn results are very similar to the ONERA ones.

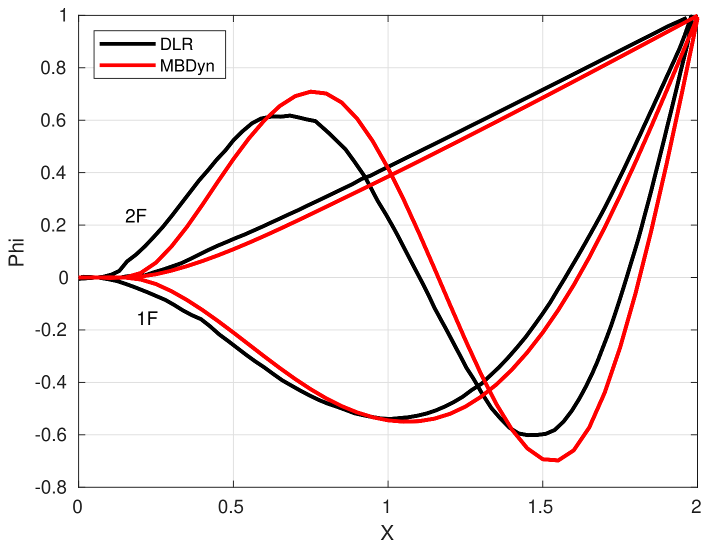


Figure 5.5: Flap mode shapes: first, second and third.

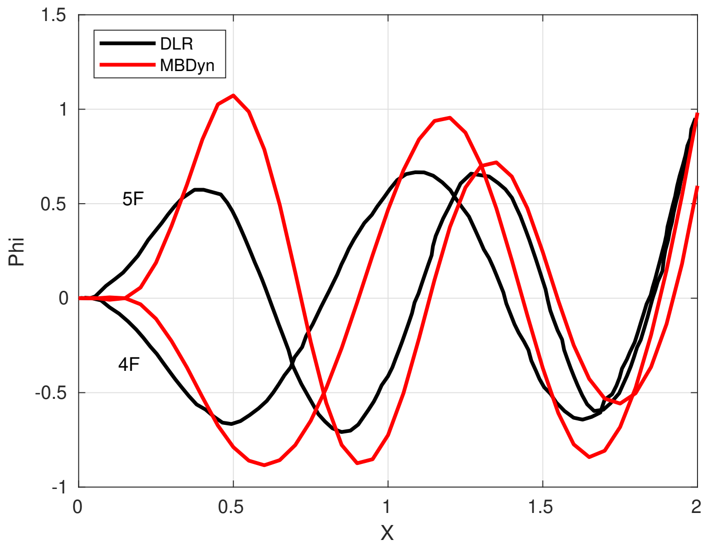


Figure 5.6: Flap mode shapes: fourth and fifth.

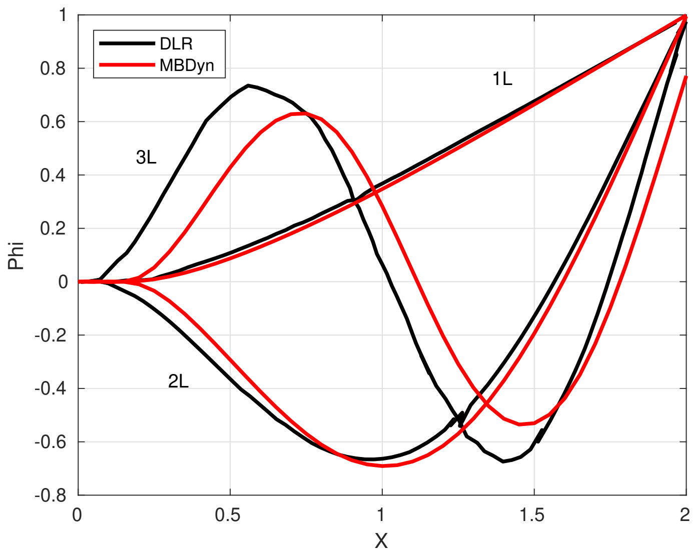


Figure 5.7: Lag mode shapes.

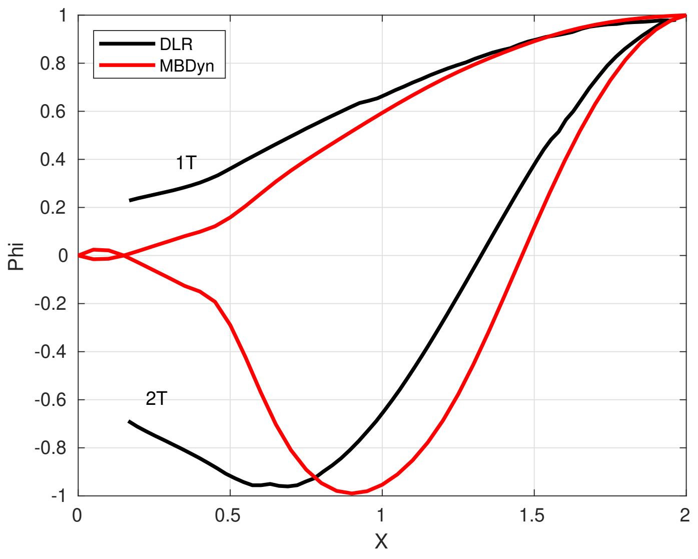


Figure 5.8: Torsion mode shapes.

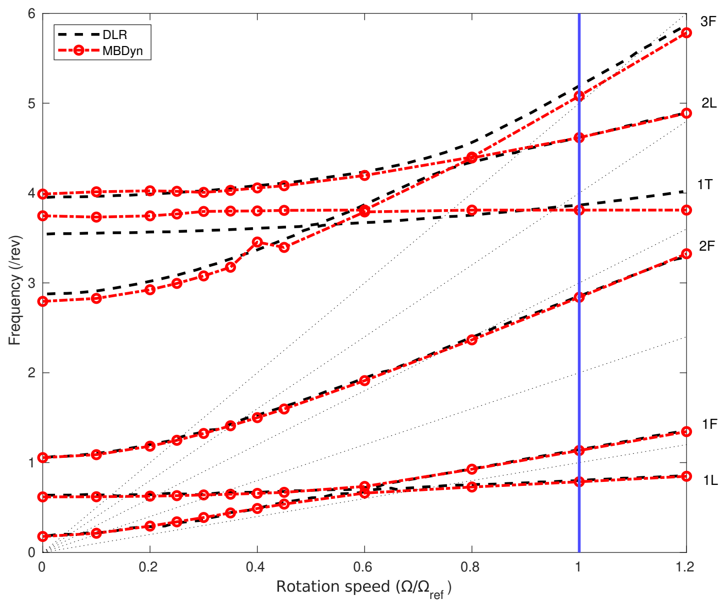


Figure 5.9: Frequency diagram of the Bo105 model rotor. MBDyn (colored) and DLR (black) data compared. The dotted lines are the excitation frequencies [n/rev].

5.3.2 Hover

The HART-II test campaign does not include the hover condition, hence no validation with experimental data can be made. This simulation is used only as a first assessment of the HART-II multi-body model. The controls are listed in Table 5.3. The shaft is vertical and no roll or pitch moments are expected, since the rotor is trimmed to stay in hover.

Table 5.3: HART-II hover controls.

Collective	3.8°
Lateral cyclic	0°
Longitudinal cyclic	0°
Rotor shaft angle of attack	0°

A rotor thrust of 1200 N is obtained, with no pitching or rolling moment. The three different inflow models yield to identical results, which is expected. In Figure 5.10 it is clearly visible that after a brief transient, the rotor thrust settles on the nominal value, whereas pitch and roll are null (Figures 5.11 and 5.12).

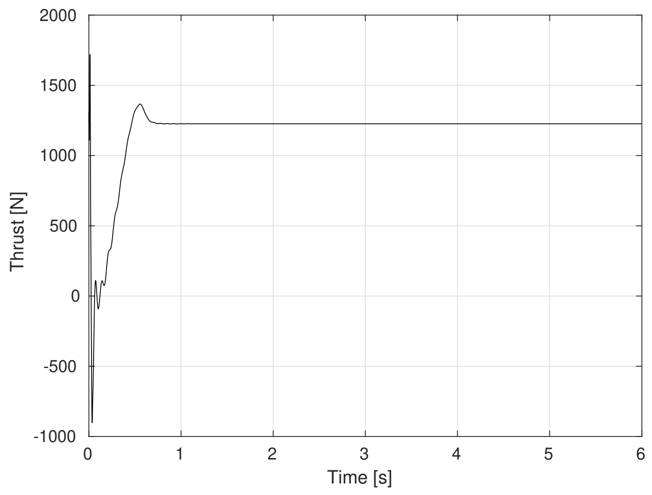


Figure 5.10: HART-II hover: thrust.

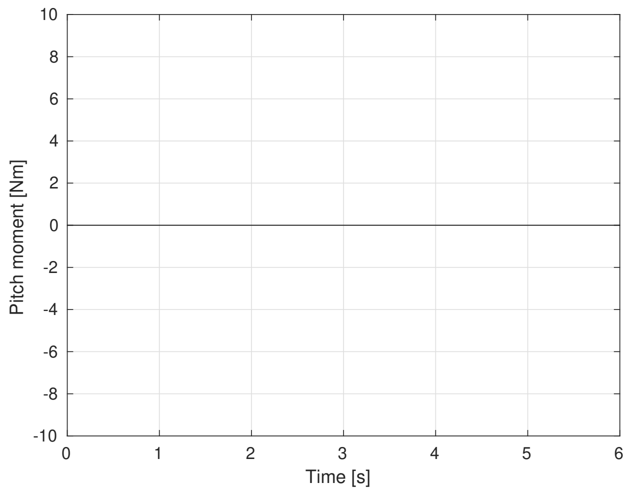


Figure 5.11: HART-II hover: pitch moment.

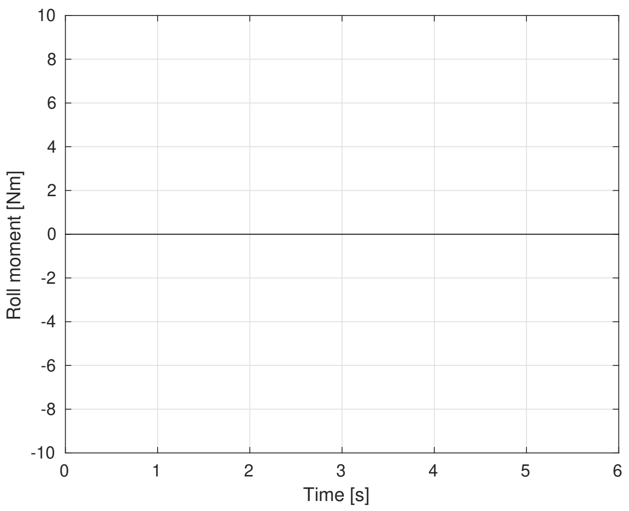


Figure 5.12: HART-II hover: roll moment.

5.3.3 Descent Flight

The descent flight condition is then analysed. Validations are performed for the baseline rotor (BL) at an advance ratio of 0.151. The rotor is trimmed to reach the requested thrust with the controls listed in Table 5.4. The shaft angle is tilted upward of 4.5°.

Table 5.4: HART-II descent controls.

Collective	3.8°
Lateral cyclic	1.92°
Longitudinal cyclic	-1.34°
Rotor shaft angle of attack	4.5°

The predicted thrust is in agreement with the experimental data as well as with the prediction of the other codes involved in the HART-II workshop. The pitch and roll moments are expected to be improved after the coupling with SU2. In Table 5.5 it is given a comparison between three MBDyn inflow models used in this context.

Table 5.5: Comparison of rotor thrust, pitch and roll moments for different inflow models. Descent flight.

	Thrust	Pitch	Roll
DLR (reference)	3300 N	-20 Nm	20 Nm
Uniform inflow	3326.83 N	-273 Nm	465 Nm
Glauert inflow	3364.71 N	-86 Nm	163 Nm
Dynamic inflow	3263.37 N	-177 Nm	79 Nm

Figure 5.13, 5.14 and 5.15 show the time histories of flap, lead-lag and elastic torsion at the blade tip. MBDyn is compared with the experimental results as well as with other codes

involved in the HART-II workshop. The flap response is in very good agreement, as shown in Figure 5.13. The lead-lag, instead, is similar to the experimental result but there is a constant offset between the predicted and experimental result for all the codes (Figure 5.14). The elastic torsion (Figure 5.15) predicted by MBDyn inflow models is quite different from the experimental data and the other comprehensive codes. The torsion mode highly depends on the aerodynamic moments and an improvement can be expected with the coupling approach. Figure 5.16 presents the minimal differences in the torsion elastic response with respect to the inflow models, namely Glauert, dynamic and uniform.

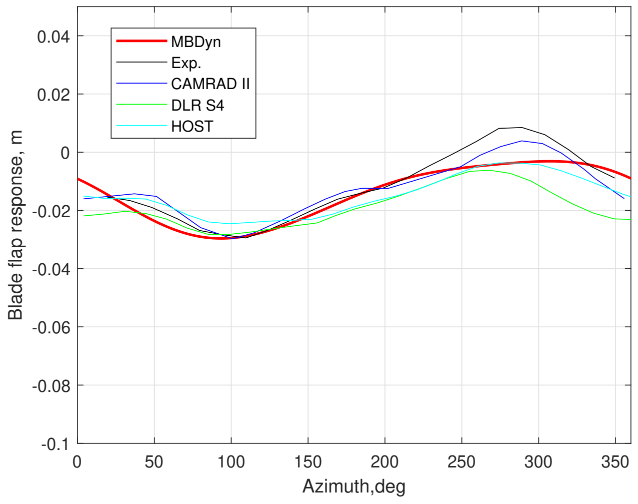


Figure 5.13: Flap tip response.

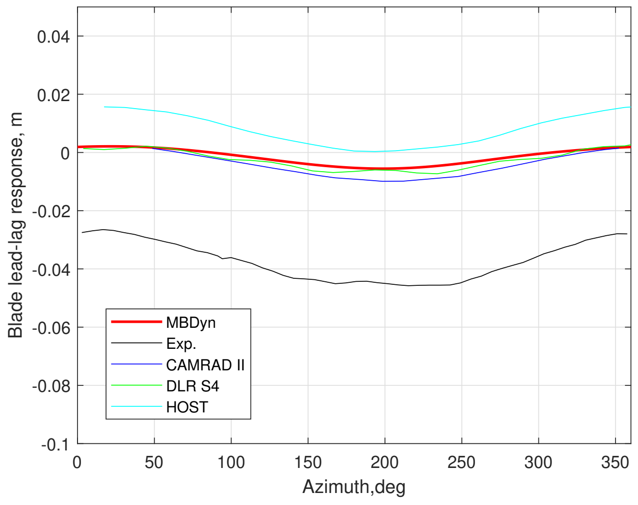


Figure 5.14: Lead-lag tip response.

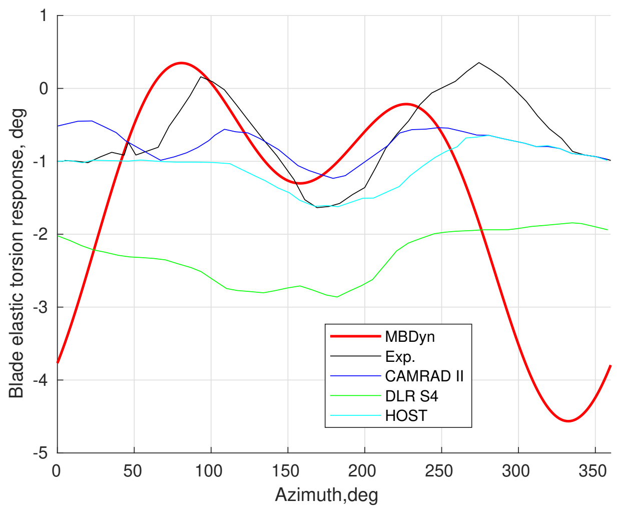


Figure 5.15: Elastic torsion tip response.

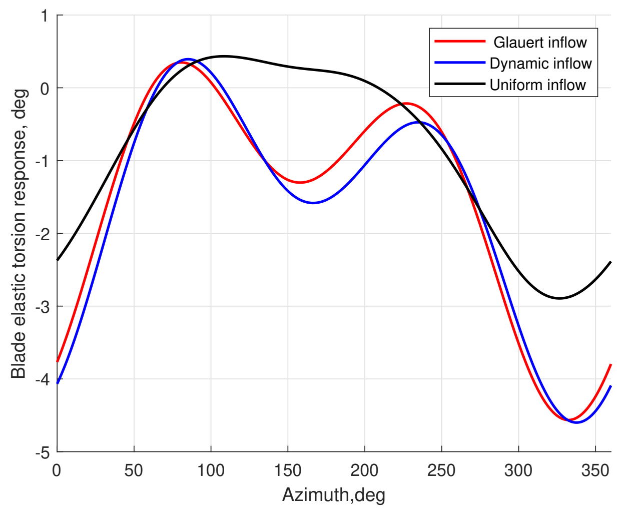


Figure 5.16: Elastic torsion tip response.

Figure 5.17 presents the comparison of sectional normal load, M^2C_n, obtained at 87% span-wise location. Dynamic inflow is used for the MBDyn data. The airloads show an acceptable correlation, although the need for a coupling with a CFD software is clear, particularly in regards to BVI peaks.

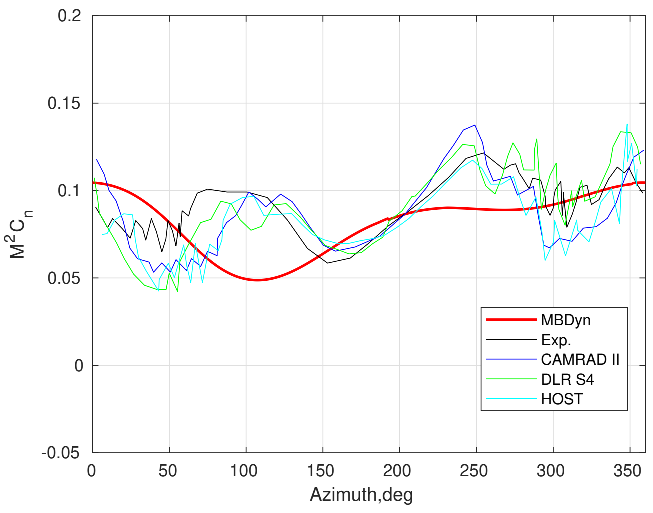


Figure 5.17: Comparison of section normal forces at 87% station.

Chapter 6

Conclusions and Future Developments

The aim of this thesis was to develop an open-source comprehensive solver using MBDyn and SU2 with the coupling library preCICE. The need for an open-source solver arises from the fact that many existent couplings employ two closed-source solvers and do not allow any modification of the source code. In addition to this, these solvers are not freely available so that any developer could modify and improve the code.
In order to perform a coupled simulation between two solvers, preCICE has to be integrated with the solver and this is done by the adapter. An adapter for SU2 was already available for Version 6.0.0. In the context of this thesis, the adapter has been ported to the newest SU2 version, which presents many differences in the routines. The design of the MBDyn adapter started from a version already developed by Politecnico di Milano [5] and specifically made for coupling with DUST, a mid-fidelity aerodynamic solver.
The adapter has been greatly modified, firstly to enable the connection with SU2, then also to allow for higher compatibility and flexibility. The solver-specific parameters are read from the configuration file (JSON) and the user does not have to specify them elsewhere. Now the adapter can support 2D and 3D domains, with more than one FSI interface.
The verificatiton of the proposed adapter has been made with two test cases. The first one was a flexible vertical flap immersed in an Euler flow. The structural parameters combined with the freestream velocity resulted in very large displacements, which served as a test both for the robustness of the mesh mapping and for the data exchange. The results were compared with other couplings present in literature, all of which employed preCICE as the coupling library.
The second, more involved, test case was a flexible flap behind a bluff body immersed in a Navier-Stokes laminar flow, a largely studied case in the context of fluid-structure interaction simulations [82]. Initially, vortex shedding generates from the square as a result of the sharp edges. Then, this vortex trail causes the flap to start oscillate, which results in another vortex trail detaching. The results showed good agreement with the literature and were used also to compare the various implicit coupling methods offered by the coupling library.

6.1 Future Developments

The final aim of the thesis was to apply the open-source comprehensive solver to rotorcraft simulations. For this reason, HART-II test case has been chosen as a starting point and a preliminary analysis of the structural model was made. The multi-body model was compared to the experimental data of the HART-II campaign and with other codes present in literature.

First, the frequencies of the blade *in vacuo* were assessed and a fan-plot obtained for the frequencies up to the 3^{rd} flap. Then, hover and descending flight conditions were assessed with MBDyn inflow models. The preliminary results obtained with the mid-fidelity aerodynamic model showed an acceptable accuracy, which can be raised by the coupling with SU2.
The next step will be the coupled simulation, first of the hover condition, then of the descent flight with high advance ratio, which is the most interesting one, where BVI and other complex aeroelastic phenomena can be observed and the accuracy of the CFD solver can be appreciated.
In order to simulate rotors, the displacements from MBDyn should be collected in rotating reference frames, to allow the mesh to rotate other than deform. This procedure was implemented in the SU2 Version 7.1.1 adapter and it is already compatible with the RBF mesh deformation implemented in the software.
One possible step that has to be made in order to ease the coupled simulation is to create an intermediate mesh to map between the 1D beam reference line of MBDyn and the 3D very refined mesh of SU2. This will help the mapping process and the overall accuracy, since one of the crucial step of FSI coupling is the interface between the fluid and the solid mesh.
Regarding the MBDyn adapter, an advancement would be adding other type of elements, such as plate elments, i.e. shells or membranes. These type of elements are already implemented in another version of the MBDyn adapter, which can be found on the preCICE repository[1] and could be added in order for the adapter to be fully general.
In the context of rotorcraft simulations, a method for exchanging data in a loosely-coupled way should be implemented, as preCICE supports only tight-coupling at present time.

[1]MBDyn adapter, from preCICE github repository: `https://github.com/precice/mbdyn-adapter`

Appendix A

Appendix A: MBDyn Adapter Architecture

In order to perform a coupled simulation between two solvers, preCICE has to be integrated with the solver via API (Section 3.3.4) and what results from this operation is the *adapter*, as depicted in Figure A.1.

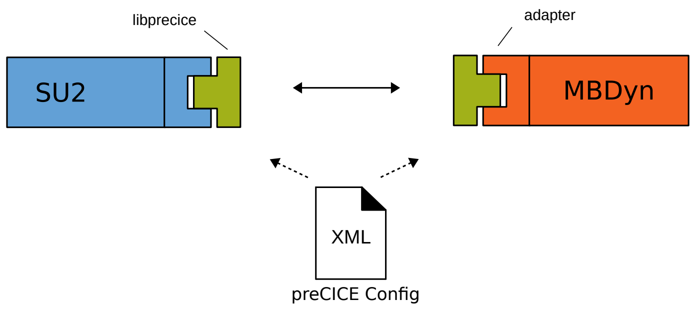


Figure A.1: Schematic representation of the coupling between SU2 and MBDyn via preCICE. The solver code and the preCICE library (libprecice) are glued together by the adapter.

In this Appendix, the adapter for the communication between preCICE and MBDyn is presented. Section A.1 gives a description of the general structure of the adapter, then the two main classes which constitute the adapter are explained in Section A.2 and A.3. The simulation is set up by the `run.py` script, presented in Section A.4. Finally, the only file that the user has to modify is explained in Section A.5.

A.1 Design of the Adapter

The MBDyn adapter is written in Python, which is also the language of the chosen MBDyn API. The Python wrapper is defined in MBDyn's library `mbpy`.
The MBDyn adapter was originally created by Mikko Folkersma at Delft University of Technology [90] and it concerned only `membrane` elements, as the main objective was to do a FSI simulation of kites [91] .
A first modification was introduced by Politecnico di Milano with the aim of using it for rotors

and wind turbines [5] and to couple it with DUST, a mid-fidelity aerodynamic solver developed again at Politecnico di Milano [92]. This led to the introduction of `beam` elements and generated a greatly modified adapter [1].
In the context of this thesis, the existing code has been further modified, in order to make it more user-friendly and suitable for the coupling with SU2, which requires different kinematic variables. While DUST accepts position, orientation, velocity and angular velocity, SU2 accepts only displacements. The additions to the existing code developed for the MBDyn-DUST coupling are:

- write displacements of the nodes instead of positions;
- accept either 2D or 3D simulations;
- accept more than one FSI interface. This concerns the method in which multiple interfaces are classified in SU2 adapter (refer to B).

The adapter does not require any installation and this is one of its main assets. It is composed of 2 layers of communication, `mbdynInterface.py` and `mbdynAdapter.py`:

- `mbdynInterface.py` collects the interface needed for converting data between MBDyn API and preCICE library,
- `mbdynAdapter.py` collects the adapter class and methods for exchanging data with SU2 through preCICE library.

MBDyn is then executed through a case-dependent Python script, whose variables can be set by a `JSON` file. In Figure A.2 the adapter structure is shown.

```
/
├── structural/
│   ├── mbdynInterface.py
│   ├── mbdynAdapter.py
│   ├── run.py
├── config.json
```

Figure A.2: Simulation folder structure, where only the MBDyn adapter files are visible. In the real case, there would be also the fluid solver folder, MBDyn simulation files and preCICE configuration.

A.2 mbdynInterface.py

`mbdynInterface.py` is the first layer of communication and it contains all the methods responsible for collecting the data from MBDyn and converting them into preCICE language.
At the beginning of the file (Listing A.1, all the required methods are imported. The first difference from the MBDyn-DUST code is the presence of `json` module: `mbdynInterface.py`

[1]MBDyn-DUST code: `https://gitlab.com/davideMontagnani/dust-mbdyn`

reads from a configuration file called `config.json` two types of information: the number of FSI interfaces and the domain dimension, either 2D or 3D (Listing A.7).

```python
import sys
from mbc_py_interface import mbcNodal

import precice
from precice import *

import numpy as np

import json
json_file = open('./../config.json')
variables = json.load(json_file)
json_file.close()

dm = variables['dimensions']
x = variables['FSI-interfaces']
```

Listing A.1: Initial part of mbdynInterface.py.

The code contains the class `mbdynInterface`, which is made of some methods and fields:

- `.data` is a dictionary which contains the kinematic variables that are exchanged through MBDyn API, namely `Position`, `DisplacementDeltas` and `Forces`. Each item can be either a vector or a scalar and has attached the type `read` or `write` to indicate whether it is a force read from SU2 or a displacement written by MBDyn. The true variable that is passed to SU2 is `DisplacementDeltas`, but in order to initialize the simulation and to retrieve the nodes coordinates, also `Position` has to be collected;
- `.socket` is an inner class, which contains the parameters to set up the socket (Listing A.2). `Host` and `port` are useful if an Internet socket is desired. If neither of them is specified, a local host is assumed. `timeout` indicate for how long the socket waits for the solver connection and a value of `-1` removes this limit. if `verbose` is set to `1`, a detailed list of the processes is given. `data and next` signalises MBDyn to expect forces and send positions. `labels`is only important if a reference node was described in the .mbd file which instructs the peer to send the forces and their orientations relative to the reference node. `nnodes` indicates the number of exposed nodes through the `external structural force` element. This value should be set in the `config.json` file, which will be discussed in Section A.5 and must coincide with the number defined in the syntax of the external structural force;

```python
  #> Inner class -----------------------------------------------
  class Socket:
    """ Class containing the socket parameters """
    """ for comm between MBDyn and mbc_py       """
    def __init__(self, \
                 path="", host="", port=0, \
                 timeout=-1, verbose=0, data_and_next=1, \
                 refnode=0, nnodes=1, labels = 0, rot=0x100, \
                 accels=0 ):
      self.path          = path
      self.host          = host
      self.port          = port
      self.timeout       = timeout
      self.verbose       = verbose
```

```
        self.data_and_next = data_and_next
        self.refnode       = refnode
        self.nnodes        = nnodes
        self.labels        = labels
        self.rot           = rot
        self.accels        = accels
```

Listing A.2: Socket set up

- `nodal` is an object of `mbcNodal` class defined in the `mbc_py_interface` (MBDyn Python library). Within this class the methods `.negotiate()`, `.recv()` and `.send()` are instantiated: the first is used to access the nodes, the second writes the kinematics to the external solver and the third sends back the forces to MBDyn;
- `initialize()` is used to establish a connection with MBDyn;
- `finalize()` closes all communications;
- `refConfigNodes()` reads the initial configuration of the nodes needed to compute the displacements. The input file is `refConfigNodes.in` and it has to be compiled with an ordered list of the coordinates of all the nodes (Listing A.3).

```
0.0000 0.0000
0.0000 0.1000
0.0000 0.2000
0.0000 0.3000
0.0000 0.4000
0.0000 0.5000
0.0000 0.6000
0.0000 0.7000
0.0000 0.8000
0.0000 0.9000
0.0000 1.0000
```

Listing A.3: refConfigNodes example file. x y coordinates of 11 nodes.

A.3 mbdynAdapter.py

`mbdynAdapter.py` is the second layer of communication, the one that takes care of the actual coupling and connects SU2 with MBDyn. The basics steps are:

- provide access to all the necessary fields for the coupling (`Position` and `Forces`),
- initialize the coupling data,
- compute the displacements from the nodes position,
- steer the simulation,
- finalize the simulation.

The class `MBDynAdapter` contains various fields and methods:

- `Participant` class sets the name of the solid solver, MBDyn;

- `Mesh` class similarly sets the name of the mesh. This name must coincide with the one declared in the `precice-config.xml`(Section 3.3.5);
- `__init__()` constructor initializes the object of `MBDynAdapter` class and the communication with SU2. It reads all the fields previously described and sets up the mesh;
- `getDisplacements()` method computes the displacements from the nodes position;
- `runPreCICE()` method initiates the actual coupling loop, which is explained in detail below.

The `runPreCICE()` method contains the actual loop of the data exchange. Listing A.4 reports all the cycle. Here parameters such as the MBDyn time-step and the preCICE time-step (`dt_precice`) are retrieved: if they do not match, the smaller one is chosen. This way, the structural solver can have a lower time-step than the coupled fluid solver, which is also known as “subcycling”. It is important to mention that forces and displacements are only exchanged at each time step and not at each substep.
The execution phases include:

1. reload state if previous iteration did not converge with `write_iteration_checkpoint()` and `read_iteration_checkpoint()` (implicit coupling). This is the first thing done in the `while` loop with the boolean argument `isCouplingOngoing()`, which is the preCICE way of controlling if the simulation is still going on
2. retrieve nodes position from MBDyn and compute displacements
3. write nodes displacements to SU2 with `write_block_vector_data`
4. read forces from the fluid solver with `read_block_vector_data`
5. check convergence: iterate or finalize the time-step with `is_action_required`
6. preCICE exits the loop and terminates the communication with `finalize()`, otherwise the loop is repeated.

```python
def runPreCICE(self):

  dt_set = variables['mbdyn-timestep']

  n = self.mbd.socket.nnodes; nd = 3

  cowic = precice.action_write_iteration_checkpoint()
  coric = precice.action_read_iteration_checkpoint()

  dt_precice = self.dt_precice

  force = np.zeros((n, nd))

  t = 0.
  previousDisplacements = self.getDisplacements()
  is_ongoing = self.interface.is_coupling_ongoing()
  while ( is_ongoing ):

    if ( self.interface.is_action_required( cowic ) ):
      pos_t   = self.mbd.data['Position']['data'][:,[0,dm-1]]

      for k in range(0,x):
          delta_t_k = []
          delta_t_k = self.mbd.data['DisplacementDeltas%d'%(k)]['data'][: :]
      self.interface.mark_action_fulfilled( cowic )

    #> Set MBDyn nodal values of forces and moments
    for i in np.arange(n):
        for k in range(0,x):
            self.mbd.nodal.n_f[i*dm:(i+1)*dm]=self.mbd.data['Forces%d'%(k)]['data'][i,:] # force[i,:]

    dt = min( dt_set, dt_precice )

    #> === Communication with MBDyn ===========================
    if ( self.mbd.nodal.send(False) ):
      break

    # Receive data from MBDyn
    if ( self.mbd.nodal.recv() ):
      print('**** break, after nodal.recv() ****'); break

    #> Read position and velocity from MBDyn
    displacements = self.getDisplacements()
    relDisplacements = displacements - previousDisplacements
    self.mbd.data['Position']['data'] = np.reshape( self.mbd.nodal.n_x, (n, nd))

    for k in range(0,x):
        self.mbd.data['DisplacementDeltas%d'%(k)]['data'] = relDisplacements

    #> Write to SU2
    for fie in self.p['fields']:
      if ( self.p['fields'][fie]['io'] == 'write' ):
        if ( self.p['fields'][fie]['type'] == 'scalar' ):
          self.interface.write_block_scalar_data( \
                                       self.p['fields'][fie]['id'], \
                                       self.p['mesh']['node_id'],   \
                                       self.mbd.data[fie]['data'] )
        if ( self.p['fields'][fie]['type'] == 'vector' ):
          self.interface.write_block_vector_data( \
                                       self.p['fields'][fie]['id'], \
                                       self.p['mesh']['node_id'],   \
                                       self.mbd.data[fie]['data'] )

    dt_precice = self.interface.advance(dt)
    is_ongoing = self.interface.is_coupling_ongoing()

    #> Receive data from SU2 and set nodal.n_f field
    for fie in self.p['fields']:
      if ( self.p['fields'][fie]['io'] == 'read' ):
        if ( self.p['fields'][fie]['type'] == 'scalar' ):
          self.mbd.data[fie]['data'] = \
            self.interface.read_block_scalar_data( \
                                       self.p['fields'][fie]['id'], \
                                       self.p['mesh']['node_id']    )
        if ( self.p['fields'][fie]['type'] == 'vector' ):
            self.mbd.data[fie]['data'] = \
              self.interface.read_block_vector_data( \
                                    self.p['fields'][fie]['id'], \
                                    self.p['mesh']['node_id']     )

      #> Check convergence: iterate or finalize the timestep
      if ( self.interface.is_action_required( coric ) ): # dt not converged
        self.mbd.data['Position']['data'] = pos_t

        for k in range(0,x):
            self.mbd.data['DisplacementDeltas%d'%(k)]['data'] = delta_t_k
        self.interface.mark_action_fulfilled( coric )
      else: # dt converged
        previousDisplacements = displacements.copy()

        if ( self.mbd.nodal.send(True) ):
          break
        # Receive data from MBDyn
        if ( self.mbd.nodal.recv() ):
          print('**** break, after nodal.recv() ****'); break
        t = t + dt

    self.interface.finalize()
    print(' Finalize coupling ')
```

Listing A.4: Coupling loop

A.4 run.py

This section illustrates the role of the case dependent file `run.py`, which is the one actually called to start the simulation.
In the first part of the script, all the required Python and preCICE libraries are imported, including the `json` package, useful to read the `config.json` file (Section A.5) where all the case dependent variables can be set. The path to MBDyn Python API `mbpy` has to be set as well and the variables for socket communication are initialized. Last but not least,`mbdynInterface.py` and `mbdynAdapter.py` are imported (A.5.

```
import time
import sys;

import json
json_file = open('./../config.json')
variables = json.load(json_file)
json_file.close()

dt_set = variables['mbdyn-timestep']
nnodes = variables['exposed-nodes']
file_name = variables['input-file-mbdyn']

# set to path of MBDyn support for python communication
sys.path.append('/usr/local/mbdyn/libexec/mbpy');

import os;
import tempfile;
tmpdir = tempfile.mkdtemp('', '.mbdyn_');
```

```
path = tmpdir + '/mbdyn.sock';

os.environ['MBSOCK'] = path;
os.system('mbdyn -f %s -o output > output.txt 2>&1 &' %(file_name)) ;

from mbc_py_interface import mbcNodal
from numpy import *
import numpy as np

import precice
from precice import *

from mbdynInterface import MBDynInterface
from mbdynAdapter   import MBDynAdapter
```

Listing A.5: First part of run.py

The interface between MBDyn and preCICE is initialized with `MBDynInterface()` class. Then, the adapter is initialized and the `runPreCICE()` method is called to start the simulation (A.6).

```
#> Initialize MBDyn/mbc_py interface: negotiate and recv()
mbd = MBDynInterface()
mbd.initialize( path=path, verbose=1, nnodes=nnodes, accels=1, \
                dumpAuxFile=True )

print(" Initialize MBDyn adapter ")
adapter = MBDynAdapter( mbd )

if ( adapter.debug ):
  print(' participant: ', adapter.p["name"] )
  print(' solver      : ', adapter.p["mesh"]["name"] )
  print(' fields      : ', adapter.p["fields"] )

#> Start coupled simulation with PreCICE
adapter.runPreCICE()
```

Listing A.6: Second part of run.py

A.5 config.json

The `config.json` is the only file that the user has to modify when setting up the simulation (Listing A.7).

```
{
    "mbdyn-timestep": 0.01,
    "exposed-nodes": 26,
    "dimensions": 2,
    "FSI-interfaces": 2,
    "input-file-mbdyn": "wing"
}
```

Listing A.7: config.json file

Below, an explanation of the variables that have to be set is given:

- `mbdyn-timestep` is the value set as MBDyn time-step

- `exposed-nodes` is the value of the nodes exposed through the external structural force
- `dimensions` is used to set the simulation in 2D or 3D
- `FSI-interfaces` lets the user set the number of FSI interfaces
- `input-file-mbdyn` is the name of the main MBDyn input file.

Appendix B

Appendix B: SU2 Adapter Architecture

The SU2 adapter was originally developed by Alexander Rusch and presented in his Bachelor's Thesis [6], where a thorough description of the adapter is given. The adapter was created for SU2 version **4.1** and has been updated by the author until version **6.0.0**. Since the latest release of SU2 introduced many changes in the code structure, the adapter has been ported to version **7.1.1 "Blackbird"** in the context of the present thesis. In Figure B.1, a schematic illustration of the adapter structure is given.

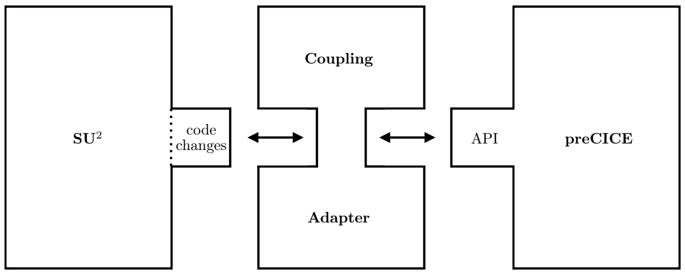


Figure B.1: Schematic representation of the SU2 adapter. Taken from [6].

Unlike the MBDyn adapter (Appendix A), SU2 adapter introduces some changes to the existing solver code and requires a *ad hoc* installation.
A C++ class named *Precice* made of a header file *precice.hpp* and a source file *precice.cpp* is the addition to SU2 original code. It encapsulates all coupling related activities and keeps them separated from the solver source code. As said before, some routines have to be modified in order to encapsulate preCICE. These changes are confined into $SU2\_CFD$ module, which contains the solvers for direct, adjoint problems and into the configuration file setup (*CConfig.cpp* and *CConfig.hpp*).
The usage of the adapter is completely embedded into the normal SU2 execution, therefore no changes to the common way of starting a simulation are required, whether it is a single-physics or a multi-physics problem. All options concerning the usage of preCICE (e.g. switching it on or off, specifying the name of the FSI interfaces and of the preCICE configuration file) can be set in the SU2 configuration file, in order to minimize even more the impact that the adapter

has on the suite and on the user.
This Appendix is organized in the following way: in Section B.1 the changes in the SU2 configuration file are presented, with an explanation of all the new features. Then, in Section B.2, the additions to the main routine $SU2\_CFD$ and the configuration setup files are described. Finally, in Section B.3, the adapter class *Precice* is thoroughly described, enancing the changes made in the context of this thesis in order to port the adapter to the newest SU2 version.

B.1 SU2 Configuration

Every option needed for the simulation is set up in the SU2 configuration file. Here, the user chooses, for example, the type of solver, the time-step and the freestream values.
The file has the extension `.cfg`. The syntax is very simple: an option in the file is written as `option_name = value`, where `option_name` is the name of the option and value is the desired option value.
For instance, the unsteady time step is set as in Eq. B.1 and the inlet is indicated as in Eq. B.2.

$$\text{UNST\_TIMESTEP} = 0.001 \tag{B.1}$$

$$\text{MARKER\_INLET} = (\text{ inlet }) \tag{B.2}$$

The new options introduced by the adapter for the usage of preCICE are listed in the following:

$$\text{PRECICE\_USAGE} = \textbf{NO}, \text{YES} \tag{B.3}$$

$$\text{PRECICE\_CONFIG\_FILENAME} = \text{./../precice-config.xml} \tag{B.4}$$

$$\text{MARKER\_PRECICE} = (\text{ flap1, flap2 }) \tag{B.5}$$

$$\text{PRECICE\_VERBOSITYLEVEL\_HIGH} = \textbf{NO}, \text{YES} \tag{B.6}$$

$$\text{PRECICE\_LOADRAMPING} = \textbf{NO},\ \text{YES} \tag{B.7}$$

$$\text{PRECICE\_LOADRAMPING\_DURATION} = 10 \tag{B.8}$$

The PRECICE_USAGE (Eq. B.3) is a flag used for determining wheter a simulation should be run with or without preCICE. The default value is NO, which means that a normal single-physics simulation is set. The other options are reasonable only if this flag is set to YES. PRECICE_CONFIG_FILENAME (Eq. B.4) specifies the name of the preCICE cofiguration file (ref. Section 3.3.5) and its location.
MARKER_PRECICE is used to set the marker name of the FSI interfaces. This option is slightly modified from the original SU2 adapter [6]: instead of having to enter a default and not changeable name for the FSI interfaces, the new version lets the user decide which marker name to use. For example, referring to the first validation case (ref. Section 4.1) where the FSI interface was the vertical flap, the user should specify here the boundary marker set also in the mesh file. In the first version of the adapter, the only name accepted was **"wetSurface"**.
Another useful improvement is that now the adapter calculates automatically the number of interfaces, so if the domain is composed of 4 blades, as in Section 5, the user just has to set the 4 blades name in MARKER_PRECICE.
The option PRECICE_NUMBER_OF_WETSURFACES is therefore deprecated.
In order to have more insight into the sequence of operations that preCICE is doing during the simulation, the user can set PRECICE_VERBOSITYLEVEL_HIGH flag to YES (Eq. B.6).
Another useful option is the PRECICE_LOADRAMPING (Eq. B.7), which helps to ease the

initial part of the simulation using a ramp function to pass the forces to the structural solver. It lets the user decide also for how many iteration this should be done (Eq. B.8).
The dynamic mesh capabilities should also be allowed:

$$\text{SURFACE\_MOVEMENT} = \text{DEFORMING} \tag{B.9}$$

And the marker of the surface that should be deformed has to be specified:

$$\text{MARKER\_MOVING} = (\text{ flap1, flap2 }) \tag{B.10}$$

B.2 Changes in SU2 Routine

The usage of preCICE within the SU2 suite impacts also the solver routine, but in a minimal way, with this being the goal of the adapter author [6]. The key feature is that the solver executable can be run with or without preCICE, without having to recompile it. While porting the adapter to the latest SU2 version, the same philosophy has been followed.
In this section, the additions and modification to all the modules will be highlighted, particularly with regard to the new SU2 code structure.
The only additional variables required by SU2_CFD are `max_precice_dt`, `dt` and calling the *Precice* class:

- `CDriver.cpp` (and its header), responsible for instantiating all of the geometry, physics packages, and numerical methods needed to solve a particular problem
- `CSinglezoneDriver.cpp` (and its header), contains the main subroutines for driving single-zone problems

In the `CSinglezoneDriver.cpp` preCICE functions have to be called in the `while` loop. The first thing that has to be set is the `max_precice_dt`, which allows the adapter to know if sub-cycling is necessary, that is if SU2 timestep coincide with the preCICE one (Section 3.3.1). Listing B.1 shows the insertion of the declarations and memory allocation of preCICE-related variables, as well as the startup of the coupling procedure.

```
precice_usage = config_container[ZONE_0]->GetpreCICE_Usage();
if (precice_usage) {
  precice = new Precice(config_container[ZONE_0]->
 GetpreCICE_ConfigFileName(), rank, size, geometry_container[
 ZONE_0], solver_container[ZONE_0], config_container,
 grid_movement[ZONE_0]);
  dt = new double(config_container[ZONE_0]->GetDelta_UnstTimeND
 ());
  max_precice_dt = new double(precice->initialize());
}
```

Listing B.1: Instantiate *Precice* into the while-loop.

Then, preCICE needs to be able to shut down SU2 when the FSI simulation should be ended: at the beginning of Listing B.2 the number of iterations required is checked and if `isCouplingOngoing()` returns false the simulation is ended (line 2). Moreover, if the selected

coupling method is implicit, the usual checkpointing procedure starts (line 5). If at the end of the sub-iterations the convergence criterion is not met, the solver has to be set to restart the time-step.
Therefore, at the beginning of each iteration the current state has to be saved (line 6), in case it should be reloaded (line 29). If the convergence is reached, the simulation can proceed and the adapter begins the advancing phase (lines 9-16). `advance()` uses the current solver timestep size as input and returns the new maximum limit for the next time instance, as it is explained in Section 3.3.1.

```
1  //*--- Run the problem until the number of time iterations
    required is reached. ---*/
2  while ( ( TimeIter < config_container[ZONE_0]->GetnTime_Iter()
    && precice_usage && precice->isCouplingOngoing() ) || (
    TimeIter < config_container[ZONE_0]->GetnTime_Iter() && !
    precice_usage ) ) {
3
4    //preCICE implicit coupling: saveOldState()
5    if(precice_usage && precice->isActionRequired(precice->
    getCowic())){
6      precice->saveOldState(&StopCalc, dt);
7    }
8
9    //preCICE - set minimal time step size as new time step size
    in SU2
10   if(precice_usage){
11     dt = min(max_precice_dt,dt);
12     config_container[ZONE_0]->SetDelta_UnstTimeND(*dt);
13   //preCICE - Advancing
14   if(precice_usage){
15     *max_precice_dt = precice->advance(*dt);
16   }
17
18   //preCICE implicit coupling: reloadOldState()
19   bool suppress_output_by_preCICE = false;
20   bool initialTimeStep = false;
21
22   if(precice_usage && precice->isActionRequired(precice->
    getCoric())){
23
24     if (TimeIter == config_container[ZONE_0]->GetRestart_Iter()
     || TimeIter == 0) {
25       initialTimeStep = true;
26     }
27
28     //Stay at the same iteration number if preCICE is not
    converged and reload to the state before the current iteration
29     precice->reloadOldState(&StopCalc, dt, initialTimeStep);
30     suppress_output_by_preCICE = true;
```

```
      TimeIter--;
    }

    /*--- Output the solution in files. ---*/
    if (!suppress_output_by_preCICE){
      Output(TimeIter);
    }

    }
```

Listing B.2: Main while-loop in CSinglezoneDriver.cpp

The finalization of preCICE is done in the `CDriver.cpp`: communications are closed along with the deallocation of used memory (Listing B.3).

```
//preCICE - Finalize
  if(precice_usage){
    precice->finalize();
    if (dt != NULL) {
      delete dt;
    }
    if (max_precice_dt != NULL) {
      delete max_precice_dt;
    }
    if (precice != NULL) {
      delete precice;
    }
  }
```

Listing B.3: Finalization in CDriver.cpp

B.3 *Precice* Class

The actual adapter class for coupling SU2 with preCICE for FSI is `precice.cpp` (along with its header `precice.hpp`). Here, all the methods and variables needed by preCICE to perform the coupling are declared and used, in a similar way to the MBDyn adapter (Appendix A). The adapter is also connected to preCICE library through the instantiation of a *SolverInterface* object.
The adapter has to take care of

- force calculation at the interface,
- managing parallelization of SU2,
- converting data from SU2 representation to preCICE and viceversa,
- deform the mesh,
- all the steps that concern the implicit coupling (checkpointing, saving and loading states).

The first step is the initialization, done by the `initialize()` method. A list of functions is performed inside it:

- `CheckDimensionalConsistency()` checks whether SU2 and preCICE have the same dimensions
- `GetPreciceMeshID()` gets the correct mesh name ID
- `SetnLocalPreciceMarkers()` sets the number of preCICE markers
- `CheckWorkingProcess()` checks if the process is still ongoing
- `SetMarkerMapping()` handles the storing of marker values in a data array
- `SetMeshVertices()` sets the mesh vertices used for the data exchange
- `SetTimeStep()` sets the preCICE time-step

The adapter also has to compute forces and this is done inside the `ComputeForces()` function. Assuming a general 3D viscous case, the force vector is composed by pressure forces and friction forces:

$$f_i = -(p_{tot} - p_{stat})n_i A + \tau_{ij} n_j A \quad \forall i = 1,2,3, \quad with \tag{B.11}$$

$$\tau_{ij} = \mu \left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i} \right) - \frac{2}{3}\mu \frac{\partial v_k}{\partial x_k}\delta_{ij} \quad \forall i,j = 1,2,3 \tag{B.12}$$

where $\boldsymbol{f}$ is the force vector, p_{tot} is the total pressure, p_{stat} is the static one, $\boldsymbol{n}$ is the outward normal unit vector and $\boldsymbol{\tau}$ is the viscous stress tensor (cfr. Eq. 2.4).
After the forces evaluation and their translation into preCICE language, the `solverInterface` class is called and the forces are written with `writeBlockVectorData`, as fofr the MBDyn adapter. The same goes for the displacements, which are retrieved using `readBlockVectorData` and translated into SU2 language.
Then, the actual loop can start and the common preCICE functions are called:

- `isCouplingOngoing()` checks if the coupling is still on, `finalize()` is directly called
- `isActionRequired()` is used to read or write checkpoints (implicit coupling)
- `getCoric` and `getCowic` are used to differentiate read or write checkpoints
- `saveOldState()` and `reloadOldState()` are called after the checkpoint, if implicit coupling is chosen. The first is used to save the old state in the event that the convergence criterion wasn't met and the iteration had to be rerun (no advancing of the time-step)
- `advance()` is called in order to restart the loop. Within this function, the forces computation and the displacements reading is done. While advancing, the adapter also need to prescribe the maximum allowed timestep and, if implicit coupling is chosen, preCICE also executes acceleration techniques.
- `finalize()` closes all communications, deallocate used memory and stops the simulation.

Appendix C

Appendix C: Setting up a Simulation

This Appendix will provide a short guide to the set up of a coupled simulation.
Firstly, let us take a look into the simulation folder (Figure C.1a). Here, the user should have a sub-folder for SU2, called *Fluid*, and one for MBDyn, called *structural*, where all the simulation files needed by each solver are. SU2 needs the mesh file and the configuration file `.cfg`. MBDyn needs the main file `.mbd` and its auxiliary files, as shown in the example of Figure C.1b. Then, the preCICE configuration file, namely `precice-config.xml` should be present, along with `config.json`, which is the MBDyn adapter configuration file.

```
coupled simulation/
├── Fluid/
├── structural/
├── precice-config.xml
└── config.json
```

(a) Simulation folder structure.

```
structural/
├── beam.elm
├── beam.nod
├── joint.elm
├── mbdynAdapter.py
├── mbdynInterface.py
├── refConfigNodes.in
├── run.py
└── wing.mbd
```

(b) MBDyn sub-folder.

Figure C.1: Preview of the simulation folder with an insight into the MBDyn files. In red, the MBDyn adapter files. Note that for each simulation, the MBDyn input files could differ.

In order for SU2 to communicate with preCICE, the beginning of the configuration file should look like Listing C.1. In the proposed example there are two surfaces which interact with the structure, namely *BLADE 1* and *BLADE 2*. An initial load ramping to ease the computation is set, for a total duration of 30 iterations.

```

% ------------- PRECICE PROBLEM DEFINITION ------------%

PRECICE_USAGE= YES
%
PRECICE_CONFIG_FILENAME= precice-config.xml
%
MARKER_PRECICE= ( BLADE_1, BLADE_2 )
```

```
%
PRECICE_VERBOSITYLEVEL_HIGH= NO
%
PRECICE_LOADRAMPING= YES
%
PRECICE_LOADRAMPING_DURATION= 30
```

Listing C.1: First flags of SU2 configuration file.

The user should, then, set up the `precice-config.xml` and the `config.json`. In the preCICE configuration file, which is shown in Section 3.3.5, the most relevant variables to set are:

- solver-interface dimensions
- name of the data vector that should be exchange. E.g., for one FSI interface it should be *"Forces0"* and *"DisplacementDeltas0"*
- type of mapping. See Section 3.3.3
- type of coupling scheme, along with the timestep and the related parameters. Note that the used timestep can be different for the two participants, hence the data transfer will occur at the largest timestep, while the other participant will sub-cycle. See Section 3.3.1.

The MBDyn adapter directly reads most of the parameters from the `config.json` file, as Listing C.2. The user should specify here:

- `mbdyn-timestep`, the value set as MBDyn time-step
- `exposed-nodes`, number of nodes exposed through the external structural force
- `dimensions`, 2D or 3D simulation
- `FSI-interfaces`, number of FSI interfaces
- `input-file-mbdyn`, name of the main MBDyn input file.

```
{
    "mbdyn-timestep": 0.001,
    "exposed-nodes": 10,
    "dimensions": 2,
    "FSI-interfaces": 1,
    "input-file-mbdyn": "structural/wing.mbd"
}
```

Listing C.2: config.json file.

The simulation can now be properly started. The two participants require to be called in two separate shell environments from the principal folder. For SU2, the standard command is used:

```
mpirun -n NP SU2_CFD Fluid/config_file_name.cfg
```

where mpirun is used to run a parallel simulation and `NP` represents the number of processors. The MBDyn adapter is called with:

```
python3 run.py
```

In order to do that, another execution file, called **`run.sh`** can be added to the principal folder, in order to simplify the execution of both participants. An example is given in Listing C.3

```
SU2_CONFIG=Fluid/config_SU2_coupled.cfg
mpirun -n 4 SU2_CFD ${SU2_CONFIG} > Fluid.log 2>&1 &

python3 structural/run.py > Solid.log 2>&1
```

Listing C.3: Optional script used to start the simulation.

Bibliography

[1] R. Schlinker and R. Amiet. Rotor-vortex interaction noise. 1983.

[2] Jean Donea, Antonio Huerta, J.-Ph. Ponthot, and A. Rodríguez-Ferran. *Arbitrary Lagrangian–Eulerian Methods*, chapter 14. American Cancer Society, 2004.

[3] Masarati Pierangelo. *Dynamics and Control of the Flexible Aircraft.* Lectures from DCFA course, 2020.

[4] David Gregory Tipton, Mark Allen Christon, and Marc Stuart Ingber. A super-sampled projection method for level-set construction in fluid-solid interaction problems. *International Journal for Numerical Methods in Fluids*, 2009.

[5] Alessandro Cocco, Alberto Savino, Davide Montagnani, Matteo Tugnoli, Federico Guerroni, Michele Palazzi, Andrea Zanoni, Alex Zanotti, and Vincenzo Muscarello. Simulation of tiltrotor maneuvers by a coupled multibody-mid fidelity aerodynamic solver. In *46th European Rotorcraft Forum (ERF 2020)*, 09 2020.

[6] Alexander Rusch. Extending su2 to fluid-structure interaction via precice. *Bachelor's thesis, Technische Universität München*, 2016.

[7] Berend G. van der Wall. 2nd hhc aeroacoustic rotor test (hart ii) - part i: Test documentation -. Technical report, 2003. LIDO-Berichtsjahr=2003,.

[8] Ted Belytschko. Methods and programs for analysis of fluid-structure systems. *Nuclear Engineering and Design*, 42(1):41–52, 1977.

[9] Ted Belytschko. Fluid-structure interaction. *Computers & Structures*, 12(4):459–469, 1980.

[10] K.J. Bathe and W.F. Hahn. On transient analysis of fluid-structure systems. *Computers & Structures*, 10(1):383–391, 1979.

[11] J. Donea, S. Giuliani, and J. Halleux. An arbitrary lagrangian-eulerian finite element method for transient dynamic fluid-structure interactions. *Computer Methods in Applied Mechanics and Engineering*, 33:689–723, 1982.

[12] Thomas P. Combes, Arif S. Malik, Götz Bramesfeld, and Mark W. McQuilling. Efficient fluid-structure interaction method for conceptual design of flexible, fixed-wing micro-air-vehicle wings. *AIAA Journal*, 53(6):1442–1454, 2015.

[13] *Coupled Analysis of an Offshore Monopile Wind Turbine Subjected to Wind and Waves*, volume All Days of *International Ocean and Polar Engineering Conference*, 10 2020. ISOPE-I-20-1161.

[14] Yuanchuan Liu, Qing Xiao, and Atilla Incecik. A coupled cfd/multibody dynamics analysis tool for offshore wind turbines with aeroelastic blades. In *ASME 2017 36th International Conference on Ocean, Offshore and Arctic Engineering*, page V010T09A038, 06 2017.

[15] Chee Tung, Francis X. Caradonna, and Wayne R. Johnson. The prediction of transonic flows on an advancing rotor. *Journal of the American Helicopter Society*, 31(3):4–9, 1986.

[16] J. Bridgeman, R. Strawn, F. Caradonna, and Ching-Shung Chen. Advanced rotor computations with a corrected potential method. In *Proceedings of the 45th American Helicopter Society Annual Forum, Boston, MA*, 1989.

[17] O. Bauchau and J. Ahmad. *Advanced CFD and CSD methods for multidisciplinary applications in rotorcraft problems*, chapter Technical Papers. Pt. 2, pages 1441–1451. AIAA, NASA and ISSMO, 1996.

[18] Marilyn Jones Smith. *A fourth order Euler/Navier-Stokes prediction method for the aerodynamics and aeroelasticity of hovering rotor blades.* PhD thesis, Georgia Inst. of Tech., Atlanta, GA., January 1994.

[19] O.A. Bauchau, C.L. Bottasso, and Y.G. Nikishkov. Modeling rotorcraft dynamics with finite element multibody procedures. *Mathematical and Computer Modelling*, 33(10):1113–1137, 2001.

[20] T. Hablowetz. Advanced helicopter flight and aeroelastic simulation based on general purpose multibody code. In *AIAA Dynamics Specialists Conference Atlanta, GA, USA*, 2000.

[21] Björn Hübner, Elmar Walhorn, and Dieter Dinkler. A monolithic approach to fluid–structure interaction using space–time finite elements. *Computer Methods in Applied Mechanics and Engineering*, 193(23):2087–2104, 2004.

[22] J. Alonso and A. Jameson. Aiaa 94-0056 fully-implicit time-marching aeroelastic solutions. 2000.

[23] J. Ahmad and R. Biedron. Code-to-code comparison of cfd/csd simulation for a helicopter rotor in forward flight. In *2011 AIAA Applied Aerodynamics Conference, Honolulu, Hi*, 2011.

[24] Mark Potsdam, Hyeonsoo Yeo, and Wayne Johnson. Rotor airloads prediction using loose aerodynamic/structural coupling. *Journal of Aircraft*, 43(3):732–742, 2006.

[25] K. Pahlke and Berend G. van der Wall. Calculation of multibladed rotors in high-speed forward flight with weak fluid-structure-coupling. In *27th European Rotorcraft Forum, Moscow (Russia), 11-14 September 2001*, pages 27.1–27.11, 2001. LIDO-Berichtsjahr=2001,.

[26] Altmikus, Wagner, Beaumier, and Servera. A comparison: Weak versus strong modular coupling for trimmed aeroelastic rotor simulations. In *American Helicoper Society 58th Annual Forum, Montreal, Canada*, 2002.

[27] Chunhua Sheng, Jacob Ickes, Jingyu Wang, and Qiuying Zhao. *CFD/CSD Coupled Simulations for Helicopter Rotors in Forward and Maneuver Flights.*

[28] Berend G van der Wall and Casey L Burley. 2nd hhc aeroacoustic rotor test (hart ii)-part ii: representative results. 2005.

[29] Joon Lim, C. Tung, Y.H. Yu, Casey Burley, T. Brooks, D. Jr, Berend Wall, O. Schneider, Hugues Richard, M. Raffel, P. Beaumier, J. Bailly, Yves Delrieux, K. Pengel, and E. Mercker. Hart ii: Prediction of blade-vortex interaction loading. 01 2003.

[30] Joon Lim and Berend Wall. Investigation in the effect of a multiple trailer free wake model for descending flights. pages TechnicalSession:DynamicsII–, 01 2005.

[31] Hyeonsoo Yeo and Wayne Johnson. Assessment of comprehensive analysis calculation of airloads on helicopter rotors. *Journal of Aircraft - J AIRCRAFT*, 42:30, 01 2004.

[32] Byung-Young Min, Lakshmi Sankar, JVR Prasad, and Daniel Schrage. A physics-based investigation of gurney flaps for rotor vibration reduction. *65th Annual AHS Forum*, 1:139–149, 01 2009.

[33] Jeonghwan Sa, Younghyun You, Jae-Sang Park, Sung Jung, Soo Hyung Park, and Y.H. Yu. Assessment of cfd/csd coupled aeroelastic analysis solution for hart ii rotor incorporating fuselage effects. 1:736–748, 01 2011.

[34] J. W. Lim. An assessment of rotor dynamics correlation for descending flight using cfd/csd coupled analysis. *64th Annual AHS Forum*, 1:239–260, 2008.

[35] Marilyn Smith, Joon Lim, Berend Wall, James Baeder, Robert Biedron, D. Jr, Buvana Jayaraman, Sung Jung, and Byung-Young Min. An assessment of cfd/csd prediction state-of-the-art using the hart ii international workshop data. volume 1, pages 1–41, 05 2012.

[36] B. G. van der wall. A comprehensive rotary-wing data base for code validation: the hart ii international workshop. *The Aeronautical Journal*, 115(1164):91–102, 2011.

[37] John D. Anderson. *Fundamentals of aerodynamics*. McGraw-Hill, 5th edition, 2011.

[38] Pijush K. Kundu, Ira M. Cohen, and David R. Dowling. *Fluid Mechanics (Sixth Edition)*. Academic Press, Boston, sixth edition edition, 2016.

[39] L. E. MALVERN. *Introduction to the mechanics of a continuous medium*. Englewood Cliffs, N.J: Prentice-Hall, 1969.

[40] R. Sánchez, R. Palacios, T. Economon, H. Kline, J. Alonso, and F. Palacios. Towards a fluid-structure interaction solver for problems with large deformations within the open-source su2 suite. *Aiaa Journal*, 2016.

[41] M. Nazem, J.P. Carter, and D.W. Airey. Arbitrary lagrangian–eulerian method for dynamic analysis of geotechnical problems. *Computers and Geotechnics*, 36(4):549–557, 2009.

[42] R.W. Ogden. *Non-linear Elastic Deformations*. Dover Civil and Mechanical Engineering. Dover Publications, 1997.

[43] Pierangelo Masarati, Marco Morandini, and Paolo Mantegazza. An Efficient Formulation for General-Purpose Multibody/Multiphysics Analysis. *Journal of Computational and Nonlinear Dynamics*, 9(4), 07 2014. 041001.

[44] Björn Hübner, Elmar Walhorn, and Dieter Dinkler. A monolithic approach to fluid-structure interaction using space-time elements. *Computer Methods in Applied Mechanics and Engineering*, 193:2087–2104, 06 2004.

[45] C. Michler, S.J. Hulshoff, E.H. van Brummelen, and R. de Borst. A monolithic approach to fluid–structure interaction. *Computers & Fluids*, 33(5):839–848, 2004. Applied Mathematics for Industrial Flow Problems.

[46] Frederic J. Blom. A monolithical fluid-structure interaction algorithm applied to the piston problem. *Computer Methods in Applied Mechanics and Engineering*, 167(3):369–391, 1998.

[47] Serge Piperno, Charbel Farhat, and Bernard Larrouturou. Partitioned procedures for the transient solution of coupled aroelastic problems part i: Model problem, theory and two-dimensional application. *Computer Methods in Applied Mechanics and Engineering*, 124(1):79–112, 1995.

[48] Gaëlle Servera, Philippe Beaumier, and Michel Costes. A weak coupling method between the dynamics code host and the 3d unsteady euler code waves. *Aerospace Science and Technology*, 5(6):397–408, 2001.

[49] LONG CHEN, YIZHAO WU, and JIAN XIA. Aeroelastic analysis of rotor blades using cfd/csd coupling in hover mode. *Modern Physics Letters B*, 24(13):1307–1310, 2010.

[50] A Zaki, N Reveles, Marilyn Smith, and Olivier Bauchau. Using tightly-coupled cfd/csd simulation for rotorcraft stability analysis. *Annual Forum Proceedings - AHS International*, 4, 01 2010.

[51] T. Baker and P. Cavallo. Dynamic adaptation for deforming tetrahedral meshes. In *14th Computational Fluid Dynamics Conference*, 1999.

[52] Richard P Dwight. Robust mesh deformation using the linear elasticity equations. In *Computational fluid dynamics 2006*, pages 401–406. Springer, 2009.

[53] K. Stein, Tayfun Tezduyar, and R. Benney. Mesh moving techniques for fluid-structure interactions with large displacements. *Journal of Applied Mechanics*, 70:58–63, 01 2003.

[54] Luca Cavagna, Giuseppe Quaranta, and Paolo Mantegazza. Application of navier–stokes simulations for aeroelastic stability assessment in transonic regime. *Computers & Structures*, 85(11):818–832, 2007. Fourth MIT Conference on Computational Fluid and Solid Mechanics.

[55] Hariyo Pratomo, Fandi Suprianto, and Teng Sutrisno. Preliminary study on mesh stiffness models for fluid-structure interaction problems. *E3S Web of Conferences*, 130:01014, 01 2019.

[56] A. de Boer, M.S. van der Schoot, and H. Bijl. Mesh deformation based on radial basis function interpolation. *Computers & Structures*, 85(11):784–795, 2007. Fourth MIT Conference on Computational Fluid and Solid Mechanics.

[57] T. C. S. Rendall and C. B. Allen. Parallel efficient mesh motion using radial basis functions with application to multi-bladed rotors. *International Journal for Numerical Methods in Engineering*, 81(1):89–105, 2010.

[58] Myles Morelli, Tommaso Bellosta, and Alberto Guardone. Efficient radial basis function mesh deformation methods for aircraft icing. *Journal of Computational and Applied Mathematics*, 392:113492, 2021.

[59] Marco Biancolini. *FSI Workflow Using Advanced RBF Mesh Morphing*, pages 225–256. Springer, Cham, 01 2017.

[60] M. Lombardi, Nicola Parolini, and Alfio Quarteroni. Radial basis functions for inter-grid interpolation and mesh motion in fsi problems. *Computer Methods in Applied Mechanics and Engineering*, 256:117–131, 04 2013.

[61] Gang Wang, Haris Hameed Mian, Zheng-Yin Ye, and Jen-Der Lee. Improved point selection method for hybrid-unstructured mesh deformation using radial basis functions. *AIAA Journal*, 53(4):1016–1025, 2015.

[62] Liang Xie and H. Liu. Efficient mesh motion using radial basis functions with volume grid points reduction algorithm. *J. Comput. Phys.*, 348:401–415, 2017.

[63] Myles Morelli, Tommaso Bellosta, and Alberto Guardone. Development and preliminary assessment of the open-source cfd toolkit su2 for rotorcraft flows. *Journal of Computational and Applied Mathematics*, 389:113340, 2021.

[64] Rolland L. Hardy. Multiquadric equations of topography and other irregular surfaces. *Journal of Geophysical Research (1896-1977)*, 76(8):1905–1915, 1971.

[65] M. D. Buhmann. Radial basis functions. *Acta Numerica*, 9:1–38, 2000.

[66] Holger Wendland. *Scattered Data Approximation.* Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2004.

[67] Thomas D Economon, Francisco Palacios, Sean R Copeland, Trent W Lukaczyk, and Juan J Alonso. Su2: An open-source suite for multiphysics simulation and design. *Aiaa Journal*, 54(3):828–846, 2015.

[68] Philippe Spalart and Steven Allmaras. A one-equation turbulence model for aerodynamic flows. *AIAA*, 439, 01 1992.

[69] Florianr Menter. Zonal two equation kw turbulence models for aerodynamic flows. In *23rd fluid dynamics, plasmadynamics, and lasers conference*, page 2906, 1993.

[70] Bram van Leer. Towards the ultimate conservative difference scheme. v. a second-order sequel to godunov's method. *Journal of Computational Physics*, 32(1):101–136, 1979.

[71] Beckett Yx Zhou, Myles Morelli, Nicolas R. Gauger, and Alberto Guardone. *Simulation and Sensitivity Analysis of a Wing-Tip Mounted Propeller Configuration from the Workshop for Integrated Propeller Prediction (WIPP).*

[72] W. Schiehlen. Multibody system dynamics: Roots and perspectives. *Multibody System Dynamics*, 1:149–188, 1997.

[73] Gian Ghiringhelli, Paolo Mantegazza, and Pierangelo Masarati. Multibody implementation of finite volume c beams. *Aiaa Journal - AIAA J*, 38:131–138, 01 2000.

[74] Giuseppe Quaranta, Giampiero Bindolino, Pierangelo Masarati, and Paolo Mantegazza. Toward a computational framework for rotorcraft multi-physics analysis: Adding computational aerodynamics to multibody rotor models. *30th European Rotorcraft Forum*, 2005, 04 2012.

[75] Dale Pitt and David Peters. Theoretical prediction of dynamic-in ow derivatives. *Vertica*, 5, 01 1981.

[76] J. Degroote, P. Bruggeman, R. Haelterman, and J. Vierendeels. Stability of a coupling technique for partitioned solvers in fsi applications. *Computers & Structures*, 86:2224–2234, 2008.

[77] J. Degroote, K. Bathe, and J. Vierendeels. Performance of a new partitioned procedure versus a monolithic procedure in fluid-structure interaction. *Computers & Structures*, 87:793–801, 2009.

[78] Bernhard Gatzhammer. Efficient and flexible partitioned simulation of fluid-structure interactions. *PhD thesis, Technische Universität München*, 2015.

[79] S. Étienne and D. Pelletier. A general approach to sensitivity analysis of fluid–structure interactions. *Journal of Fluids and Structures*, 21(2):169–186, 2005.

[80] Hans-Joachim Bungartz, Florian Lindner, Bernhard Gatzhammer, Miriam Mehl, Klaudius Scheufele, Alexander Shukaev, and Benjamin Uekermann. precice – a fully parallel library for multi-physics surface coupling. *Computers & Fluids*, 141:250–258, 2016. Advances in Fluid-Structure Interaction.

[81] Miriam Mehl, Benjamin Uekermann, Hester Bijl, David Blom, Bernhard Gatzhammer, and Alexander van Zuijlen. Parallel coupling numerics for partitioned fluid–structure interaction simulations. *Computers & Mathematics with Applications*, 71(4):869–891, 2016.

[82] E Ramm and W Wall. Fluid-structure interaction based upon a stabilized (ale) finite element method. In *4th World Congress on Computational Mechanics: New Trends and Applications, CIMNE, Barcelona*, pages 1–20, 1998.

[83] Christophe Geuzaine and Jean-François Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.

[84] Wulf Dettmer and D. Perić. A fully implicit computational strategy for strongly coupled fluid–solid interaction. *Archives of Computational Methods in Engineering*, 14:205–247, 09 2007.

[85] Christophe Kassiotis, Adnan Ibrahimbegovic, Rainer Niekamp, and Hermann Matthies. Nonlinear fluid-structure interaction problem. part i: Implicit partitioned algorithm, nonlinear stability proof and validation examples. *Computational Mechanics*, 47:305–323, 03 2011.

[86] Bradley Froehle and Per-Olof Persson. A high-order discontinuous galerkin method for fluid–structure interaction with efficient implicit–explicit time stepping. *Journal of Computational Physics*, 272:455–470, 09 2014.

[87] Hermann Matthies and Jan Steindorf. Partitioned strong coupling algorithms for fluid-structure interaction. *Computers & Structures*, 81:805–812, 05 2003.

[88] Clare Wood, Antonio Gil, O. Hassan, and Javier Bonet. A partitioned coupling approach for dynamic fluid–structure interaction with applications to biological membranes. *International Journal for Numerical Methods in Fluids*, 57:555 – 581, 06 2008.

[89] Habchi C., Russeil S., Bougeard D., Harion J.-L.and Lemenand T., Ghanem A., Valle D., and Peerhossaini H. Partitioned solver for strongly coupled fluid-structure interaction. *Computers and Fluids*, 71(11 - 12):793 – 801, 2013.

[90] M Folkersma. Coupling openfoam and mbdyn with precice coupling tool. *In Proceedings ofCFD with OpenSource Software*, 2018.

[91] Niklas Johannes Adam. Computational simulation of fluid-structure interaction of soft kites. *Master's Thesis, Universität Stuttgart*, 2018.

[92] Matteo Tugnoli, Davide Montagnani, Federico Fonte, Alex Zanotti, Monica Syal, and Giovanni Droandi. Mid-fidelity analysis of unsteady interactional aerodynamics of complex vtol configurations. In *45th European Rotorcraft Forum*, 09 2019.