DTU

Master thesis presentation

Implementation of an Open-source Tool for Aeroelastic Simulations of Horizontal Axis Wind Turbines

Index:

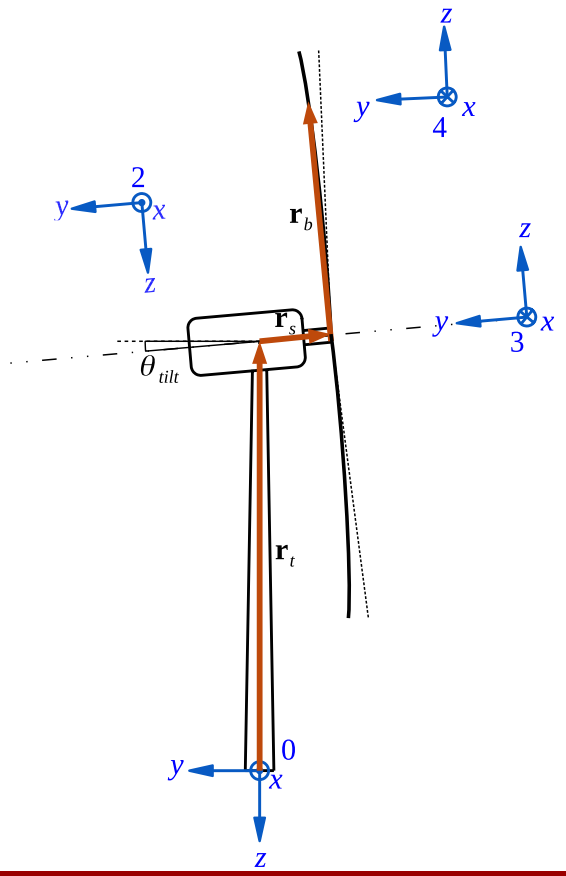
- Objectives
- Structural module
- Aerodynamic module
- Implementation
- Validation
- Questions

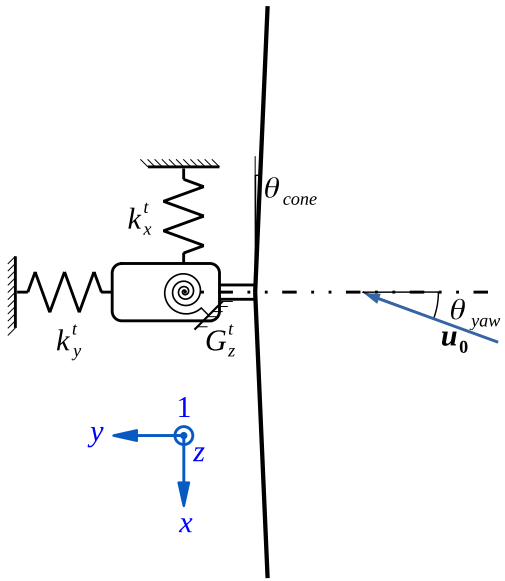
Objectives:

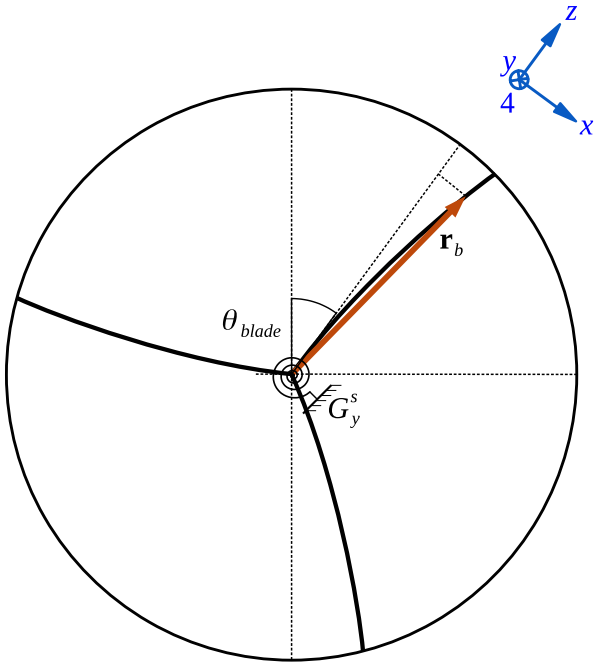
- Implement a **simple** open-source **aeroelastic tool**, (HarPy), for the simulation of **horizontal axis wind turbines** under **steady and turbulent wind** inflow conditions in the **time domain**.
- **Complete enough** to achieve reasonably **accurate results**.
- **Simple enough** to allow a **straightforward connection between** the basic concepts, **problem physics**, **model** assumptions, key **equations**, the **algorithm** and the implemented **code**.
- The code must be **open-source** and **free** so the use of the code would not be a barrier to the use, **customisation** and **expandability** of the code.

Wind turbine reference kinematics:

- The complete turbine dynamics is obtained using 1 inertial and 4 moving reference of frames.

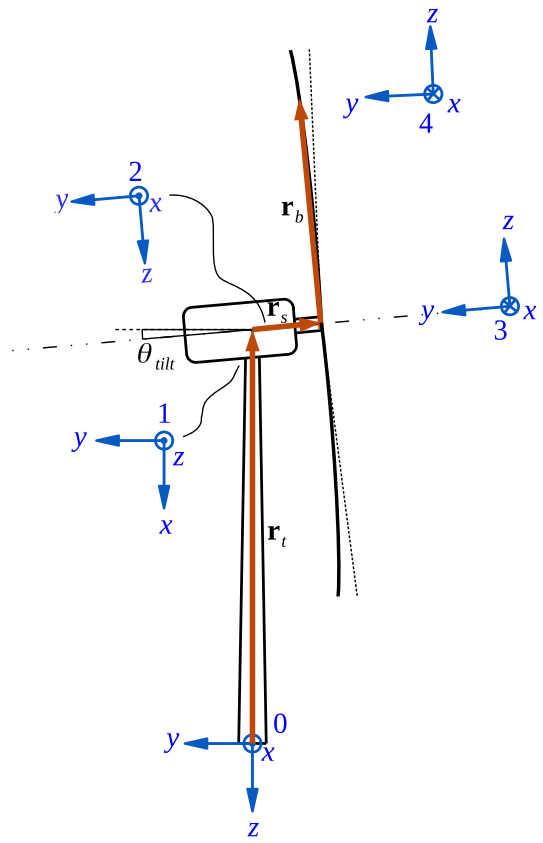






Wind turbine reference kinematics:

- The complete turbine dynamics is obtained using 1 inertial and 4 moving reference of frames.



Reference frames

- **I_0 : Inertial**
- **I_1 : Tower-top**
- **I_2 : Shaft bearing**
- **I_3 : Shaft tip**
- **I_4 : Blade root**

Transformation matrices:

$$\mathbf{A}_{01} = \begin{bmatrix} \cos(\theta_{t,z}(t)) & \sin(\theta_{t,z}(t)) & 0 \\ -\sin(\theta_{t,z}(t)) & \cos(\theta_{t,z}(t)) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{A}_{12} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_{tilt}) & \sin(\theta_{tilt}) \\ 0 & -\sin(\theta_{tilt}) & \cos(\theta_{tilt}) \end{bmatrix}$$

$$\mathbf{A}_{23} = \begin{bmatrix} \cos(\eta + \theta(t) + \theta_{s,y}(t)) & 0 & -\sin(\eta + \theta(t) + \theta_{s,y}(t)) \\ 0 & 1 & 0 \\ \sin(\eta + \theta(t) + \theta_{s,y}(t)) & 0 & \cos(\eta + \theta(t) + \theta_{s,y}(t)) \end{bmatrix}$$

$$\mathbf{A}_{34} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_{cone}) & \sin(\theta_{cone}) \\ 0 & -\sin(\theta_{cone}) & \cos(\theta_{cone}) \end{bmatrix}$$

Blade kinematics:

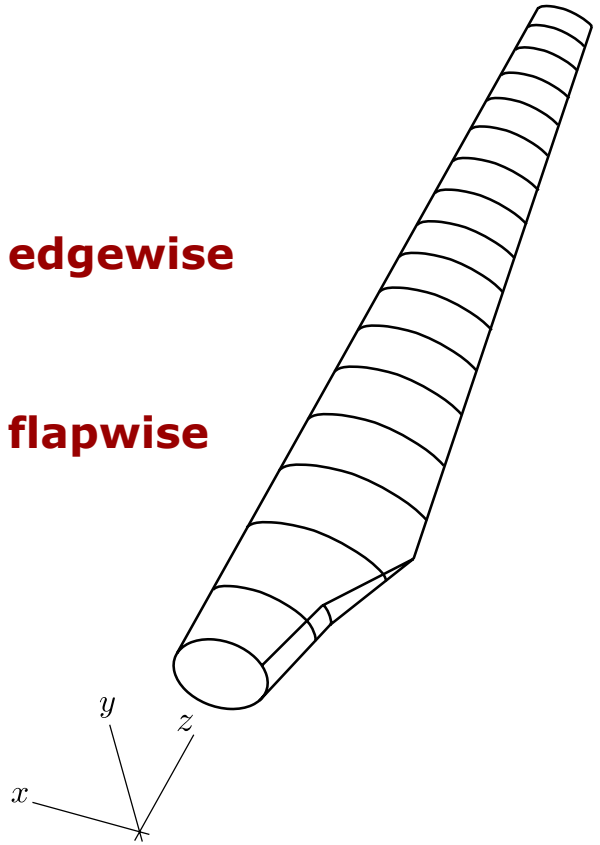
- The turbine **blades displacement** is described using a **linear combination of its mode shapes**.

$$ {}_4\mathbf{r}_b(t,z) = \begin{bmatrix} x_b(t,z) \\ y_b(t,z) \\ z \end{bmatrix} $$

$$ x_b(t,z) = x_b(t_0,z) + \sum_{i=0}^{n} \phi_{i,x}(z)\, q_i(t) \quad \textbf{edgewise} $$

$$ y_b(t,z) = y_b(t_0,z) + \sum_{i=0}^{n} \phi_{i,y}(z)\, q_i(t) \quad \textbf{flapwise} $$

Stiff radially

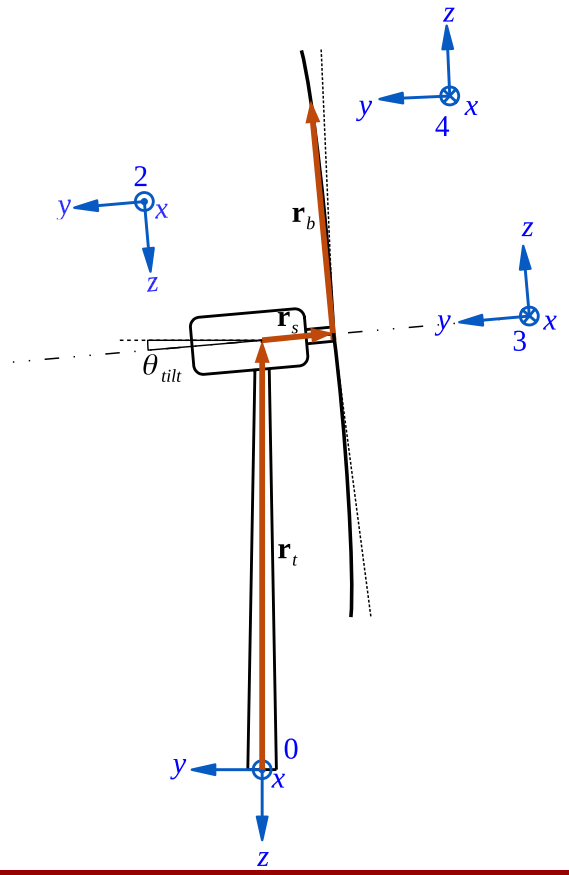


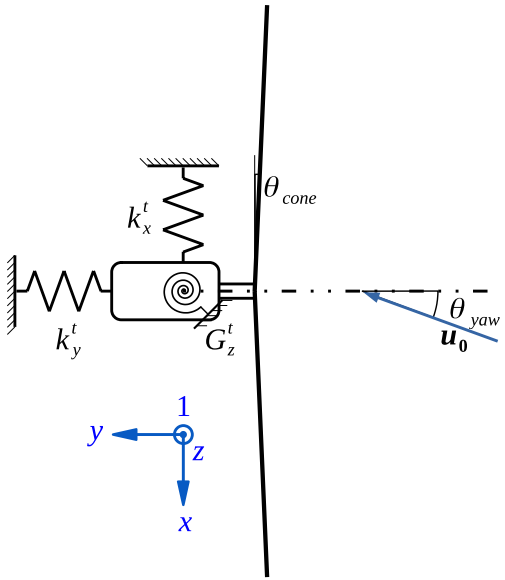
Nacelle and shaft kinematics:

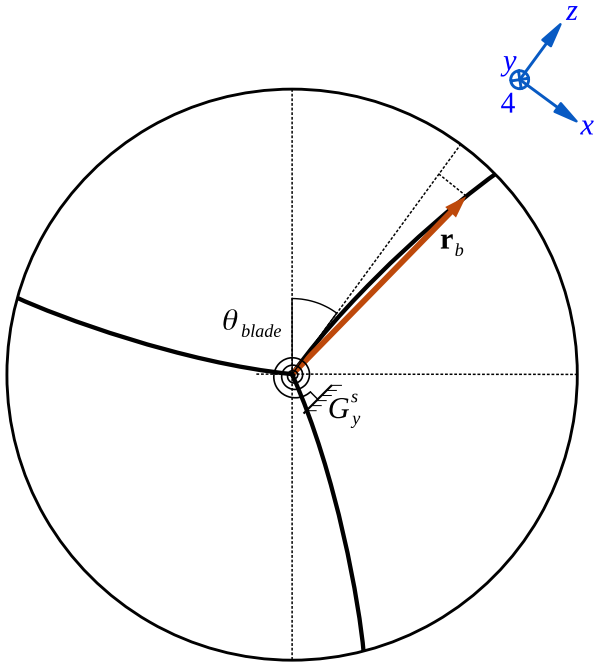
- The **tower top** and **shaft tip** displacements are modelled using **linear and angular stiffness**.

$$ {}_0\mathbf{r}_t = \begin{bmatrix} u_{t,x}(t) \\ u_{t,x}(t) \\ -h_t \end{bmatrix} $$

$$ {}_3\mathbf{r}_s = \begin{bmatrix} 0 \\ -s_l \\ 0 \end{bmatrix} $$







Wind turbine reference kinematics:

- The **position** and absolute **velocity** of the **blades sections** are then obtained.

$$ {}_0\mathbf{r}\left(t\right) = {}_0\mathbf{r}_t + \mathbf{A}_{01}^T\left(t\right) \cdot \mathbf{A}_{12}^T \cdot \mathbf{A}_{23}^T\left(t\right) \cdot \left[{}_3\mathbf{r}_s + \mathbf{A}_{34}^T \cdot {}_4\mathbf{r}_b\left(t\right)\right] $$

$$ {}_4\mathbf{v}\left(t\right) = \mathbf{A}_{04} \cdot \frac{d}{dt}\left({}_0\mathbf{r}_t\right) + \mathbf{A}_{34} \cdot \left[{}_3\mathbf{\Omega}_{01}\left(t\right) + {}_3\mathbf{\Omega}_{23}\left(t\right)\right] \cdot {}_3\mathbf{r}_s + \left[{}_4\mathbf{\Omega}_{01}\left(t\right) + {}_4\mathbf{\Omega}_{23}\left(t\right)\right] \cdot {}_4\mathbf{r}_b\left(t\right) + \frac{d}{dt}\left({}_4\mathbf{r}_b\right) $$

$$ {}_1\mathbf{\Omega}_{01} = \mathbf{A}_{01} \cdot \frac{d}{dt}\left(\mathbf{A}_{01}^T\right) $$

Energy equations:

- To derive the **equations of motion** using the **Lagrange method** is necessary to derive the **kinetic** and **potential** energy of the wind turbine and the **work done by non-conservative forces**.

$$E_{kin} = \frac{1}{2}\left(m_n + m_h\right)\left[\dot{u}_{t,x}^2\left(t\right) + \dot{u}_{t,x}^2\left(t\right)\right] + \frac{1}{2} I_{t,z} \dot{\theta}_{t,z}^2\left(t\right) + \frac{1}{2}\sum_{b=0}^{n_b}\int_{z_0}^{z_R} m\left(z\right) v_b^2\left(t,z\right) dz$$

$$E_{pot} = \frac{1}{2}\left[k_{t,x} u_{t,x}^2\left(t\right) + k_{t,y} u_{t,y}^2\left(t\right) + g_{t,z}\theta_{t,z}^2\left(t\right)\right] + \frac{1}{2}\left[g_{s,y}\theta_s^2\right] - \sum_{b=0}^{n_b} g \int_{z_0}^{z_R} m\left(z\right) {}_0 r_{b,z}\left(t,z\right) dz +$$
$$+ \sum_{b=0}^{n_b}\sum_{i=0}^{n_m} \omega_i^2 \int_{z_0}^{z_R} m\left(z\right)\left[\phi_{i,x}^2\left(z\right) + \phi_{i,y}^2\left(z\right)\right] q_{3b+i}\left(t\right) dz$$

$$\mathcal{W}^{(nc)} = \sum_{b=0}^{n_b}\int_{z_0}^{z_R} {}_4\vec{f}\left(z,t\right) \cdot {}_4\vec{r}\left(z,t\right) dz$$

Lagrangian mechanics:

- In Lagrangian mechanics, the **equations of motion** of a system are derived by solving the **Lagrange equations**.
- The Lagrange method often offer a **more** straightforward and **convenient** method **to** implement on **symbolic computational tools**.

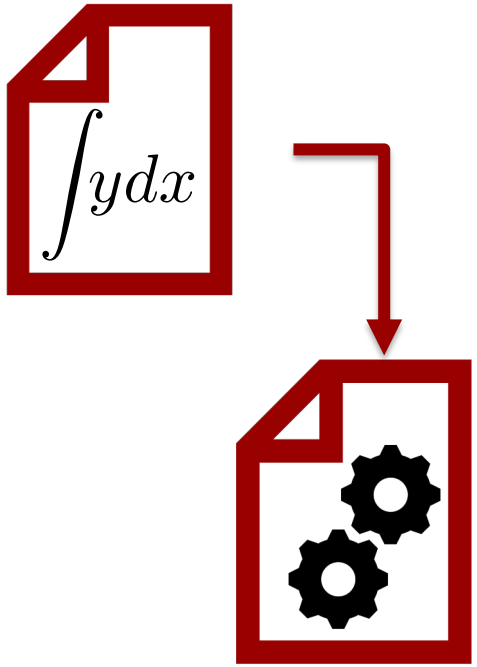
$$\mathbf{E}\left[j\right] = \frac{d}{dt}\left(\frac{E_{kin}}{\dot{q}_j}\right) - \frac{E_{kin}}{q_j} + \frac{E_{pot}}{q_j} - \frac{\mathcal{W}^{(nc)}}{q_j} = \mathbf{0}$$

$$\mathcal{M} \cdot \ddot{\mathbf{q}} = \mathcal{F}$$

$$\mathbf{M} \cdot \ddot{\mathbf{q}} + \mathbf{C} \cdot \dot{\mathbf{q}} + \mathbf{K} \cdot \mathbf{q} = \mathbf{F}$$

Symbolic implementation:

- The **symbolic derivation** was carried out in Python using the symbolic manipulation package Sympy.
- A code generator was adapted to **translate** the symbolic equations **into numerical code** and generate the structural module code automatically.

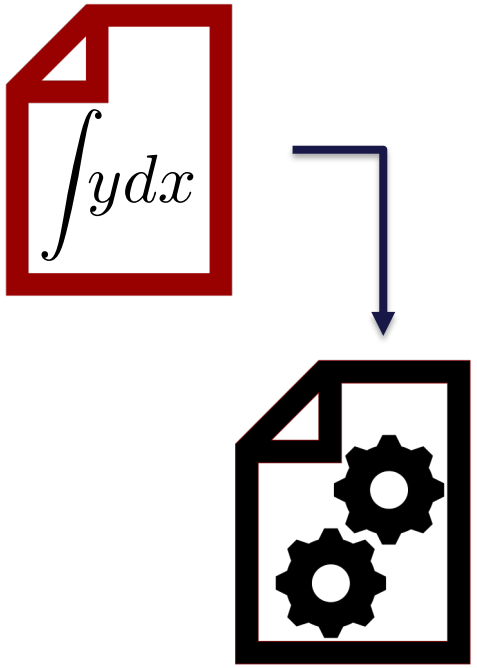


$$\int_0^R m\left(z\right) \phi_{[x,0]}^2\left(z\right) dz$$

```
numpy.trapz(self.m*self.phi_x[0,:]**2,self.z)
```

Structural module

Symbolic derivation:




```
r_4_b2 = r_4.xreplace(dict_b2) # blade 2, reference frame 4
```

2.12 Absolute blade velocity vector in blade frame (frame 4)

The absolute velocity is obtained taking the derivative of the position equation. However, it is often more convenient to express the velocity in the local referential frame, which is achieved simply multiplying the terms in the velocity vector equation by the transformation tensors.

$$ {}_4\mathbf{v}(t) = \mathbf{A}_{04} \cdot \frac{d}{dt}({}_0\mathbf{r}_t) + \mathbf{A}_{34} \cdot [{}_3\mathbf{\Omega}_{01}(t) + {}_3\mathbf{\Omega}_{23}(t)] \cdot {}_3\mathbf{r}_s + [{}_4\mathbf{\Omega}_{01}(t) + {}_4\mathbf{\Omega}_{23}(t)] \cdot {}_4\mathbf{r}_b(t) + \frac{d}{dt}({}_4\mathbf{r}_b) $$

The absolute velocity vector is generally more compact when written in the object frame of reference. The velocity equation derived does not require to be simplified. The kinetic energy is an scalar and does not depend of the frame of reference in which the velocity is described, but the time used to simplify the kinetic energy equation is greatly affected by it.

[17]:
```
v_4 = A_04*r_t.diff(t) + A_34*(Omega_01_3 + Omega_23_3)*r_s + (Omega_01_4 +␣
↪Omega_23_4)*r_b + r_b.diff(t)
v_4 = v_4.expand()

# Printing the equations
lhs_print('{}_{_4}\mathbf{v}', v_4)
```

$$ {}_4\mathbf{v} = \begin{bmatrix} z\frac{d}{dt}\theta(t) + \phi_{0x}(z)\frac{d}{dt}\mathrm{qb}_0(t) + \phi_{1x}(z)\frac{d}{dt}\mathrm{qb}_1(t) + \phi_{2x}(z)\frac{d}{dt}\mathrm{qb}_2(t) + \cos(\eta+\theta(t))\frac{d}{dt}\mathrm{q}[9](t) \\ \phi_{0y}(z)\frac{d}{dt}\mathrm{qb}_0(t) + \phi_{1y}(z)\frac{d}{dt}\mathrm{qb}_1(t) + \phi_{2y}(z)\frac{d}{dt}\mathrm{qb}_2(t) + \frac{d}{dt}\mathrm{q}[10](t) \\ -\phi_{0x}(z)\,\mathrm{qb}_0(t)\frac{d}{dt}\theta(t) - \phi_{1x}(z)\,\mathrm{qb}_1(t)\frac{d}{dt}\theta(t) - \phi_{2x}(z)\,\mathrm{qb}_2(t)\frac{d}{dt}\theta(t) - x(z)\frac{d}{dt}\theta(t) + \sin(\eta+\theta(t))\frac{d}{dt}\mathrm{q}[9](t) \end{bmatrix} $$

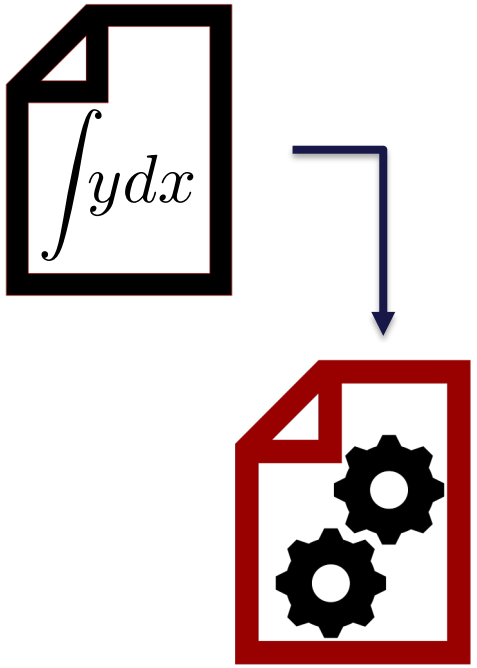
3 Kinectic energy

It is always possible to derive the systems equations of motion from the direct application of Newton's laws of motion. However, the Lagrange or energy method often offer a more straightforward and automatic method to implement with the aid of symbolic computational tools. In addition to simplifying the process of obtaining the equations of motion, the use of computational math packages is especially convenient for automatic code generation as will be show later. In order to derive the equations of motion using the Lagrange method is first necessary to derive the kinetic and potential energy of the wind turbine and the work done by non-conservative forces acting on it as a function of the generalised degrees of freedom.

The total kinetic energy associated with the motion of the wind turbine is given by the equation below. The first term represents the kinetic energy associated with the nacelle and hub translation velocity while the second and third terms represents the energy associated with its angular velocity. The fourth term is the integral of the energy associated with the translation of the blade segments along the three blades. It must be noted that the energy associated with the rotation of the blades segments around their respective centre of mass is small and is therefore disregarded.

Structural module

Numerical code output:



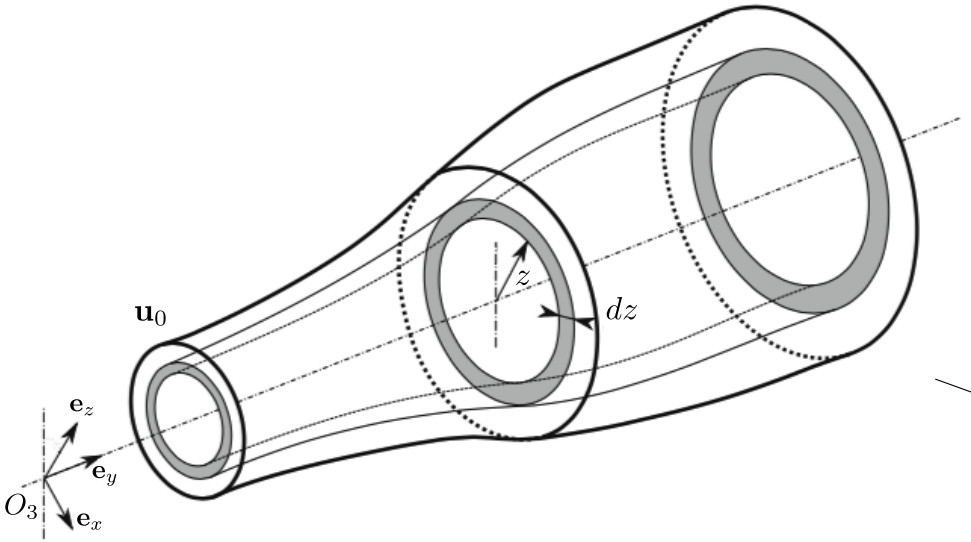
wind_turbine_structural.py

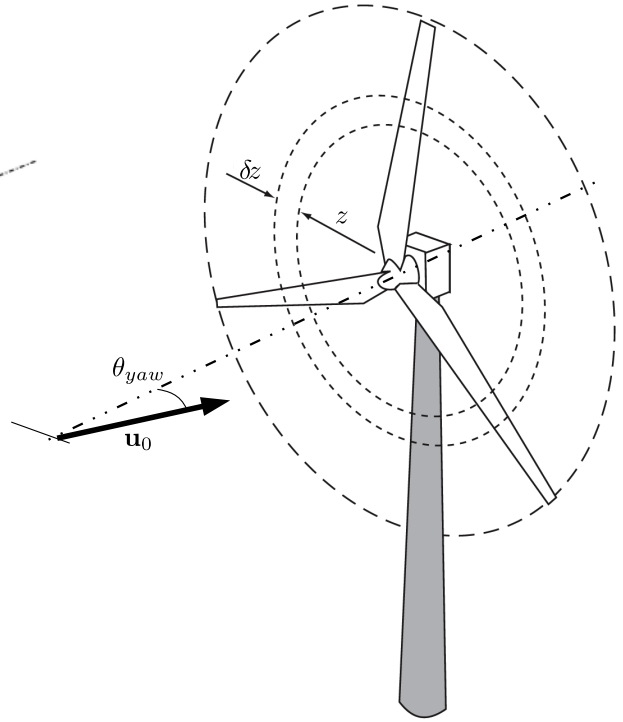
wind_turbine_structural.py 〉 initilialise

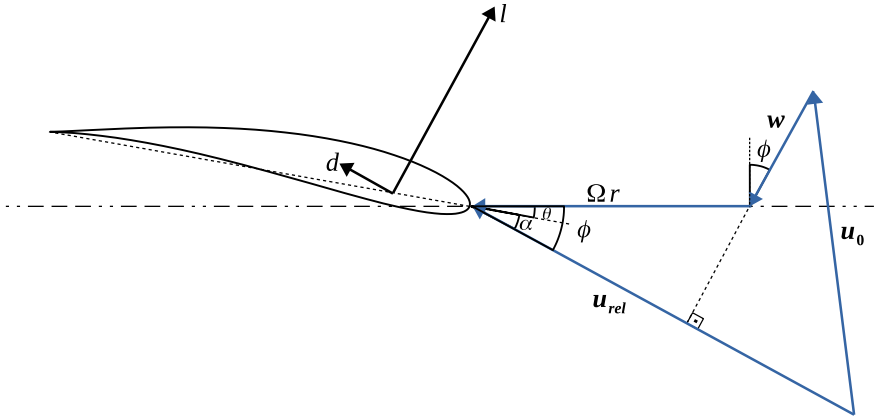
```python
import numpy

#%%

class WindTurbineStructural(object):
    '''
    Created automatically by Harpy symbolic module
    '''

    def __init__(self):
        '''
        This init method only declares the variables as place as place
        holders, i.e. without declaring the type or allocating memory.The
        instance creator(s) methods will be implemented via classmethods for
        allowing more flexibility for different imput methods.
        '''
        self.xint = numpy.zeros((108,), dtype=float)
        self.ytrig = numpy.zeros((12,), dtype=float)
        self.q = numpy.zeros((16,), dtype=float)
        self.q_dot = numpy.zeros((16,), dtype=float)
        self.q_ddot = numpy.zeros((16,), dtype=float)

    def reference_matrices(self):
        '''
        This method calculates the transformation tensors between reference
        frames and the angular velocity tensors associated with the movig
        reference frames.
        '''
        self.A_01 = numpy.zeros((3, 3), dtype=float)
        self.A_01[0, 0] = numpy.cos(self.q[14])
        self.A_01[0, 1] = numpy.sin(self.q[14])
        self.A_01[1, 0] = -numpy.sin(self.q[14])
        self.A_01[1, 1] = numpy.cos(self.q[14])
        self.A_01[2, 2] = 1
        #
        self.A_12 = numpy.zeros((3, 3), dtype=float)
        self.A_12[0, 0] = 1
        self.A_12[1, 1] = numpy.cos(self.tilt)
        self.A_12[1, 2] = numpy.sin(self.tilt)
        self.A_12[2, 1] = -numpy.sin(self.tilt)
        self.A_12[2, 2] = numpy.cos(self.tilt)
        #
```

Aerodynamic module:

- The wind turbine aerodynamics is modelled using the **unsteady Blade Element Momentum theory** under **yaw**.







Dynamic inflow:

- The **wind velocity induced by the rotor on flow** is calculated using the two filters **dynamic inflow model** proposed by Stig Øye .

$$ {}_3\mathbf{w}_{int} + \tau_1 \frac{d_3\mathbf{w}_{int}}{dt} = {}_3\mathbf{w}_{qs} + k\tau_1 \frac{{}_3\mathbf{w}_{qs}}{dt} $$

$$ {}_3\mathbf{w}_0 + \tau_2 \frac{d_3\mathbf{w}_0}{dt} = {}_3\mathbf{w}_{int} $$

$$ \tau_1 = \frac{1.1}{1 - 1.3a} \frac{R}{u_0} $$

$$ \tau_2 = \left[0.39 - 0.26 \left(\frac{r}{R}\right)^3\right] \tau_1 $$

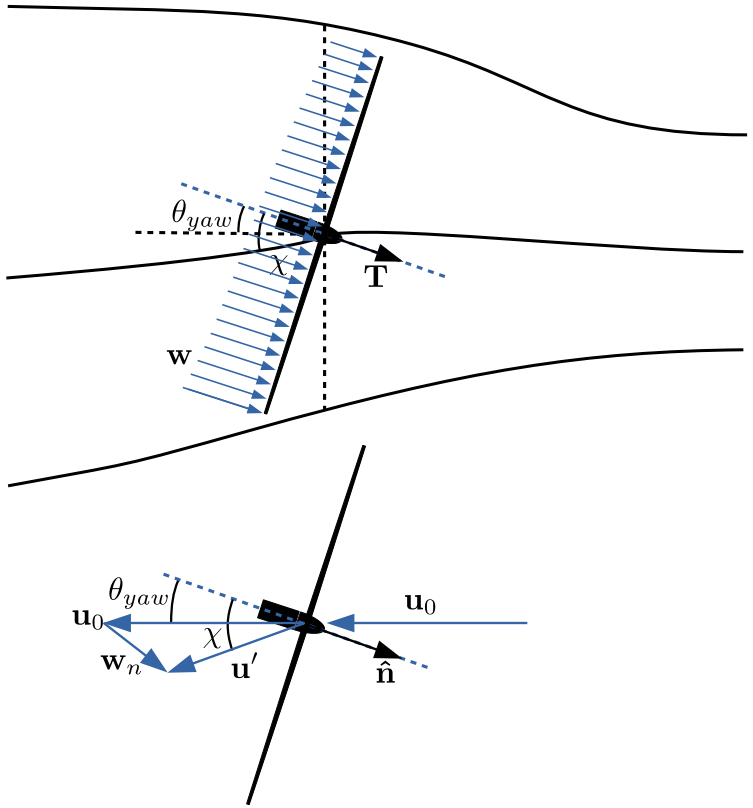
$$ {}_3w_{qs,x} = -\frac{B\ {}_3f_x}{4\pi\rho r F\left|\mathbf{u}_0 + f_g\mathbf{w}_n\right|} $$

$$ {}_3w_{qs,y} = -\frac{B\ {}_3f_y}{4\pi\rho r F\left|\mathbf{u}_0 + f_g\mathbf{w}_n\right|} $$

$$ {}_3w_{qs,z} = 0 $$

Yaw / Tilt correction:

- The **wake skew angle** effect on the induced wind velocity is calculated using the model proposed by Glauert as referenced in [18] and [35].



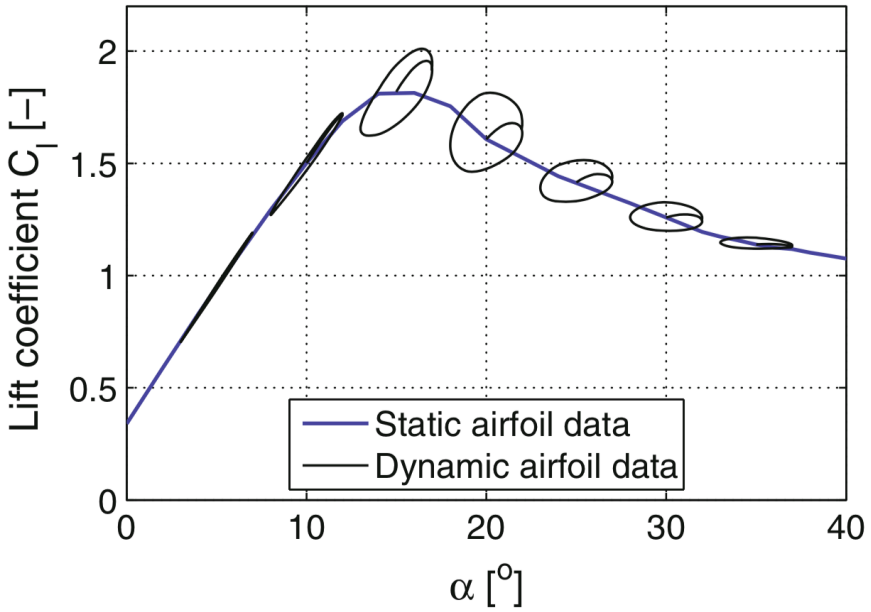
$$\mathbf{w} = \mathbf{w}_0 \left[1 + \frac{r}{R} \tan\left(\frac{\chi}{2}\right) \cos\left(\theta_{r,b} - \theta_{r,w}\right)\right]$$

$$\chi = \tan^{-1}\left(\frac{\hat{\mathbf{n}} \cdot \mathbf{u}'}{|\mathbf{u}'|}\right)$$

$$\chi = (0.6a + 1)\,\theta_{yaw}$$

Dynamic stall model:

- The **dynamic stall** is modelled using the **separation factor** f_s between the **inviscid** and the **fully separated lift coefficients** proposed by Stig Øye.

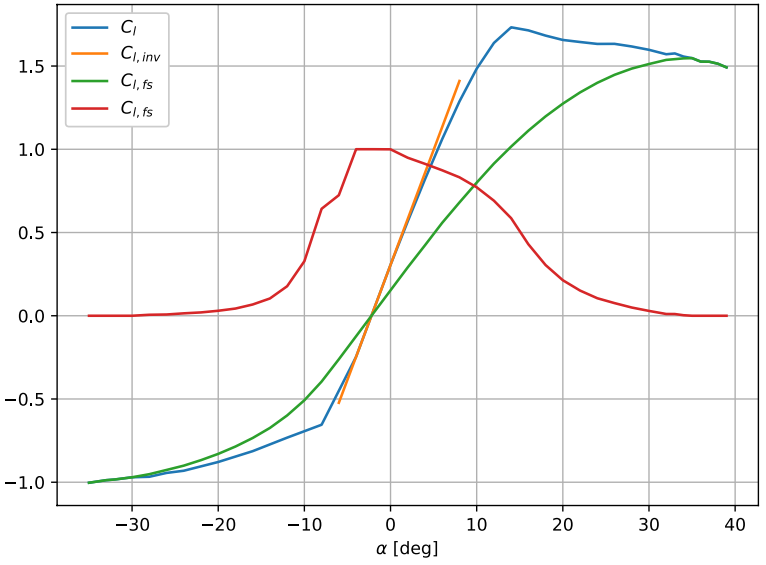


$$C_l = f_s C_{l,inv}(\alpha) + (1 - f_s) C_{l,fs}(\alpha)$$

$$\frac{df_s}{dt} = \frac{f_{s,st} - f_s}{\tau_{fs}} \qquad \tau_{fs} = \frac{4c}{u_{rel}}$$

$$C_{l,inv}(\alpha) = \left.\frac{dC_l}{d\alpha}\right|_{\alpha_0} (\alpha - \alpha_0)$$

$$C_{l,fs}(\alpha) = \begin{cases} \frac{1}{2} \left.\frac{dC_l}{d\alpha}\right|_{\alpha_0} (\alpha - \alpha_0) & \alpha \ll \alpha_{inv} \\ C_l & \alpha \geq \alpha_{inv} \end{cases}$$



Numeric tool:

- The software **code** is implemented in Python using the **object-oriented** concept, organising the attributes and functions into classes of objects.

Model Script

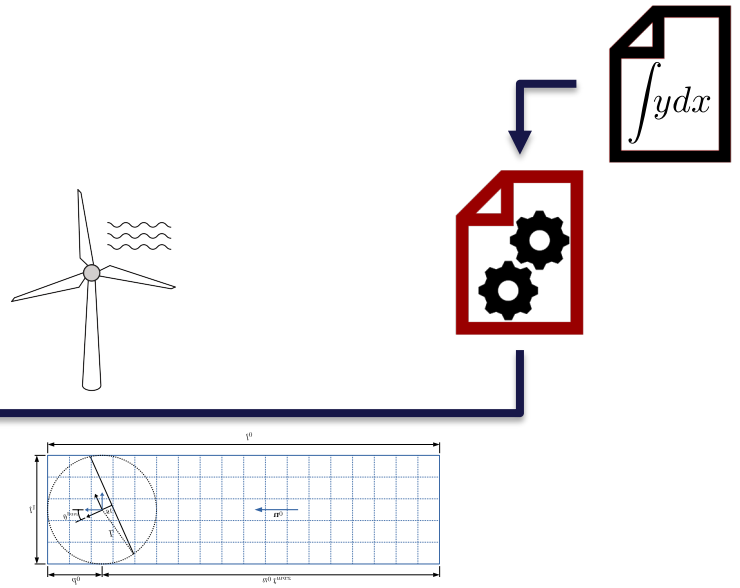
- Create objects
- Create differential equation
- Time integration

Wind Turbine
Wind turbine structural

Wind Box

Simulation object

Controllers



Study cases:

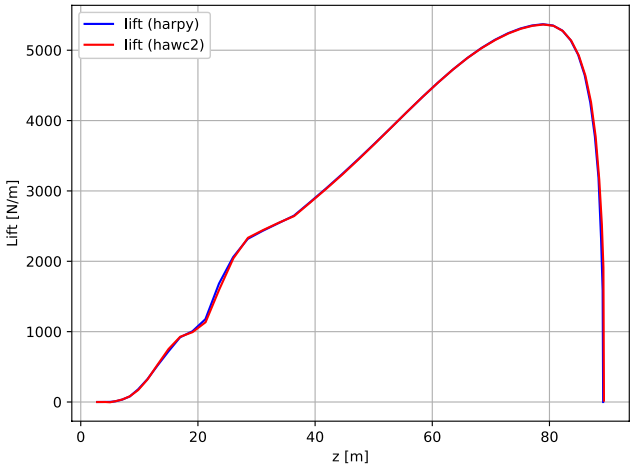
- In order to evaluate the implementation, the results given by the project aeroelastic code under different load scenarios were compared with DTU's aeroelastic code HAWC2.

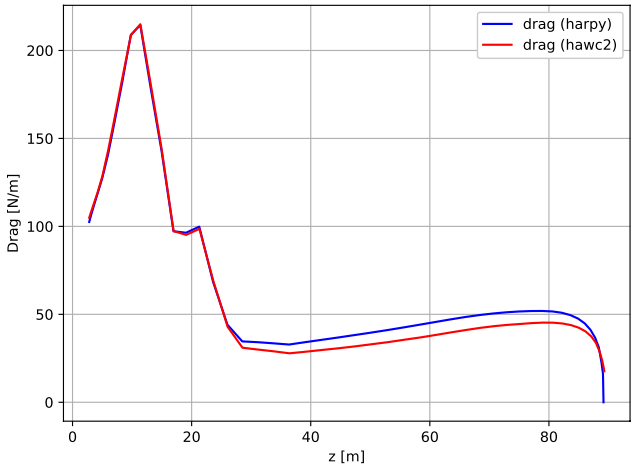
Case	θ_{yaw} $[^o]$	θ_{tilt} $[^o]$	θ_{cone} $[^o]$	shear	TI	Stiff
1.0	0	0	0	0	0	yes
1.1	0	0	2.5	0	0	yes
1.2	0	5	2.5	0	0	yes
1.3	-15	5	2.5	0	0	yes
1.4	0	5	2.5	0.2	0	yes
2.0	0	5	2.5	0.2	0	no
2.1	0	5	2.5	0.2	0.203	no

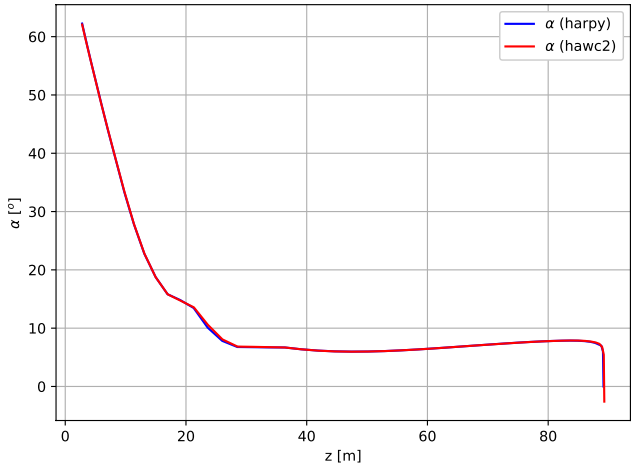
Case 1.0:

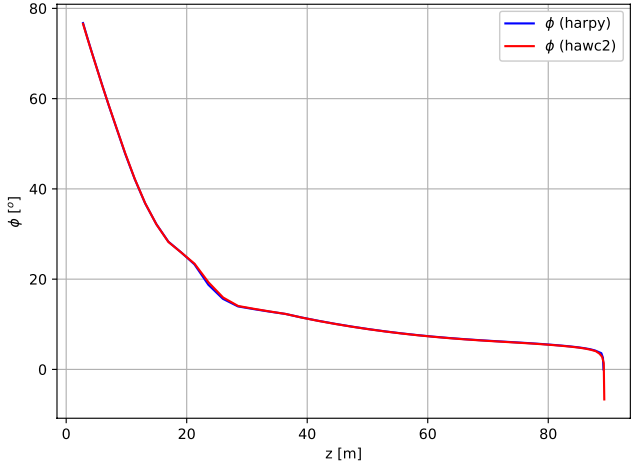
Case	θ_{yaw} [^{o}]	θ_{tilt} [^{o}]	θ_{cone} [^{o}]	shear	TI	Stiff
1.0	0	0	0	0	0	yes

Validation







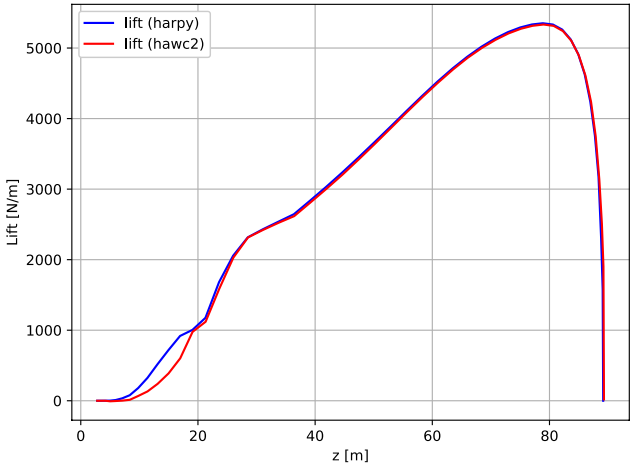


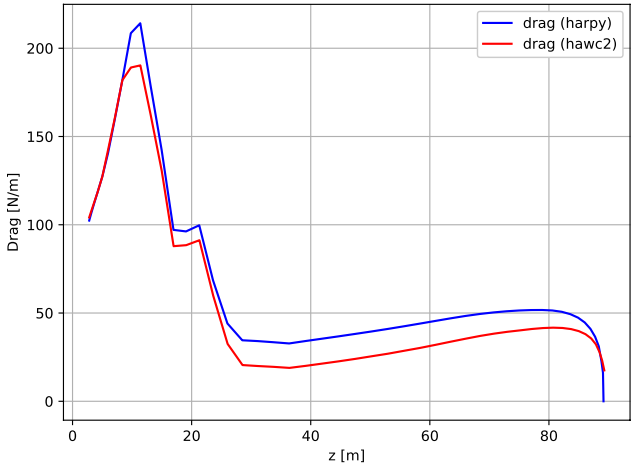
	Harpy	HAWC2	Difference [%]
C_P	4.783e-01	4.841e-01	1.200e+00

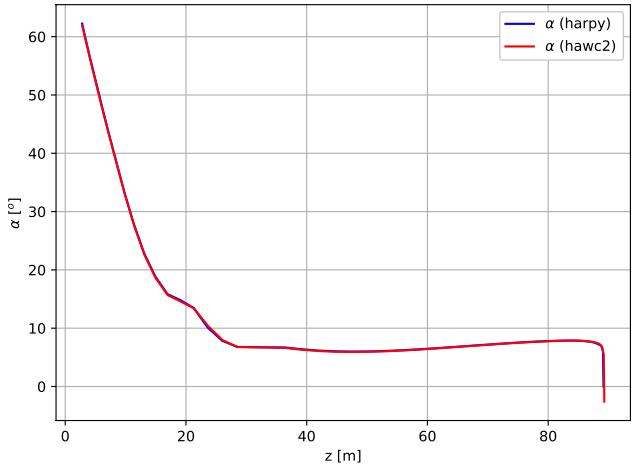
Case 1.1:

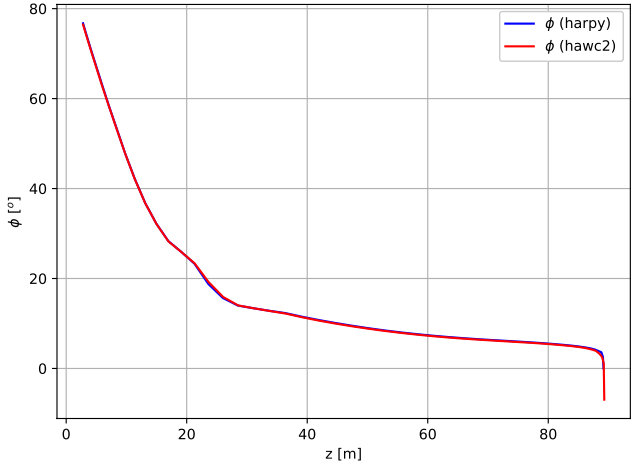
Case	θ_{yaw} $[^o]$	θ_{tilt} $[^o]$	θ_{cone} $[^o]$	shear	TI	Stiff
1.1	0	0	2.5	0	0	yes

Validation







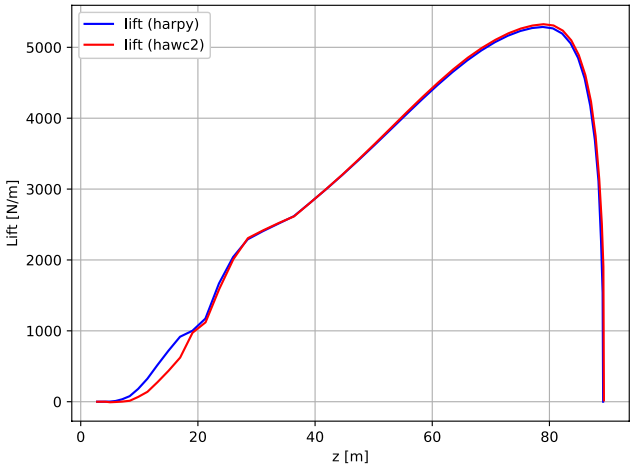


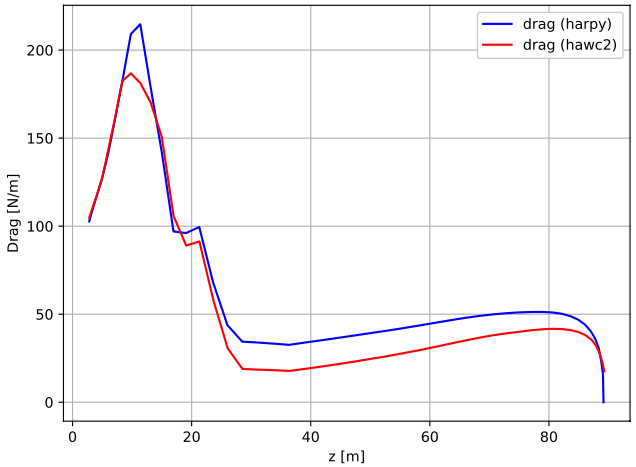
	Harpy	HAWC2	Difference [%]
C_P	4.772e-01	4.783e-01	2.164e-01

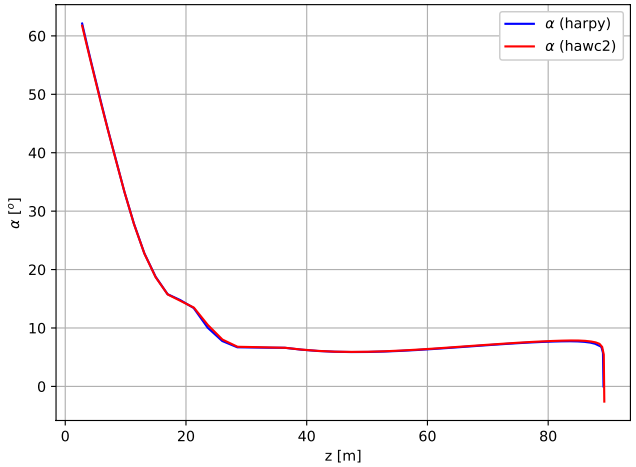
Case 1.2:

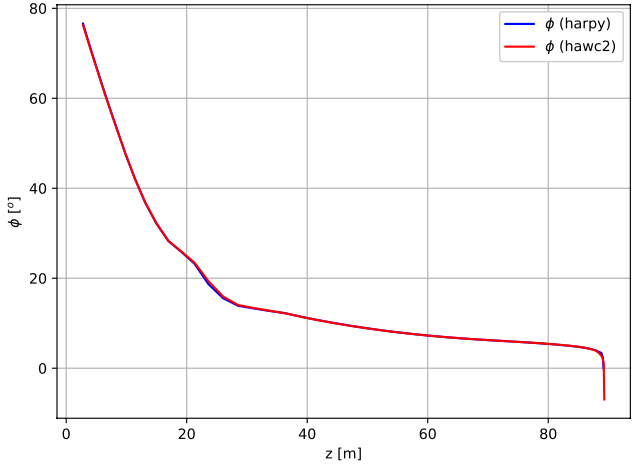
Case	θ_{yaw} $[^o]$	θ_{tilt} $[^o]$	θ_{cone} $[^o]$	shear	TI	Stiff
1.2	0	5	2.5	0	0	yes

Validation







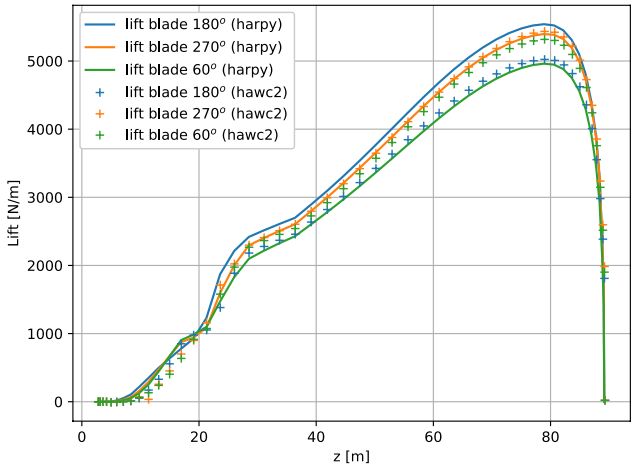


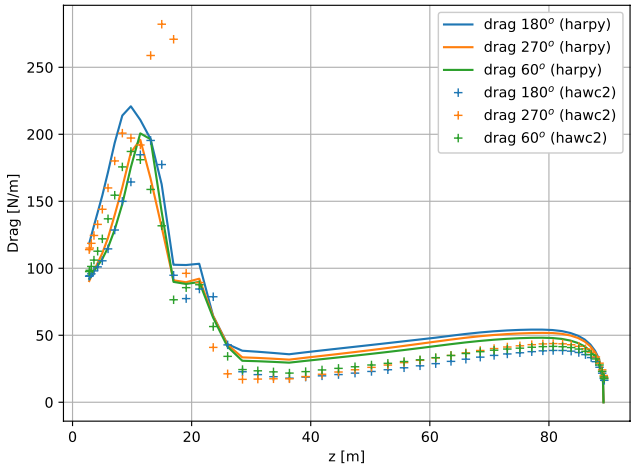
	Harpy	HAWC2	Difference [%]
C_P	4.760e-01	4.779e-01	3.961e-01

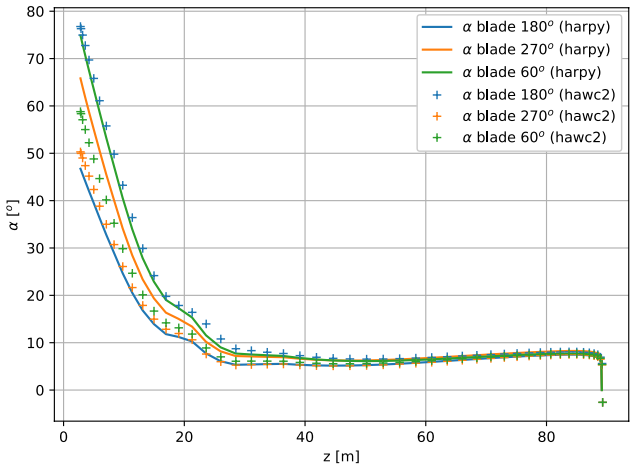
Case 1.3:

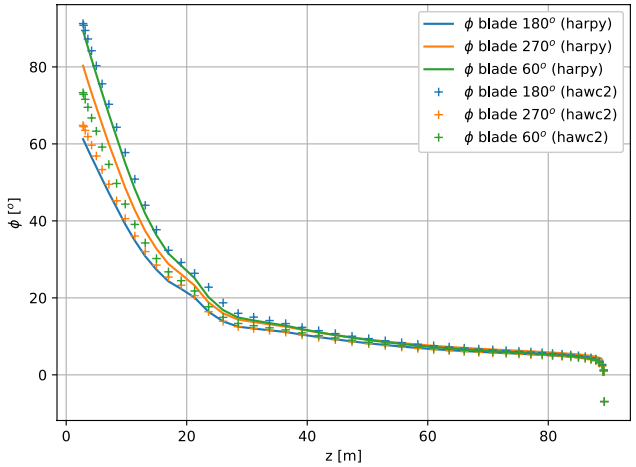
Case	θ_{yaw} [^{o}]	θ_{tilt} [^{o}]	θ_{cone} [^{o}]	shear	TI	Stiff
1.3	-15	5	2.5	0	0	yes

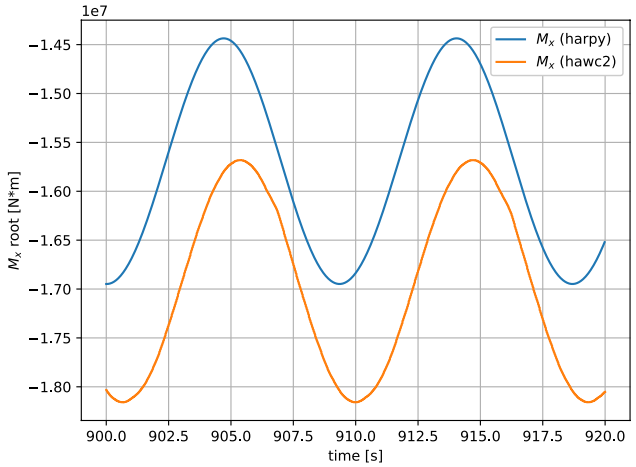
Validation

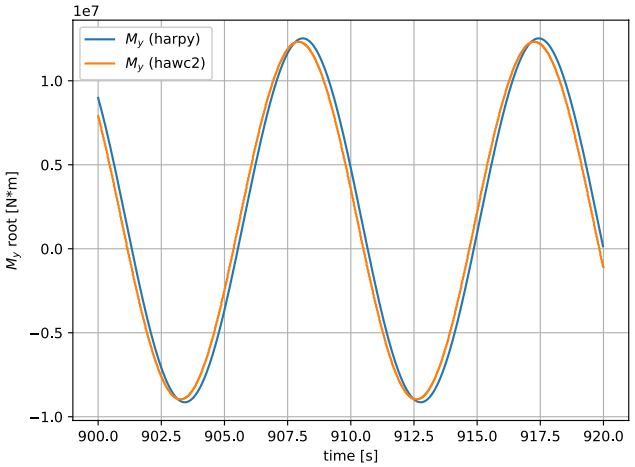










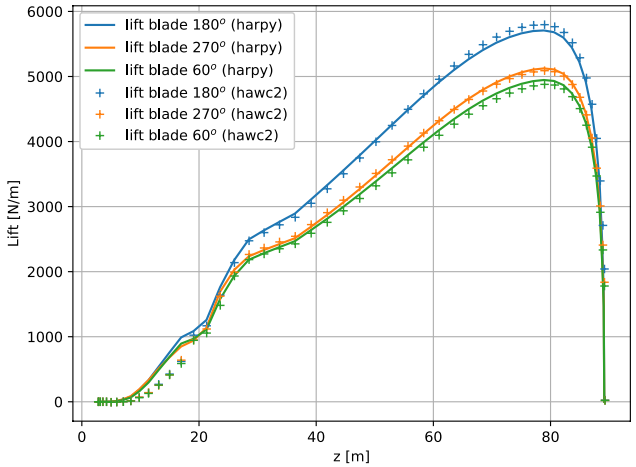


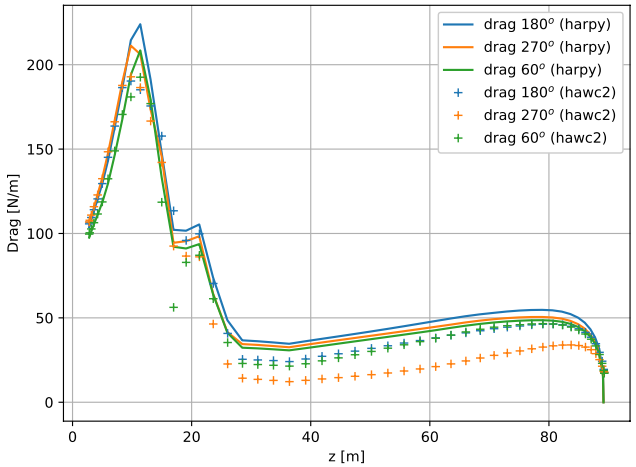
	Harpy	HAWC2	Difference [%]
C_P	4.610e-01	4.556e-01	1.182e+00
Shaft M_y [Nm]	5.365e+06	5.316e+06	1.546e+00

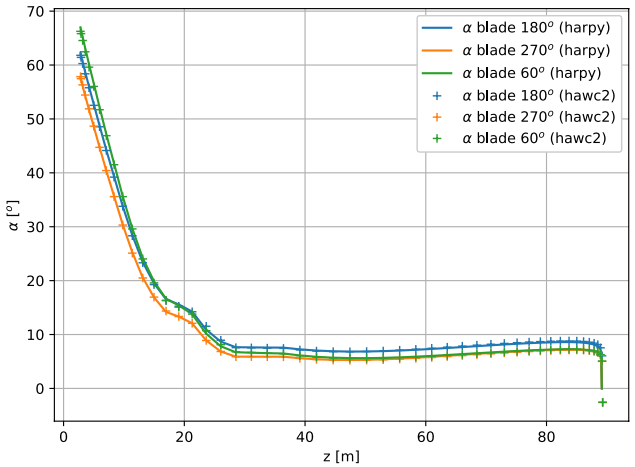
Case 1.4:

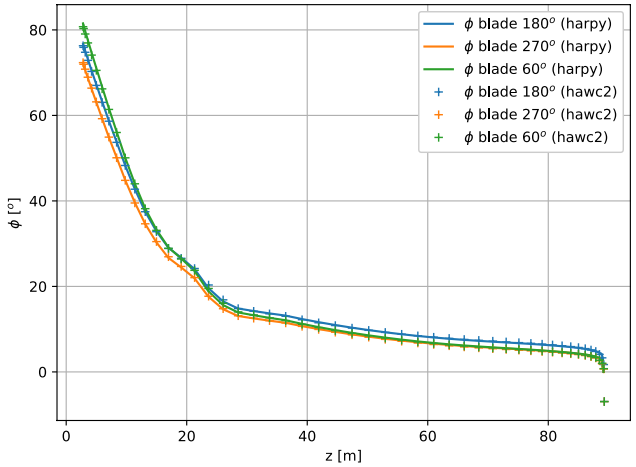
Case	θ_{yaw} [^{o}]	θ_{tilt} [^{o}]	θ_{cone} [^{o}]	shear	TI	Stiff
1.4	0	5	2.5	0.2	0	yes

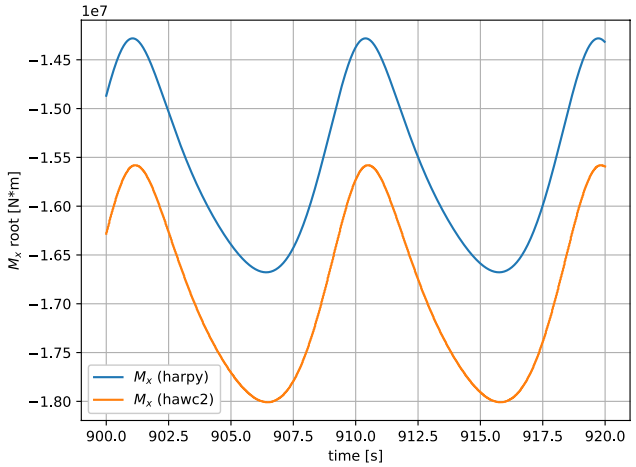
Validation

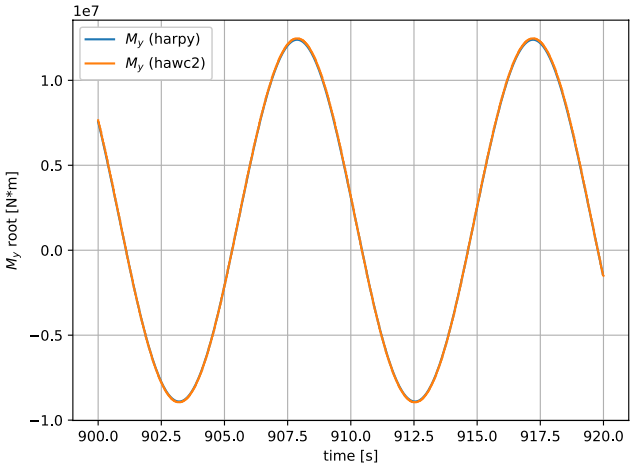










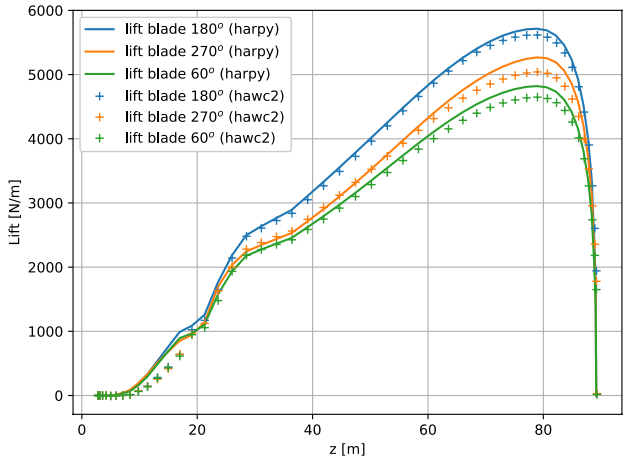


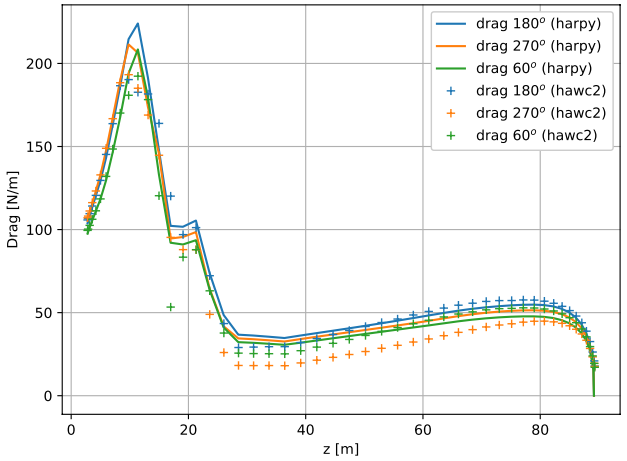
	Harpy	HAWC2	Difference [%]
C_P	4.623e-01	4.669e-01	9.827e-01

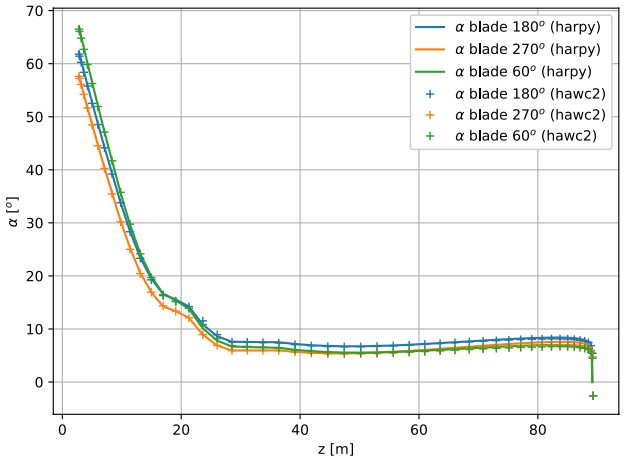
Case 2.0:

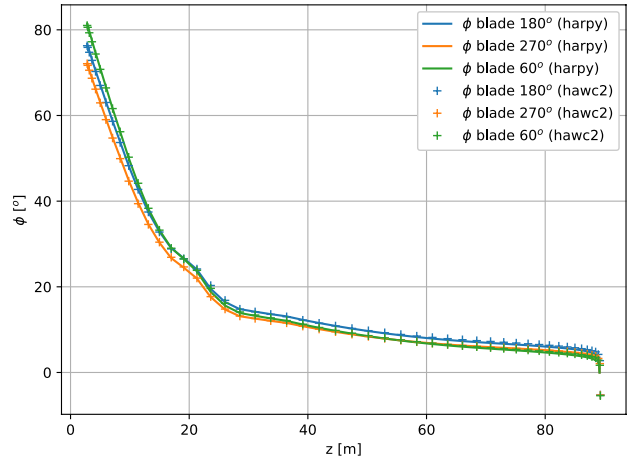
Case	θ_{yaw} [^{o}]	θ_{tilt} [^{o}]	θ_{cone} [^{o}]	shear	TI	Stiff
2.0	0	5	2.5	0.2	0	no

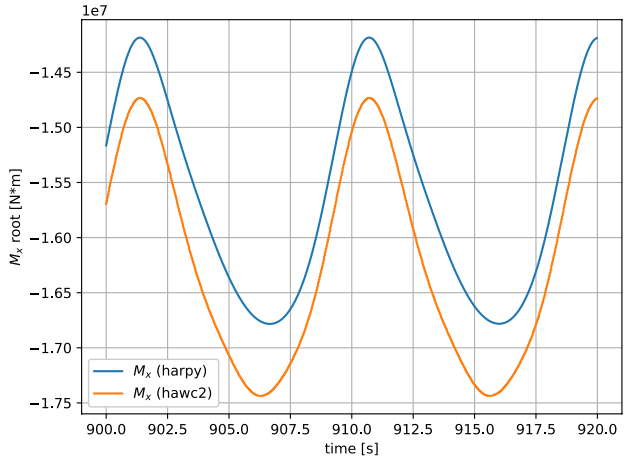
Validation

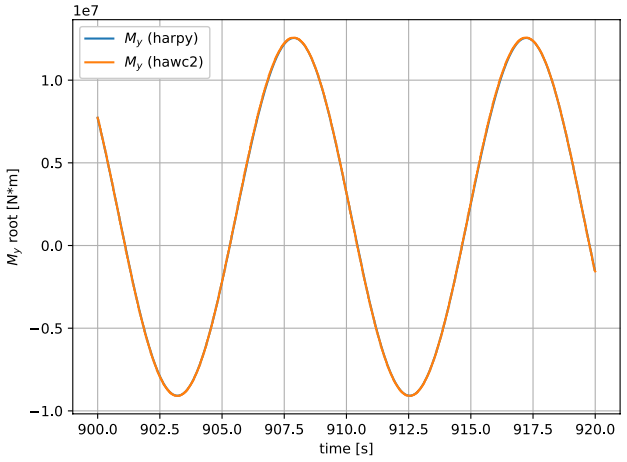


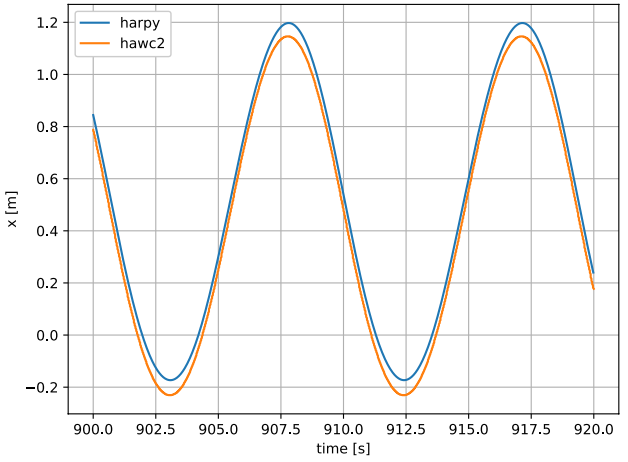


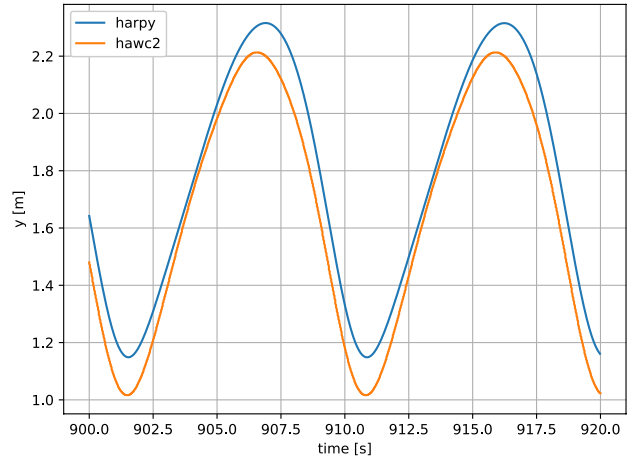










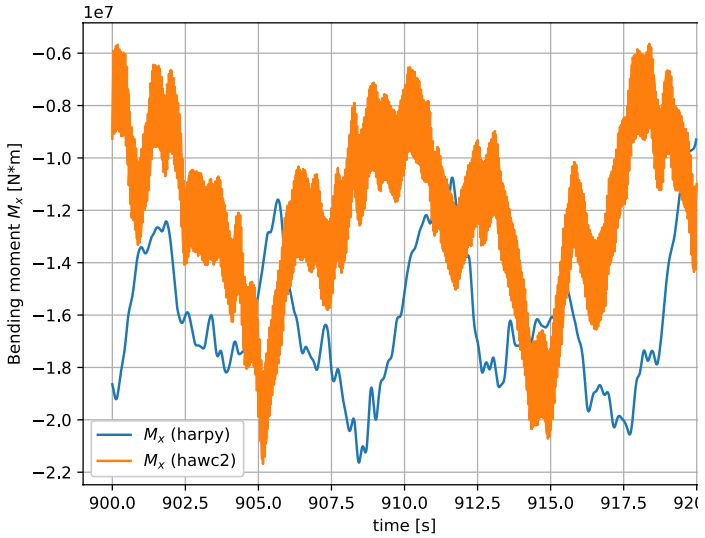


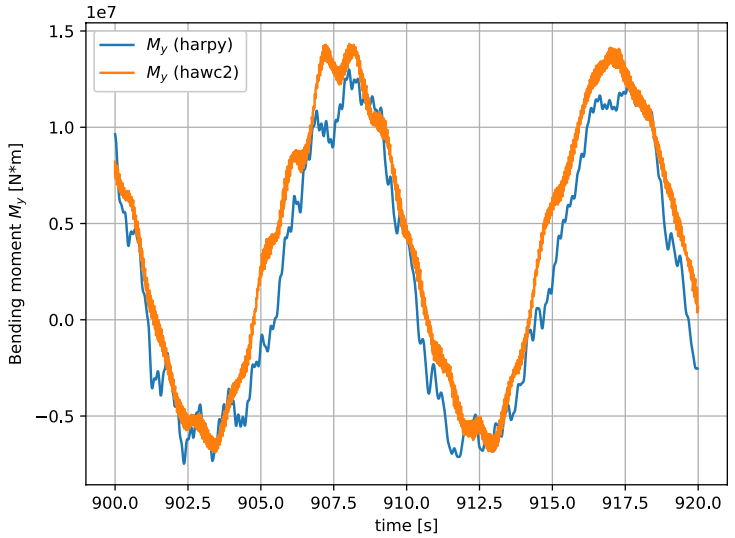
	Harpy	HAWC2	Difference [%]
C_P	4.607e-01	4.635e-01	5.997e-01

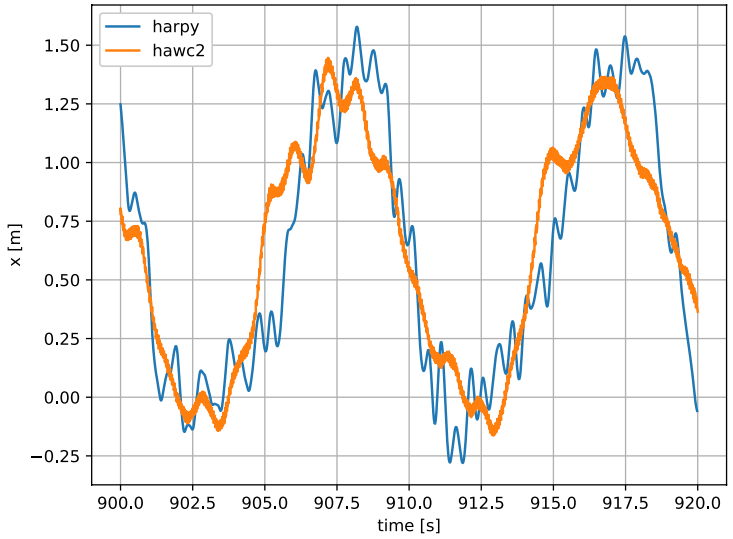
Case 2.1:

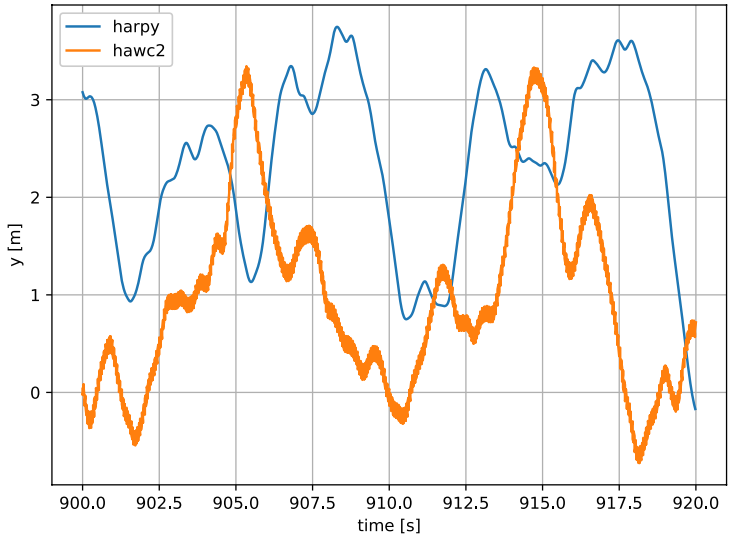
Case	θ_{yaw} [^{o}]	θ_{tilt} [^{o}]	θ_{cone} [^{o}]	shear	TI	Stiff
2.1	0	5	2.5	0.2	0.203	no

Validation









$$M_{ar} = \sqrt{M_a \left(M_a + M_m\right)}$$

$$M_{aq} = \left[\frac{\sum_{j=1}^{k} N_j \left(M_{ar,j}\right)^m}{N_B}\right]^{\frac{1}{m}}$$

	Harpy	HAWC2	Error [%]
$M_{y,eq}$ [Nm]	6.264e+06	8.073e+06	2.241e+01

Questions