

# A novel fixed-azimuth blade-element real-time rotor simulation model: FABES

Miguel González Cuadrado\*

The FABES model is a novel approach to blade-element real-time rotor simulation that, unlike traditional approaches, can run in constant time independent of rotational speed without loss of accuracy, and can therefore be used with no compromise for tail rotors. The FABES model implements a local free-form inflow field that can respond appropriately to highly dynamic maneuvers, non-uniform ground effect, local interference with fuselage and with other rotors, and can compute rotor vibrations directly from its rotor state representation. It has been developed and implemented by the author for Indra Sistemas as a generic, parameterizable model, in the scope of a larger object-oriented real-time simulation environment.

## Nomenclature

$a_0$	Fourier expansion 0th-order coefficient (p. 6)
$a_i$	Fourier expansion $i$ th-order cosine coefficient (p. 6)
$b$	Number of rotor blades (p. 9)
$b_i$	Fourier expansion $i$ th-order sine coefficient (p. 6)
$\beta_e$	Blade element local side-slip angle (p. 19)
$c_d$	Blade element drag coefficient (p. 17)
$c_l$	Blade element lift coefficient (p. 17)
$c_m$	Blade element pitch moment coefficient (p. 17)
$e_\beta$	Flapping hinge offset (p. 7)
$H$	Flapping feedback matrix (p. 8)
$k_G$	Ground effect factor (p. 13)
$M$	Blade element local Mach number (p. 13)
$M_{\beta_E}$	Total external flapping moment (p. 7)
$n_a$	Number of discretization azimuths (p. 4)
$n_e$	Number of blade elements per blade (p. 4)
$n_F$	Order of the Fourier expansion (p. 6)
$Q_{a_0}$	“Virtual work” for $a_0$ (p. 7)
$Q_{a_i}$	“Virtual work” for $a_i$ (p. 7)
$Q_{b_i}$	“Virtual work” for $b_i$ (p. 7)
$R$	Rotor radius (p. 19)
$r$	Blade element offset (p. 13)
$S$	Rotor area (p. 9)
$\mathbf{s}(t)$	Vector of Fourier coefficients (p. 6)
$\mathbf{T}$	Rotor traction (p. 9)
$\mathbf{v}_0$	Aerodynamic velocity with respect to the unperturbed air (p. 9)
$\mathbf{v}_i$	Previous inflow velocity (p. 9)
$\mathbf{v}_i^+$	Resulting inflow velocity (p. 9)
$x_B$	Rotor blade reference $x$ -axis (p. 11)
$x_{B'}$	Rotor virtual blade reference $x$ -axis (p. 11)
$x_H$	Helicopter body reference $x$ -axis (p. 11)
$x_R$	Rotor reference $x$ -axis (p. 11)

---

\*Simulation Expert Engineer and Technical Manager, Indra Sistemas, mgcuadrado@indra.es

$y_B$	Rotor blade reference $y$ -axis (p. 11)
$y_{B'}$	Rotor virtual blade reference $y$ -axis (p. 11)
$y_H$	Helicopter body reference $y$ -axis (p. 11)
$y_R$	Rotor reference $y$ -axis (p. 11)
$z_B$	Rotor blade reference $z$ -axis (p. 11)
$z_{B'}$	Rotor virtual blade reference $z$ -axis (p. 11)
$z_G$	Distance to the ground (p. 13)
$z_H$	Helicopter body reference $z$ -axis (p. 11)
$z_R$	Rotor reference $z$ -axis (p. 11)

#### *Subscripts*

B	Rotor blade (p. 11)
B'	Rotor virtual blade (p. 11)
E	External (p. 7)
G	Ground effect (p. 13)
H	Helicopter body (p. 11)
R	Rotor (p. 11)

#### *Symbols*

$\alpha_e$	Blade element local angle of attack (p. 12)
$\beta$	Rotor blade flapping (p. 11)
$\beta_{\max}$	Flapping angle upper limit (p. 11)
$\beta_{\min}$	Flapping angle lower limit (p. 11)
$\delta_3$	Flapping-to-pitch coupling angle (p. 17)
$\Delta M$	Dynamic equilibrium error (p. 7)
$\Delta t_b$	Delay between consecutive blade sweeps (p. 9)
$\Omega$	Rotor rotational speed around its axis (p. 4)
$\psi$	Azimuth angle in the rotor disk (p. 4)
$\rho$	Local air density (p. 13)
$\varsigma$	Rotor blade lagging (unusual sign convention) (p. 11)
$\theta$	Blade pitch (p. 12)

## I. Introduction

THIS paper describes FABES, the “Fixed-Azimuth Blade-Element” real-time rotor simulation model developed by the author for Indra Sistemas. It details the high-fidelity requirements and real-time constraints that shaped FABES, and its features and improvements on existing blade-element models. It also gives a brief technical description of the model and its implementation, and an overview of the results obtained from its validation.

This section gives a brief account of the circumstances and motivations behind the FABES model: the helicopter training simulation center program for which it was initially developed; the problems that made traditional approaches unfit; the special requirements of that program; and Indra Sistemas’ long-term objectives in the area of real-time simulation.

### I.A. The helicopter training simulation center program

In 1999, Indra Sistemas was awarded a contract by the Spanish Army to develop and build a training center for helicopter pilots and maintenance staff, including

- high-fidelity full mission simulators for a medium-lift conventional-rotor helicopter and a heavy-lift tandem-rotor helicopter;
- fixed-based simulators for the same helicopters;
- maintenance simulators for the same helicopters;
- a high-fidelity synthetic environment integrated with the four flight training devices, allowing joint and independent training, controlled by a general tactical station;

- on-board instructor stations;
- miscellaneous facilities for observation, communications, debriefing, et cætera.

Indra Sistemas had not developed previously helicopter flight models that would meet the fidelity requirements for this project. Two alternatives were considered: commercial models, and in-house development of a complete helicopter flight model. Expecting long-term returns in helicopter simulator development, Indra Sistemas opted for the second alternative, and commissioned the author with the development of a new flight simulation environment; the environment had to be generic, parameterizable, and reusable.

A survey of available rotor simulation code and models showed that traditional models failed in some important issues for the program at hand. The author developed and tested a new concept for real-time rotor simulation, that included some innovations that would solve the problems encountered with traditional models; when it proved to be sound and efficient, he developed it fully into the FABES model.

### **I.B. Problems with traditional approaches**

Traditional approaches for blade-element rotor models usually consist of an integrated description of rotor blade kinematics, local per-blade-element aerodynamic force and moment generation, blade dynamics subject to kinematic constraints, and global inflow field model. The treatment of blade dynamics leads to ordinary differential equations that describe the movement of each physical blade and its blade elements with respect to time, which makes it difficult to treat the inflow field locally.

These approaches usually have at least these problems:

- with higher rotational speed, at least one of the following adverse effects result:
  - either accurateness of the results decreases,
  - or required time step decreases (i.e. required simulation frequency increases),
  - or execution time per time step increases;
- usually, for that reason, blade-element rotor models are not used for tail rotors, since they have very high rotational speeds;
- global inflow field models are too simplistic to yield good results outside of the set of maneuvers for which they have been developed (they yield unrealistic results in e.g. highly dynamic maneuvers, high-speed cruise, steep descents, auto-rotation, hover with highly asymmetrical mass configurations, et cætera);
- global inflow field models are usually too simplistic to accurately predict and accommodate aerodynamic interferences with other rotors or with other parts of the helicopter (of which the most important is the fuselage)<sup>a</sup>.

### **I.C. Requirements of the helicopter training simulation center program**

The special fidelity requirements of the program for which it was originally developed added some specific goals for the FABES model:

- global inflow field models do not take into account non-uniform ground effect (as when partially over a ship deck)<sup>b</sup>; the rotor model for that program was required to handle non-uniform ground effect with appropriate accurateness;
- in the same line, the model was required to accept local non-uniform inflow field contributions, like wakes from other aircraft, or vertical currents from forest fires.

---

<sup>a</sup>The global inflow field can be modified locally in order to take into account non-uniform interference (or non-uniform ground effect), but the representativeness of the result is questionable, since local modifications affect the global inflow field, and they have no dynamics.

<sup>b</sup>See previous footnote.

### I.D. Company’s requirements for reusability

Internally, Indra Sistemas’ strategic vision in the domain of helicopter simulators for training required a reusable simulation environment with a generic, parameterizable rotor model. This led to the following additional goals for the FABES model:

- the FABES model had to be able to cope with any kind of main and tail rotor, by flexible construction-time parameterization; the model itself had to describe everything that is true of all rotors (whether tandem, main, or tail rotors), while leaving for parameterization everything that can change from rotor to rotor;
- the parameterization of the FABES model had to be done by specifying directly measurable magnitudes with physical meaning;
- the FABES generic model had to be implemented as fully reusable code, i.e. not one line of its code should have to be changed in order to use it for a different rotor.

## II. The approach for FABES

In this section, the FABES mathematical model is discussed. First, the novel features of the model are presented, including its basic approach to the problem of modeling the rotor, and the new problems that this approach creates, with their solutions. Then, we examine the features that, although not exclusive to the FABES model, have to be considered in a new way because of the model peculiarities.

The implementation of the model is discussed later in section III.

### II.A. The novel features

This section discusses how the FABES model differs from traditional rotor models, both in its conception and in the resulting new problems and solutions.

#### II.A.1. From following blades to watching blades

Traditional blade-element rotor models usually simulate the evolution of each rotor physical blade as it turns around the rotor axis. Each blade is treated as a solid whose movement through the local air mass (including local wind and induced flow) generates aerodynamic forces and moments at each blade element; the blade equations of motion are solved taking into account these forces and moments, other external forces and moments, and kinematic restrictions. The induced flow is hard to treat locally, since it is hard to maintain local inflow velocity from one blade sweep to the next, so a global induced flow model is used. We say that such rotor models *follow* the blades. This is depicted by figure 1, in which the physical blades are partitioned in blade elements; in this figure,  $n_e$  is the number of blade elements per blade. The azimuth  $\psi$  of each physical blade evolves as the integral of  $\Omega$ , which is the rotational speed of the rotor around its axis:

$$\psi = \int \Omega dt \tag{1}$$

In contrast, the FABES model does not follow the blades, but *watches* them as they pass through defined azimuth values: instead of maintaining the state of each blade, the FABES model maintains the state that each blade has each time it has a defined fixed azimuth value. The states at which a blade has one of those defined fixed azimuth values is described as a “virtual blade”; virtual blades “capture” the state of blades as they pass through given azimuths. The azimuths at which there is a virtual blade are called “discretization azimuths”. Virtual blades are divided into virtual blade elements, in the same way as physical blades in traditional approaches; each virtual blade element remains at a fixed location in the rotor disk. See figure 2 for a graphical description of the virtual blade and blade element discretization just described; in this figure,  $n_a$  is the number of discretization azimuths, and  $n_e$  is, as before, the number of blade elements per blade.

Since virtual blades represent the state of physical blades at given azimuths, they have full kinematics with non-null linear and angular velocities, in spite of being at fixed azimuths. For the same reason, they have flapping and lagging, but their flapping and lagging velocities do not integrate into their flapping and

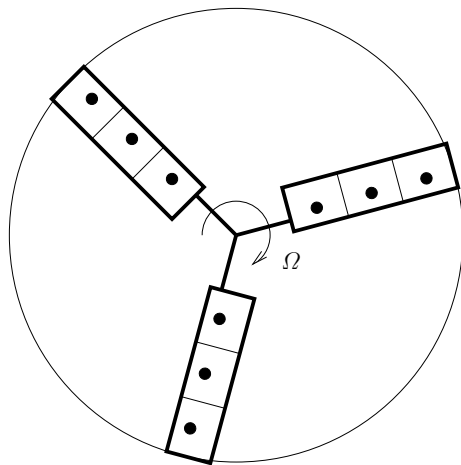


Figure 1. Blade elements fixed to the blades: following ( $n_e = 3$ )

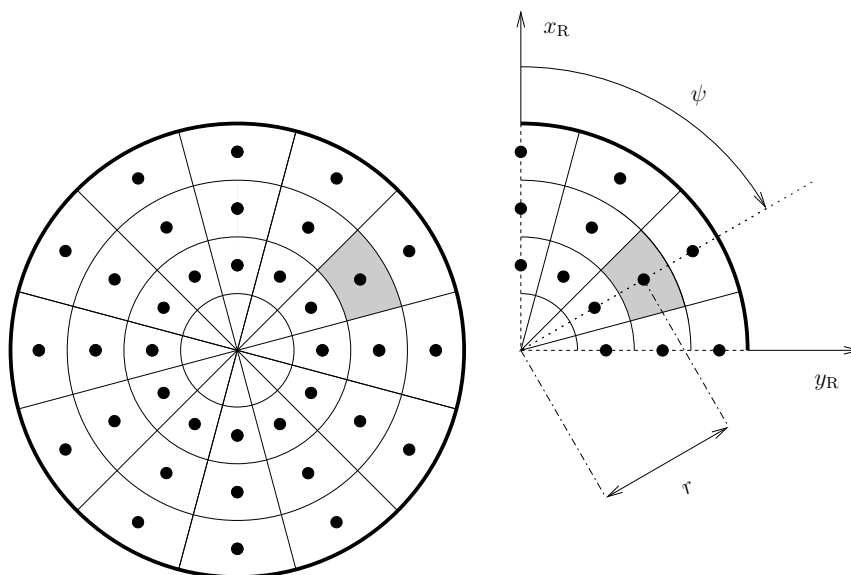


Figure 2. Blade elements fixed to the rotor disk: watching ( $n_e = 3, n_a = 12$ )

lagging angles directly (but they do integrate as expected with respect to azimuth, which is ensured by their kinematics description based on Fourier coefficients; see section II.A.2). Their aerodynamics is consistent with their instantaneous velocity (including flapping and lagging velocities), taking into account their own induced flow and local wind; their inertial forces and moments are consistent with their instantaneous acceleration; their weight is consistent with their position and orientation with respect to the earth.

Since the virtual blade elements have fixed locations in the disk, it becomes easy to maintain an inflow velocity for each of them, representing the tube of inflow field that passes through the virtual blade element; see section II.A.4 for a complete description.

### II.A.2. Fourier description of blade kinematics

For a given blade, flapping  $\beta(t)$  and lagging  $\zeta(t)$  are defined as functions of azimuth  $\psi$  through the use of Fourier coefficients that evolve with time:

$$f(t) \equiv f(\mathbf{s}(t), \psi) \quad (2)$$

$$= a_0(t) + \sum_{i=1}^{n_F} (a_i(t) \cos i\psi + b_i(t) \sin i\psi) \quad (3)$$

where  $\mathbf{s}(t)$  is the vector of the Fourier coefficients used, i.e. up to the order of the Fourier expansion  $n_F$ . The Fourier coefficients are independent for  $\beta(\psi)$  and  $\zeta(\psi)$ .

This substitution of  $f(t)$  by  $f(\mathbf{s}(t), \psi)$  makes it possible to shift from *following* the blade elements as they turn around the rotor axis to *watching* them as they pass through *fixed* discretization azimuths (see section II.A.1), *while maintaining kinematic coherence between the virtual blades*, since the description of flapping and lagging kinematics is global to the rotor, although its evolution is determined by local equations at each virtual blade (see section II.A.3). This requires that the evolution of the vectors of Fourier coefficients  $\mathbf{s}(t)$  be computed for flapping and lagging, so as to satisfy the dynamic equations of the blade at each azimuth discretization value.

Unlike in classical approaches, the rotor kinematics is known for all considered discretization azimuths, *at all times*, since it is described by functions global to the rotor through their Fourier coefficients, which are computed continuously. The Fourier coefficients make it easy to obtain, not only the flapping and lagging at each azimuth, but also their time derivative and second derivative, when the evolution of the coefficients is considered slow compared to the change they describe as a blade rotates:

$$f(\psi) = a_0 + \sum_{i=1}^{n_F} (a_i \cos i\psi + b_i \sin i\psi) \quad (4)$$

$$\frac{df}{dt} \simeq \Omega \cdot \sum_{i=1}^{n_F} (-ia_i \sin i\psi + ib_i \cos i\psi) \quad (5)$$

$$\frac{d^2f}{dt^2} \simeq \Omega^2 \cdot \sum_{i=1}^{n_F} (-i^2a_i \cos i\psi - i^2b_i \sin i\psi) \quad (6)$$

### II.A.3. Evolution of the Fourier description

With traditional approaches, total blade forces and moments are computed and applied to each blade, and blade motion is determined by solving numerically the classical dynamics ordinary differential equations, taking into account the constraints of blade kinematics. In the FABES model, the relation from blade forces and moments to blade kinematics is no longer direct, since blade kinematics is described by Fourier coefficients. Instead, the feedback loop depicted in figure 3 is implemented.

The general problem in determining the instantaneous Fourier coefficients is to satisfy the dynamic equilibrium *at each discretization azimuth, for each time step*. The dynamic equilibrium equations establish that, at each discretization azimuth, the total forces and moments applied on the blades must be compatible with the blade linear and angular acceleration, subject to the blade kinematic constraints. The blade kinematic constraints can be accounted for by expressing these dynamic equilibrium equations directly in the degrees of freedom of the blades, i.e. flapping and lagging. Thus, for instance, the dynamic equilibrium

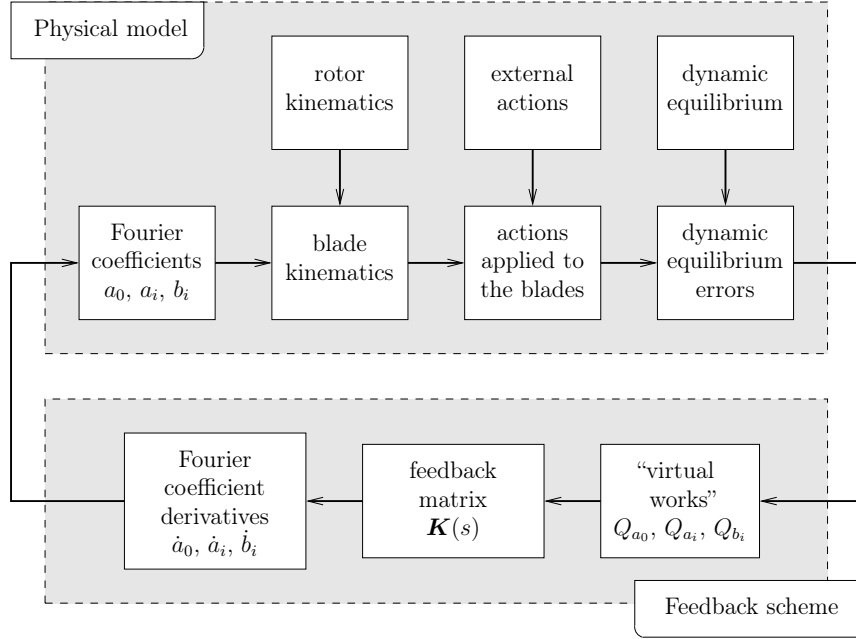


Figure 3. Feedback loop for the flapping and lagging Fourier description

in flapping can be expressed as

$$M_{\beta_E} - \Omega^2 \beta \int_0^R r (r - e_\beta) dm - \ddot{\beta} \int_0^R (r - e_\beta)^2 dm = 0 \quad (7)$$

where  $M_{\beta_E}$  is the total external flapping moment applied to the blade, including the moment resulting from aerodynamics, weight, and other external forces and moments, but not including centrifugal or inertial flapping moment;  $e_\beta$  is the flapping hinge offset; and  $dm$  is the differential blade mass along blade span. In this equation,  $\beta$  and  $\ddot{\beta}$  are computed per equations (4) and (6).

But the general problem is too computationally expensive to solve in real time, and it may not even have any solution in the general case. Instead of solving the general problem, the FABES model dictates an evolution for the flapping and lagging Fourier coefficients that *tends* all the time to solve the real problem. The left-hand side of equation (7) is called the “flapping dynamic equilibrium error”, and it is fed back into the flapping Fourier coefficients in such a way that the error tends to zero at all times. The feedback scheme is devised and tuned for maximum speed of error convergence to zero; in the practice, the convergence speed approaches the practical limit for other systems, and is always well below useful characteristic times for training for typical integration rates in modern real-time simulators (at least 50 Hz). Usually, a convergence time of four times the integration time step (a convergence time of 80 ms for a 50 Hz simulation frequency) yields good, robust convergence.

In order to determine the optimum feedback scheme for the Fourier coefficients, we use the dynamic equilibrium error as a feedback error which acts on a first-order system defined by the Fourier coefficients<sup>c</sup>. By a procedure similar to the computation of virtual works in analytic mechanics, the equivalent of Lagrange multipliers are obtained for this first-order system:

$$Q_{a_0} \delta a_0 + \sum_{i=1}^n (Q_{a_i} \delta a_i + Q_{b_i} \delta b_i) = \int_{-\pi}^{+\pi} \Delta M(\psi) \delta \beta(\psi) d\psi \quad (8)$$

where  $\Delta M$  is the dynamic equilibrium error, and the terms  $Q_{a_0}$ ,  $Q_{a_i}$ , and  $Q_{b_i}$  are the “virtual works” for  $a_0$ ,  $a_i$ , and  $b_i$  respectively. The term  $\Delta M(\psi)$  can be made explicit and linearized around the working point:

$$\Delta M(\psi) \sim -k_A \beta' - k_I (\beta + \beta'') \quad (9)$$

<sup>c</sup>Equivalent results can be reached by finding the Fourier transform of the dynamic equilibrium error, or by considering the error in flapping or lagging acceleration instead of the dynamic equilibrium error. The fact that the base functions of the Fourier transform are mutually orthogonal explains that the equivalent development described in the main text as “virtual works” results in a feedback matrix with no coupling between Fourier coefficients of different orders.

where  $\beta'$  and  $\beta''$  are respectively the first and second derivatives of  $\beta$  with respect to  $\psi$ ,  $k_A\beta'$  is the linearized aerodynamic moment,  $k_I\beta$  is the linearized centrifugal moment, and  $k_I\beta''$  is the linearized inertial moment, all of them with respect to flapping.

The general feedback scheme can be written as

$$\frac{d}{dt} \begin{pmatrix} a_0 \\ a_1 \\ b_1 \\ \vdots \\ a_{n_F} \\ b_{n_F} \end{pmatrix} = H \cdot \begin{pmatrix} Q_{a_0} \\ Q_{a_1} \\ Q_{b_1} \\ \vdots \\ Q_{a_{n_F}} \\ Q_{b_{n_F}} \end{pmatrix} \quad (10)$$

where  $H$  is the flapping feedback matrix. Given the form of equation (9), it is readily seen that  $H$  needs only couple Fourier coefficients of the same order. Specifically, the linearized “virtual works” are

$$Q_{a_0} \sim -2\pi k_I a_0 \quad (11)$$

$$\begin{pmatrix} Q_{a_i} \\ Q_{b_i} \end{pmatrix} \sim \pi \begin{pmatrix} k_I (i^2 - 1) & -k_A i \\ +k_A i & k_I (i^2 - 1) \end{pmatrix} \begin{pmatrix} a_i \\ b_i \end{pmatrix} \quad (12)$$

and they can be decoupled by

$$\dot{a}_0 = \frac{1}{m_0} Q_{a_0} \quad (13)$$

$$\begin{pmatrix} \dot{a}_i \\ \dot{b}_i \end{pmatrix} = \frac{1}{m_i} \begin{pmatrix} -k_I (i^2 - 1) & -k_A i \\ +k_A i & -k_I (i^2 - 1) \end{pmatrix} \begin{pmatrix} Q_{a_i} \\ Q_{b_i} \end{pmatrix} \quad (14)$$

A characteristic convergence time  $\tau_0$  is chosen for this feedback scheme, which should be as small as possible, but not so small that it generates spurious vibrations; typically, a value in the order of four times the integration time step yields appropriate results. By combining equations (11), (12), (13), and (14), the poles are found to be at

$$p_0 = -\frac{2\pi}{m_0} k_I \quad (15)$$

$$p_i = -\frac{\pi}{m_i} \left( k_I^2 (i^2 - 1)^2 + k_A^2 i^2 \right) \quad (16)$$

The adjustment of  $m_0$  and  $m_i$  to yield characteristic convergence times of  $\tau_0$  gives the following values

$$m_0 = 2\pi k_I \tau_0 \quad (17)$$

$$m_i = \pi \left( k_I^2 (i^2 - 1)^2 + k_A^2 i^2 \right) \tau_0 \quad (18)$$

In order to implement the feedback scheme described by equations (13) and (14), equation (8) is fully developed into

$$Q_{a_0} = \int_{-\pi}^{+\pi} \Delta M(\psi) d\psi \quad (19)$$

$$Q_{a_i} = \int_{-\pi}^{+\pi} \Delta M(\psi) \cos i\psi d\psi \quad (20)$$

$$Q_{b_i} = \int_{-\pi}^{+\pi} \Delta M(\psi) \sin i\psi d\psi \quad (21)$$

which can be implemented by numerical integration over the discretization azimuths.

The development for the determination of the Fourier coefficients for the lagging motion is analogous to the one described for the flapping motion. Additionally, dampers can easily be accounted for by considering their effect on the moment computation (by adding their contribution to  $\Delta M$  in equations (19) to (21)) and on the feedback scheme tuning (by including their linearized contribution in equation (9) and forwarding it to the rest of the development).

#### II.A.4. Free-form, local inflow field

Having tied the discretization to fixed locations in the rotor disk, as described in section II.A.1, each blade element can now maintain its own inflow field tube. The inflow field is then composed of as many tubes as there are blade elements, and is as general as possible (with as many degrees of freedom as possible), and completely physically based, with no *a priori* form. It features full, free dynamics that can respond to dynamic conditions and local modifications of inflow field (such as non-uniform interference and ground effect), and a level of detail equal to the rotor disk discretization, which is useful for interference with other rotors and aerodynamic elements.

Any global inflow field theory can be used at this level, as long as care is exercised in adapting it to a partition in blade elements, so as to maintain the combined per-blade-element result analogous to the original global theory. Another option is to use an inflow field theory specifically developed for local determination of inflow at fixed locations in the rotor disk.

**LOCAL INFLOW FIELD BY MOMENTUM THEORY** The development given here uses as an example the momentum theory. In its global form, the momentum theory states that the effect of the rotor on the air mass is equivalent to accelerating the air in a cylinder of section equal to the rotor disk area.<sup>1</sup> By momentum considerations, the inflow velocity  $\mathbf{v}_i^+$  resulting from a rotor with area  $S$  that is generating a traction  $\mathbf{T}$ , with an aerodynamic velocity with respect to the unperturbed air  $\mathbf{v}_0$ , and a previous inflow velocity  $\mathbf{v}_i$ , with all these magnitudes in vector form except  $S$ , is

$$\mathbf{v}_i^+ = \frac{\mathbf{T}}{S} \frac{1}{2\rho \|\mathbf{v}_0 + \mathbf{v}_i\|} \quad (22)$$

Sometimes it is useful to liken the establishment of the resulting inflow velocity to the acceleration of a certain mass of air. This results in a first-order low-pass filter  $E(s)$  applied to  $\mathbf{v}_i$ ,

$$E(s) = \frac{1}{1 + \frac{m_s s}{S 2\rho \|\mathbf{v}_0 + \mathbf{v}_i\|}} \quad (23)$$

where  $m_s$  is the mass of air<sup>d</sup>.

Equation (22) is converted to deal with blade elements:

$$\mathbf{v}_i^+ = \frac{\Delta\mathbf{T}}{\Delta S} \frac{1}{2\rho \|\mathbf{v}_0 + \mathbf{v}_i\|} \quad (24)$$

where  $\Delta\mathbf{T}$  is the traction generated by the blade element divided by the number of virtual blades  $n_a$  and multiplied by the number of physical blades  $b$  (see section II.A.5),  $\Delta S$  is the rotor disk area covered by the blade element, and all the rest of magnitudes refer to the specific blade element (but  $S$  and  $m_s$  in equation (23) keep their global value, or are both distributed among blade elements following a certain criterion, resulting in inflow tubes unevenly accelerated among blade elements).

Additionally, if appropriate, the delay between consecutive blade sweeps can be taken into account by the following filter. Let  $\Delta t_b$  be the delay between consecutive blade sweeps:

$$\Delta t_b = \frac{2\pi}{b\Omega} \quad (25)$$

where  $b$  is the number of rotor blades. A pure delay can be modelled by the filter

$$G_{\text{ideal}}(s) = e^{-s\Delta t_b} \quad (26)$$

but since this filter is hard to mechanize, the following approximation is used instead:

$$G(s) = \frac{1}{\left(1 + \frac{1}{2}s\Delta t_b\right)^2} \quad (27)$$

In conclusion, connecting equation (24) with  $\mathbf{v}_i$  through the filters (23) and (27), the loop is closed by the following feedback formula:

$$\mathbf{v}_i(s) = G(s) \cdot E(s) \cdot \mathbf{v}_i^+(s) \quad (28)$$

<sup>d</sup>This formula comes from  $m_s \dot{\mathbf{v}}_i = \mathbf{T} - \mathbf{T}_v$ , where  $\mathbf{T}_v$  is the traction that results in an equilibrium inflow velocity equal to the current one, i.e.  $\mathbf{v}_i S 2\rho \|\mathbf{v}_0 + \mathbf{v}_i\|$ .

LOCAL INFLOW FIELD THROUGH OTHER INFLOW THEORIES As stated before, the fixed-location discretization featured by the FABES model supports any form of inflow theory. In the practice, the momentum theory will often be too inaccurate and applicable only to a small range of flight conditions. More advanced theories will have to be used, such as Young’s modified momentum theory, or specific or tuned inflow models. The way to incorporate them is very similar to what has been described for the momentum theory, and most often, only  $\mathbf{v}_i^+$  changes expressed as a function  $\mathbf{v}_i^+(\Delta\mathbf{T}, \Delta S, \mathbf{v}_0, \mathbf{v}_i)$ , as described by equation (24) for the momentum theory.

#### II.A.5. Force and moment integration

The forces and moments generated by the virtual blades are used to obtain the forces and moments generated by the physical blades. In most cases, the precise position of the physical blades is not important, and it suffices to add together the forces and moments generated by all virtual blades, then to divide by the number of virtual blades  $n_a$ , and multiply by the number of physical blades  $b$ . This approach is valid even when vibrations must be computed from the rotor forces and moments, since these can be obtained directly as explained in section II.A.6, without having to extract them from the evolution of forces and moments with time.

When the precise position of the physical blades is important for the computation of forces and moments, the data at the virtual blades can be numerically integrated for the range swept by each blade in one time step<sup>e</sup>.

#### II.A.6. Vibrations generated by the rotor

A bonus of the FABES model is the fact that it makes it easy to compute the vibrations generated by the rotor. Vibrations are obtained directly from the azimuth-wise distribution of forces and moments generated by the rotor, without any time-dependent computation.

In order to obtain the vibration characteristics of a force or moment in a given axis (usually  $x_R$ ,  $y_R$ , or  $z_R$ , though any other axis not necessarily aligned with the rotor reference frame can be used), one must first obtain the corresponding magnitude generated by each virtual blade. Let  $f(\psi)$  the value of that magnitude generated by the virtual blade at azimuth  $\psi$ . Its Fourier coefficients of non-zeroth order are obtained by

$$a_i = \frac{1}{\pi} \int_{-\pi}^{+\pi} f(\psi) \cos i\psi \, d\psi \quad (29)$$

$$b_i = \frac{1}{\pi} \int_{-\pi}^{+\pi} f(\psi) \sin i\psi \, d\psi \quad (30)$$

From these Fourier coefficients, the amplitude  $A_i$  and phase  $\phi_i$  for the vibration at a frequency  $i\Omega \text{ rad} \cdot \text{s}^{-1}$  are

$$A_i = \sqrt{a_i^2 + b_i^2} \quad (31)$$

$$\phi_i = \arg(a_i - jb_i) \quad (32)$$

where  $j$  is the imaginary unit.

But these values refer to a single blade. For the  $b$  blades of the rotor, the complex value  $A_i e^{i\phi_i}$  must be added for each blade, phased by the blade azimuth phase, and the result must be converted back to amplitude and phase. This results in vibrations that cancel out for all  $i\Omega$  when  $i$  is not a multiple of  $b$ . The vibrations that do not cancel out occur at frequencies  $bk\Omega$ , and their magnitude  $(A_{bk})_R$  and phase  $(\phi_{bk})_R$  can be computed from the values for  $A_i$  and  $\phi_i$  obtained from equations (31) and (32) as

$$(A_{bk})_R = bA_{bk} \quad (33)$$

$$(\phi_{bk})_R = \phi_{bk} \quad (34)$$

---

<sup>e</sup>The FABES model implementation does not include this kind of computation, since no application has been found for it.

### II.A.7. Flapping and lagging limitations

The natural method to implement flapping and lagging limitations is to detect when a virtual blade is hitting a limitation, and then to apply to that blade the exact moment that will bring it back within the limits. Nevertheless, this method makes the whole model more rigid, and can generate undesirable artifacts when limits are reached.

Instead, a simpler approach is used. Let the flapping limits be  $\beta_{\min}$  and  $\beta_{\max}$ . The procedure limits the flapping angle Fourier coefficients to ensure that

$$\beta([- \pi, + \pi]) \subset [\beta_{\min}, \beta_{\max}] \quad (35)$$

The Fourier coefficients are tested in ascending order: first  $a_0$ , then  $a_1$  and  $b_1$ , then  $a_2$  and  $b_2$ , et cætera. Whenever the coefficient or coefficients for an order are limited, the higher-order coefficients are all set to zero.

## II.B. Classical features with a novel implementation

### II.B.1. Conventions and reference axes

The rotor reference axes are  $z_R$  in the rotor axis downward,  $x_R$  orthogonal to  $z_R$  forward in the helicopter body ( $x_H, z_H$ ) plane for a tandem or main rotor, or in the helicopter body ( $x_H, y_H$ ) plane for a tail rotor, and  $y_R$  completing ( $x_R, y_R, z_R$ ) as an orthonormal positively oriented base (see figure 4).

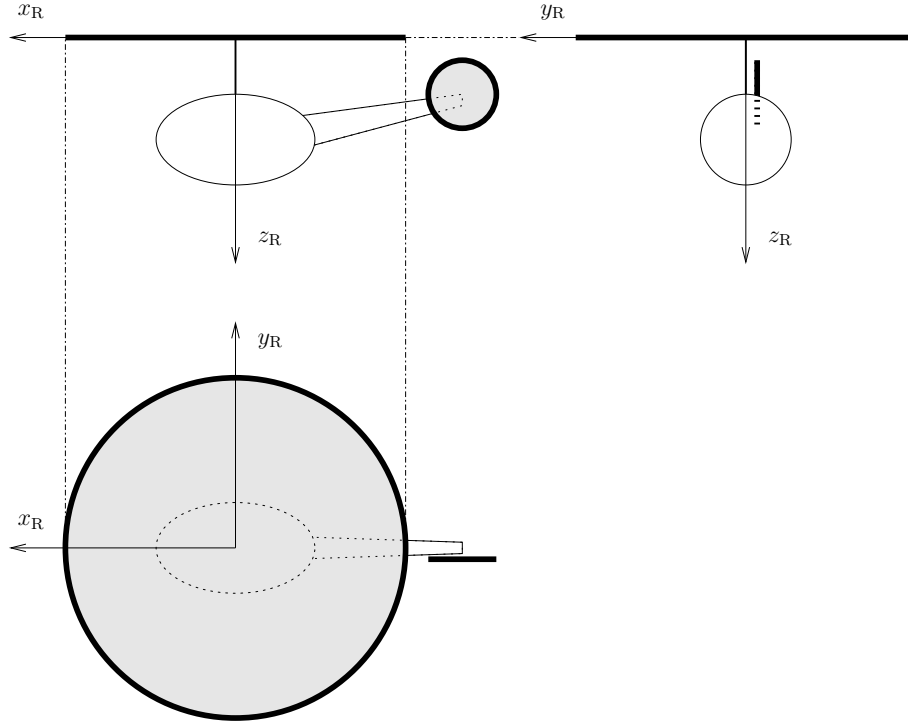


Figure 4. Rotor axis definition

The azimuth angle  $\psi$  is 0 forward in the  $x_R$  axis, and  $\frac{\pi}{2}$  to the right in the  $y_R$  axis; the rotational speed  $\Omega = \dot{\psi}$  is positive when it is a right-hand rotation about  $z_R$  (see figure 5).

Flapping  $\beta$  is positive upwards, and lagging  $\varsigma$  is positive when it makes the blade advance<sup>f</sup> (see figure 5).

Rotor blade axes ( $x_B, y_B, z_B$ ) match rotor axes ( $x_R, y_R, z_R$ ) when  $\psi = \beta = \varsigma = 0$ , and travel with the physical blade through its rotation, flapping, and lagging. Rotor virtual blade axes ( $x_{B'}, y_{B'}, z_{B'}$ ) match rotor blade axes ( $x_B, y_B, z_B$ ) for the same value of  $\psi$  when  $\beta = \varsigma = 0$ , (they match the physical blade with its rotation, but not with its flapping or lagging).

<sup>f</sup>This sign convention is opposite to the usual one, and was chosen so for coherence; with this sign convention,  $\varsigma$  is a “leading” rather than a “lagging”.



local Mach number  $M$ , local air density  $\rho$ , and rotor icing conditions. A typical parameter is blade element offset  $r$  (distance from rotor hub), that makes it possible to take into account non-uniform cord, airfoil distribution along the blade, and blade tip loss.

### II.B.5. Blade inertial forces and moments

The inertial forces and moments generated by the blades resulting from their motion in rotation, flapping, lagging, and as they follow the helicopter motion, are taken into account in the same way as in traditional models, except that they are applied to the virtual blades instead of the physical blades, and are thus used in the computation of the evolution of the Fourier coefficients for flapping and lagging. The inertial forces and moments considered are:

- rotor acceleration,
- flapping and lagging centrifugal forces and moments,
- flapping and lagging acceleration,
- gyroscopic moments,
- linear and angular accelerations due to helicopter acceleration.

### II.B.6. Ground effect

Ground effect is modelled by its reducing effect on inflow velocity. The FABES model supports any ground effect model. For illustration purposes, Cheeseman's theory will be used here.<sup>2</sup> The global form of Cheeseman's theory states that induced flow velocity is modified by the proximity to the ground by the formula

$$\frac{(v_i)_{\text{IGE}}}{(v_i)_{\text{OGE}}} = k_G(z_G) = 1 - \left( \frac{R}{4z_G} \right)^2 \quad (36)$$

where  $(v_i)_{\text{OGE}}$  is the induced flow speed out of ground effect (i.e., that predicted by the inflow field model),  $(v_i)_{\text{IGE}}$  is the induced flow speed in ground effect,  $k_G$  is the ground effect factor, and  $z_G$  is the distance to the ground. The value of  $k_G$  is limited to be non-negative, in order to avoid unrepresentative results.

Equation (36) can be adapted easily for the FABES model, and thus give results for non-uniform ground effect (i.e., the situation in which the difference of distance to the ground along the rotor axis varies considerably through the rotor disk) as follows. For each blade element,  $z_G$  is computed as the distance from the blade element to the nearest obstacle along  $z_R$ . Then, the value resulting from the right-hand side of equation (28) is modified by multiplying it by  $k_G(z_G)$ .

### II.B.7. Interferences

Having the inflow field defined per blade element results in two desirable features for interference modeling:

- the inflow field is available in a high level of detail, resulting in a rotor generated flow that can accurately influence other rotors or other aerodynamic elements;
- if a detailed model of interference from other rotors or other aerodynamic elements is available, it can be taken advantage of by the free-form inflow model.

The first aspect produces a description of an inflow field attached to the rotor that can be introduced in other models. The second aspect enables the parameterization to specify how other models influence in each blade element at each time during the simulation. The FABES model supports any kind of interference model, for both output and input of inflow velocities.

Figures 6 and 7 depict very simple interference models. Figure 6 shows how the main rotor inflow field can be introduced in the tail rotor, rudder, and stabilator (if present) models, by simply considering that the inflow field extends along a cylinder parallel to the mean inflow velocity, and that it adds to the inflow velocity at the tail rotor and to the local wind at the rudder and stabilator. Figure 7 shows how both rotors in a tandem configuration affect each other, by adding to each blade element inflow velocity the one resulting from the overlapping blade element in the other rotor, if any, following the direction of mean inflow velocity. The modifications to the inflow field can also take into account interference from other aircraft.

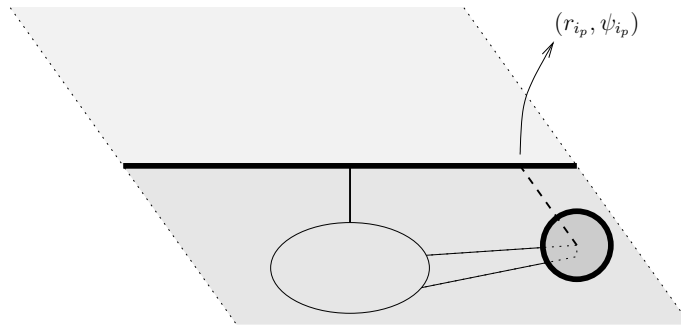


Figure 6. Simple interference from the main rotor to the tail rotor, rudder, and stabilator

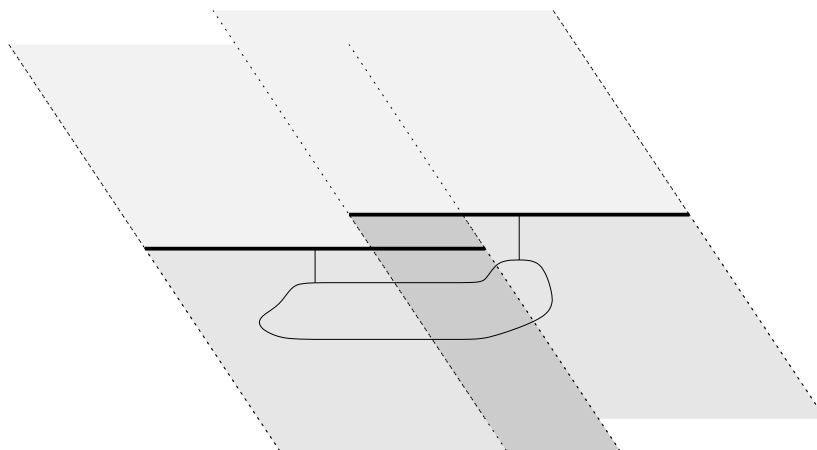


Figure 7. Simple rotor-to-rotor interference in a tandem configuration

## III. Model implementation

### III.A. Simulation environment

The FABES model was implemented as a part of a complete environment for real-time simulation. The goals of the environment were natural expression and implementation of physical models, easy validability, testability, and maintainability. It was designed and implemented following object-oriented techniques, in the C++ language.

This section gives a brief outline of the simulation environment features that are important to understand section III.B, which details the implementation of the FABES model.

#### III.A.1. Simulation model

In the simulation environment, a *model* is a class that shows a state that must evolve with time. The class `Model` offers an implementation for that concept, and adds to it recursive subscription, i.e. the possibility of a model to request to be executed (i.e. to have its state updated) just before a “parent” model is. This allows for tree-like structuring of models, paralleling functional structure or dependency.

#### III.A.2. Numerical integration

The numerical integration module implements numerical integration of ordinary differential equations with respect to time. An abstract base class `Integrator` defines the interface for the integration schemes, with methods for initializing, integrating (by passing a new value for the integrand), and consulting the current value.

Zeroth, first, and second-order linear integration schemes are also implemented, with explicit and implicit variants. These schemes can be instantiated and passed to any model that requires numerical integration.

#### III.A.3. Geometry

The geometry module provides classes to represent three-dimensional vectors (class `Vector`), three-by-three matrices (class `Matrix`), and unit-norm quaternions (class `Quaternion`), used to represent rotations in three-dimensional space. Vectors can represent mathematical vectors or points. Matrices can represent linear transformations in three-dimensional space. Unit-norm quaternions can represent rotations or attitudes. All usual constructors, operations, conversions, and stream operators, are available.

Quaternions can be composed as rotations, and can be used to rotate vectors. A quaternion can be converted to the matrix that perform the same rotation. The three classes feature conversion to and from interpretations based in Euler angles.

#### III.A.4. Kinematics

Zeroth-order kinematics is implemented by the class `Reference_0`. A `Reference_0` object represents a position and attitude with respect to another `Reference_0`, or with respect to the absolute reference frame. The position and attitude of a `Reference_0` can be updated, so that it can represent a moving frame as well as one that is fixed with respect to its definition frame.

The class `Reference_0` allows for easy and efficient conversion of vectors, points, and attitudes between any two reference frames. Intermediate results for such conversions are cached for execution time efficiency<sup>§</sup>.

Analogously, first-order kinematics is implemented by the class `Reference_1`, which is derived from `Reference_0`. A `Reference_1` object represents a moving `Reference_0` whose linear and angular velocities are specified. Like `Reference_0` it can be composed, and its specification can be updated.

The class `Reference_1`, in addition to all the capabilities of `Reference_0`, can convert, between any two reference frames, linear velocities at given points, and angular velocities.

#### III.A.5. Rigid solid

The rigid solid module implements classes and operations to integrate of the equations of motion of a rigid solid, and to access its state.

---

<sup>§</sup>This is very important for the implementation of the FABES model, which uses reference frame composition extensively.

Solid mass distributions are represented by the class `SolidMass`, which features useful constructors (point mass, centered mass with specified inertia), arithmetics (to scale and combine mass distributions), consultors (mass, center of gravity, inertia), operations (displacement and reference to a given point), and stream operators.

Similarly, the class `SolidAction` represents a pair of force and moment, and it features useful constructors (pure force, pure moment, combined force and moment), arithmetics (to scale and combine), consultors (force, moment), operations (displacement and reference to a given point), and stream operators.

Finally, the class `Solid` represents a rigid solid subject to the equations of rigid solid dynamics. It is derived from `Model` (since it has evolving state) and `Reference_1` (so that it can be used as a reference frame, especially for other `Reference_1` objects to be defined with respect to it; for instance, the rotor reference frame is expressed with respect to the helicopter reference frame, which is implemented as a `Solid`). It features subscriptions for `SolidMass` and `SolidAction`. The subscription for `SolidMass` allows the simulation program to have separate contributions to the total mass distribution of a helicopter (empty weight, varying fuel, internal loads, crew, et cætera). The subscription for `SolidAction` makes it easy to implement models that can contribute independently external forces and moments.

The simulation model for `Rigid` implements the equations of motion for a rigid solid: it adds up all contributions to mass distribution, then all contributions to external forces and moments, and uses them in the numerical integration of the ordinary differential equations that govern the dynamics of a rigid solid. The four numerical integration schemes that are used can be specified at construction time.

### III.A.6. *Functors*

A *functor* is an object that represents a mathematical function. A function of  $i$  arguments is represented by the class `Functor_` $i$ . For instance, the class `Functor_2` represents functions with two arguments.

Each functor class is defined as an abstract class, templated on its return type, and on the type of each of its arguments. The function implemented by each functor is declared as an abstract `const` method in the base functor class, with the appropriate signature. A concrete implementation for a functor must derive from the base functor class, and implement that method.

Functors with zero arguments represent values that may evolve with time if needed (e.g. functions of time, or values that depend on the state of some system). Functors with one or more arguments represent functions that may evolve with time if needed (e.g., the lift coefficient function  $c_l$  described in section III.B.3 may evolve with rotor icing condition). The state and the function implemented by a functor cannot change as a result of its evaluation, but they can evolve with time by any other means, or depend on other external values or functions.

Some useful functor implementations are defined by the functor module, that repeat the value of a variable, the value of a consultor of a given object, and a method of an object with the appropriate signature. Other implementations are given by other modules.

### III.A.7. *Filters*

Filters are represented by the class `Filter`, which is an abstract class that defines an interface to initialize, filter a value, and consult the current output value of the filter. Some concrete filter implementations are defined by the filter module: filters defined by their Laplace transform that are implemented using Tustin's formula, PID and PID-plus-second-derivative filters implemented using the "Modified Matched Pole-Zero" method,<sup>3</sup> filters implemented using an exact discrete operator, saturating filters, pure delay filters. Other implementations are given by other modules.

## III.B. Implementation of FABES proper

### III.B.1. *Philosophy for rotor parameterization*

The FABES generic parameterizable rotor model is implemented as a class `Rotor` derived from the class `Model`, since it is a system with dynamics (its state evolves with time). The parameterization is achieved by providing the class constructor with arguments to define the particular magnitudes or conditions that can affect the object that represents a particular rotor.

**CONSTANTS** Whenever a parameter can be defined as a simple constant magnitude, e.g. a number, or a vector, or an attitude, it is passed to the class constructor as a single object of the appropriate type. This is the case, for instance, for the rotor radius, the number of blades, or the number of azimuth stations of the azimuth-wise discretization, which are not expected to change in run-time for a particular rotor instance.

**NON-CONSTANT VALUES** Whenever a parameter can be defined as a simple magnitude that changes during the simulation, it is passed either as a consultant (an instance of an abstract class which only has methods defined as `const`), or as a `Functor_0` (which is a special kind of consultant, with only one `const` method defined `operator()()`, which takes no arguments and returns a value of a specified type; see section III.A.6). For instance, local air, gravity, and obstacle properties are defined as consultants, whereas  $\Omega$ , and swash-plate position and attitude are passed as `Functor_0` objects that return scalars. A `Functor_0` object is an instance of a class derived from the polymorphic abstract class `Functor_0`, which requires definition of the abstract method `operator()()` with no arguments and a specified return type.

**FUNCTIONS** Parameters that represent functions or distributions are defined as functors with an appropriate number of arguments (see section III.A.6). For instance, blade element lift, drag, and pitch moment coefficients ( $c_l$ ,  $c_d$ , and  $c_m$  respectively) are passed as functions of  $\alpha_e$ ,  $M$ , and  $r$ , implemented by `Functor_3` objects that return scalars and accept three scalar arguments, and blade twist is passed as a function of  $r$  implemented by a `Functor_1` object. As in the case of `Functor_0`, a `Functor_1` object is an instance of a class derived from the polymorphic abstract class `Functor_1`, which requires definition of the abstract method `operator()()` with one argument of a specified type, and a specified return type. The same applies to functors with more arguments (`Functor_2`, `Functor_3`, et cætera).

Sometimes a function that is represented by a functor with a certain number of arguments may need to be extended to depend on a greater number of arguments if a sufficiently accurate model has to be implemented. The fact that the functors are implemented by abstract classes make it easy to implement such extensions while maintaining full backward compatibility. See section III.B.3 for a realistic example of how and when this can be done.

**FILTERS** Whenever a parameter can be defined as a filter with respect to time, it is passed as a `Filter` object (see section III.A.7).

**IMPLEMENTATIONS FOR USUAL PARAMETERS** Very often, a given parameter takes one of a few values or forms. For instance, the function represented by the `Functor_2` object that parameterizes blade pitch as a function of azimuth and flapping usually takes the form

$$\theta(\psi, \beta) = \theta_{\text{coll}} + \theta_{\text{long}} \cos(\psi + \Delta\psi) + \theta_{\text{lat}} \sin(\psi + \Delta\psi) - \tan \delta_3 \beta \quad (37)$$

where  $\theta_{\text{coll}}$  corresponds to the collective control,  $\theta_{\text{long}}$  corresponds to the longitudinal control,  $\theta_{\text{lat}}$  corresponds to the lateral control,  $\Delta\psi$  is the control azimuth phase, and  $\delta_3$  is the flapping-to-pitch coupling angle.

Then, concrete implementations for the usual cases can be provided. For instance, for the blade pitch function, a concrete parameterizable implementation for `Functor_2` can be provided that accepts at construction time the parameters  $\theta_{\text{coll}}$ ,  $\theta_{\text{long}}$ ,  $\theta_{\text{lat}}$ , and  $\Delta\psi$ , and mechanizes equation (37). Thus, most of the usual parameterization cases can be covered without requiring the client code to implement new constants, functions, or class implementations.

### III.B.2. Rotor kinematics

The whole rotor kinematics is handled by the simulation environment through its class `Reference_1`, which is detailed in section III.A.4. This class represents the first-order kinematics of a moving reference frame, and implements the concept of first-order relative kinematics; it offers methods to

- express first-order kinematics of reference frames relative to other reference frames;
- express first-order kinematics of reference frames in absolute coordinates;
- convert first-order kinematic magnitudes (vectors, positions, attitudes, linear and angular velocities) between different reference frames.

Using this class it is straightforward to implement the whole rotor kinematics, having each step in the following chain of relative frames be an instance of `Reference_1` defined relative to the previous one:

- the helicopter body frame is a reference frame whose evolution is considered out of the rotor model<sup>h</sup>;
- the rotor frame is expressed relative to the helicopter body frame, with a constant displacement and orientation;
- the virtual blade frame is expressed relative to the rotor frame, with a fixed rotation (the virtual blade discretization azimuth) but variable, *usually non-zero* angular velocity  $\Omega \cdot \mathbf{k}_R$ ;
- an intermediate frame takes into account blade pitch by defining a rotation relative to the virtual blade frame;
- an intermediate frame takes into account flapping by defining a rotation and an angular velocity relative to the previous frame; the rotation and the angular velocity are computed from the flapping Fourier coefficients;
- an intermediate frame takes into account lagging by defining a rotation and an angular velocity relative to the previous frame; the rotation and the angular velocity are computed from the lagging Fourier coefficients;
- the blade element frame is expressed relative to the previous frame, with a constant displacement along the blade span, and a constant rotation corresponding to the blade element twist with respect to the blade root.

Having these frames, many magnitudes can be expressed simply as conversions between frames; these are the most important ones:

- weight is converted from its reference frame (local horizon, or absolute reference frame) to blade frame; other external forces and moments applied to the blades can be treated in the same way;
- rotor and blade kinematics with respect to an inertial reference frame can be obtained from the appropriate relative frames, and converted *as vectors* to rotor frame for easier computation of inertial forces and moments (centrifugal forces, gyroscopic moments, et cætera);
- the local perpendicular to the rotor disk can be converted to the reference frame in which the obstacle data base is expressed, in order to obtain distance to obstacle for local ground effect;
- local airspeed can be obtained by converting local wind speed as a vector (including blade element inflow, and additional contributions from other sources, such as interference with another rotor) to blade element frame;
- forces and moments generated by the virtual blades and virtual blade elements are converted to the rotor reference frame, added together, and multiplied by the factor  $b/n_a$ , as explained in section II.A.5.

Here is how magnitudes that are considered for rotor kinematics are supplied to the rotor model class through its constructor:

- $\Omega$  is supplied as a `Functor_0` object, usually determined by the transmission system, that adds up all torque contributions from rotors, power turbines, accessory boxes, friction losses, et cætera, and integrates the rotational dynamic equations;
- flapping and lagging limitations are supplied as `Functor_0` objects that can depend on any magnitude, whether external or local to the rotor;

---

<sup>h</sup>In the usual case, the helicopter body frame is implemented as a rigid solid (class `RigidSolid`) subject to the dynamics resulting from all forces and moments applied to it and its mass properties up to second order (class `SolidMass`); the class `RigidSolid` derives from `Reference_1`.

- blade twist is supplied as a `Functor_1` object that takes as its argument the blade element offset  $r$ , but can depend on any other magnitude, whether external or local to the rotor, or dependent on  $r$ ; the implementation can be trivially extended to depend also on  $\psi$ , if a sufficiently accurate aeroelastic model is available that takes into account per-azimuth blade load and cannot be implemented in terms of blade pitch increments;
- blade pitch is supplied as a `Functor_2` object that takes as its arguments blade azimuth  $\psi$  and blade flapping  $\beta$ , but typically depends at least on swash plate state, and possibly other external or local magnitudes.

### III.B.3. Blade element aerodynamics characterization

As explained in section II.B.4, the characterization of blade element aerodynamics is implemented in such a way that the effect of all important local conditions and blade element parameters can be implemented. An approach based on aerodynamic non-dimensional coefficients was chosen, although it could easily be generalized to cope with more complex airfoil models. The functions that yield the aerodynamic coefficients are of the form

$$c_l = c_l(\alpha_e, M, r/R) \quad (38)$$

$$c_d = c_d(\alpha_e, M, r/R) \quad (39)$$

$$c_m = c_m(\alpha_e, M, r/R) \quad (40)$$

where  $R$  is the rotor radius. The dependency on  $r$  allows the implementation of blades with varying profile or cord along their span, or blade tip losses.

These functions are implemented by `Functor_3` objects. These functor objects can additionally depend on other external factors, such as rotor icing condition, or partial rotor damage. The `Functor_3` objects are passed to the rotor model constructor as part of its parameterization.

The fact that the implementation of these functions uses the polymorphic family of functor classes enables extension to more local dependencies while maintaining backward compatibility: if a sufficiently detailed characterization of the airfoils were available, with dependency on  $\dot{\alpha}_e$  and  $\beta_e$  (blade element local side-slip angle),

$$c_l = c_l(\alpha_e, M, r/R, \dot{\alpha}_e, \beta_e) \quad (41)$$

$$c_d = c_d(\alpha_e, M, r/R, \dot{\alpha}_e, \beta_e) \quad (42)$$

$$c_m = c_m(\alpha_e, M, r/R, \dot{\alpha}_e, \beta_e) \quad (43)$$

the rotor constructor could be modified to accept `Functor_5` objects instead of `Functor_3` objects, but an overloaded constructor could still accept the old `Functor_3` objects, and wrap them into a `Functor_5` object that discards the last two arguments<sup>1</sup>.

### III.B.4. Integration with respect to azimuth

Since typically few azimuth discretization stations are required, whenever a numerical integration with respect to azimuth is needed, a simple rectangle rule is used which, given the cyclic character of the functions to integrate, yields the same result as a trapezoidal rule; thus, for instance, equation (20) is mechanized as

$$\begin{aligned} Q_{a_i} &= \int_{-\pi}^{+\pi} \Delta M(\psi) \cos i\psi \, d\psi \\ &\simeq \sum_{j=1}^{n_a} \Delta M\left(\frac{2\pi}{n_a}j\right) \cos\left(\frac{2\pi}{n_a}j\right) \cdot \frac{2\pi}{n_a} \end{aligned} \quad (44)$$

<sup>1</sup>Another, more general and powerful but less secure option is to implement a chain of specializations in which each *abstract* `Functor_i` is a *concrete* implementation of `Functor_(i + 1)` while adding a new *abstract operator*()() with one less arguments, in terms of which the implementation of `Functor_(i + 1)` is done by discarding its last argument.

### III.B.5. Integration with respect to blade span

Similarly, a rectangle rule is used for numerical integration along the blade span. For instance, when adding up force and moment contributions from the blade elements, forces and moments are simply referred to a common reference frame, then multiplied by the corresponding blade element extent span-wise and added together.

### III.B.6. Inflow model and associated parameterization

The inflow model is represented by a `Functor_4` object that implements  $\mathbf{v}_i^+(\Delta\mathbf{T}, \Delta S, \mathbf{v}_0, \mathbf{v}_i)$  as described in section II.A.4, and in particular by equation (24) for the momentum theory, and a `Filter` object that represents a filter that is applied to  $\mathbf{v}_i^+$  before feeding it back to  $\mathbf{v}_i$ , represented by the product  $G(s) \cdot E(s)$  in equation (28).

Ground effect is represented by a `Functor_2` object that implements  $(\mathbf{v}_i^+)_{\text{IGE}}((\mathbf{v}_i^+)_{\text{OGE}}, z_G)$  as described in section II.B.6. If sufficient detail is available, this function can also handle ground effect models in which the ground effect does not only scale inflow velocity, but also skew it as a result of ground surface orientation.

Interference output is implemented by having the `Rotor` class have a consultant that is passed a location in rotor axes, and returns the corresponding inflow velocity as a vector. Interference input is implemented by a `Functor_2` object that accepts an inflow velocity before interference and a location in rotor axes, and returns the inflow velocity after interference. Mutual rotor interference is then handled by having each rotor `Functor_2` object consult the inflow velocity of the other rotor at the appropriate location.

## IV. Ongoing development

### IV.A. Growing library of variant parts for the model

The way in which the parameterization of the model is implemented through values and functors makes it easy to construct a library of parameters that handle the most common cases. This library can grow with each new implementation required for a particular parameter of a given parameterization, if it is not too specific. For instance, the available library only includes one ground effect model right now, but as soon as any other model is required for a given parameterization, its implementation can be included in the library. Other candidates for such a library combine or modify existing functors or values to model other effects (e.g., available models for element aerodynamic characteristics can be modified homogeneously to take into account icing conditions).

Clear opportunities for growing in this direction are:

- a set of NACA and other common airfoils;
- theoretical airfoils;
- a set of general models to combine or modify blade element aerodynamic characteristics, including
  - analytical extension of airfoils to cover the whole  $[-\pi, +\pi]$  range,
  - icing conditions,
  - blade tip losses,
  - rotor damage;
- a set of ground effect models;
- a set of inflow field models;
- various kinds of fuselage and rotor interference.

Like for any object-oriented project, as this library grows, the task of parameterizing the model will shift from coding the parameters to selecting, modifying and combining them.

## IV.B. Validation of the model

A simulation model can be validated at several levels: theoretical, generic, parameterized. Each of these levels of validation gives a different kind of guarantee about the generic and the parameterized model: the theoretical validation ensures valid implementation, the validation of the generic model ensures accurate representation of the underlying physical process, the validation of parameterized models ensures that the particular parameterizations are correct and that the generic model is realistic and flexible enough in its parameterization capabilities.

Since FABES is a *generic* model, its validation is always an ongoing effort, but each piece of validation adds to the previous ones to increase confidence. This section describes briefly what has been done and what is being done in this area, and gives hints as to what remains to be done.

### IV.B.1. Validation against theory

The FABES model was tested, during its implementation, against theoretical results (using both disk and blade element theoretical models), and it was verified that it reproduced them to the expected extent. Many results were checked, such as global steady-state characteristics, quick convergence of the Fourier coefficients under dynamic control inputs, expected values for the Fourier coefficients for given control inputs, performances and inflow field in vertical climb and descent and cruise flight, ground effect, and vibrations.

### IV.B.2. Validation of the generic model

Limited comparison between the results of the FABES model and wind-tunnel test data has been performed to date. The criteria and setup for this kind of comparison is obtained from such a controlled environment that it can be considered that the generic model itself is validated, at least in the relevant features. We plan to extend testing in this direction, in order to find potential areas for improvements, faster and more reliable tuning procedures, and to assess the limitations and attainable performances of the model.

### IV.B.3. Validation of the parameterized model

The helicopter flight models delivered to the Spanish Army underwent extensive testing by engineers, instructors, and test pilots. After initial implementation errors were corrected, parameterizations from available data showed promising results; the parameterization of the rotor model was fine-tuned according to data and pilots' comments. Resulting performances and handling qualities in all possible conditions were compared with aircraft manuals, flight test data, and pilot experience and expectations, and found to match accurately. The conditions and maneuvers that were tested included take-off and landing, hover, vertical climb and descent, rearward and side-ward flight, static and dynamic stability in all axes in hover and slow and fast cruise, turns, maximum airspeed, auto-rotation, taxi and braking, and highly dynamic maneuvers. Tests were evaluated both with and without stability augmentation, and in all available atmospheric conditions (standard and non-standard pressure and temperatures, calm air, steady wind, gusts, turbulence, icing conditions). All the available malfunctions that affected flight were evaluated. Flight in uniform and non-uniform ground effect was tested on runways, on ships, over buildings, and while landing on heliports on hospital roofs.

The FABES model was also tested in a small, medium-fidelity fixed-based simulator for the Spanish Air Force, where the emphasis was on delivering a suitable flight model fast. The rotor model proved to be easy to tune, even using very scarce data, because its parameterization deals with physical magnitudes easy to understand and easy to relate to measurable quantities. The flexibility of the parameterization put in place in the implementation also contributed greatly to the success of this project.

Two parameterizations of the FABES model for medium-lift conventional-rotor helicopters are currently in the last phases of fine tuning for validation by JAR-STD 1H level D (the equivalent of FAA Advisory Circular AC 120 63) against flight test data, in the scope of a high-fidelity helicopter simulator program. Validation by JAR-STD 1H level D is very demanding on flight model and flight test data quality, and has very narrow tolerances. It covers extensively all the important characteristics and maneuvers of the helicopter flight.

## V. Conclusions

Indra Sistemas is using the FABES model successfully for a growing number of simulators, and applying it to all the rotors in the simulated helicopters, whether tandem or main, *and even for tail rotors*. The FABES model accurately reproduces the simulated rotors, and satisfies all the requirements encountered so far, both in its capabilities and in its characteristics. Its generic, parameterizable implementation and the simulation environment in which it is included have been designed to be flexible, maintainable, validable, and reusable in the most fundamental sense: *not requiring* any change to the generic classes or to the fundamental models, while *allowing* for growth and improvement both in the generic part and in the parameter concrete implementations.

Ongoing validation will increase confidence in the model, and give important information about its scope, possible improvements, and appropriate and efficient tuning procedures.

## References

- <sup>1</sup>Prouty, R. W., *Helicopter Performance, Stability, and Control*, Robert E. Krieger Publishing Company, 1990.
- <sup>2</sup>Leishman, J. G., *Principles of Helicopter Aerodynamics*, Cambridge Aerospace Series, 2000.
- <sup>3</sup>Stevens, B. L. and Lewis, F. L., *Aircraft Control and Simulation*, Wiley-Interscience, 2003.