

Characterisation and Extension of a Rigid Body Dynamics Solver Coupled with OpenFOAM® for Flight Performance Analysis of Flapping-wing Drones

Romain Poletti^{1,2,*}, Emanuele Bombardi^{3,4}, Lilla Koloszar¹, Miguel Mendez^{1,3,5}, and Joris Degroote^{2,6}

¹von Karman Institute for Fluid Dynamics, Waterloosesteenweg 72, Sint-Genesius-Rode, Belgium
Email address: romain.poletti@vki.ac.be

²Department of Electromechanical, Systems and Metal Engineering, Ghent University, Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium

³Aero-Thermo-Mechanics Laboratory, École Polytechnique de Bruxelles, Université Libre de Bruxelles, Av. Franklin Roosevelt 50, Brussels, 1050, Belgium

⁴Brussels Institute for Thermal-Fluid Systems and Clean Energy (BRITE), Université Libre de Bruxelles and Vrije Universiteit Brussel, Pleinlaan 2, Brussels, 1050, Belgium

⁵Department of Aerospace Engineering, Universidad Carlos III de Madrid, Av. de la Universidad 30, Leganés, Spain

⁶FlandersMake@UGent, 9000 Gent, Belgium

DOI: <https://doi.org/10.51560/ofj.v6.150>

Results with version(s): OpenFOAM® v2206

Repository: <https://github.com/polettiRomain/eABA>

Abstract. The extraordinary aerial agility of hummingbirds and insects continues to inspire the design of flapping-wing drones. To replicate them, computational fluid dynamics (CFD) simulations must be coupled with rigid body dynamics solvers. While OpenFOAM® provides a general articulated-body algorithm (ABA) for these multiphysics simulations, its current implementation is restricted to passive joints and lacks thorough verification for free-flight applications. This work addresses these limitations by introducing an extension of the ABA to support active joints with prescribed kinematics, allowing the body dynamics and the imposed motions to be solved simultaneously. The imposed motion is implemented in a new *imposedMotion* library that directly prescribes joint positions, velocities, and accelerations. The proposed approach is verified using a hierarchical strategy, including analytical benchmarks, comparison with quasi-steady aerodynamic models, and grid- and time-integration sensitivity analyses. Results demonstrate the robustness of the proposed framework and its ability to capture body–wing inertial coupling effects, contributing to the OpenFOAM® community with an extensible open-source platform for simulating partially actuated multibody systems in unsteady flows.

1. Introduction

Hummingbirds and insects exhibit exceptional flight maneuverability, seamlessly transitioning from stable hovering in wind gusts to rapid escape maneuvers, including upside-down flight [1]. Their remarkable flights have inspired the design of flapping-wing drones, offering significant potential for rescue operations in tight and hazardous spaces. However, a significant performance gap remains between engineered fliers and their biological counterparts. This gap stems from the incomplete characterization of the low-Reynolds, unsteady flow generated by the flapping wings coupled with the drone motion. High-fidelity environments are then required to simulate the unsteady aerodynamic phenomena during the drone flights.

Simulating FWMAV flight requires coupling the drone's equations of motion with an aerodynamic model. Depending on the intended level of fidelity, these aerodynamic loads can be computed using simplified models or computational fluid dynamics (CFD). While simplified models are vastly popular for stability and control-oriented studies [2–5], CFD-based approaches are used when accurate force prediction and detailed flow–structure interactions are required [6–8]. Such CFD-based simulations typically rely on dynamic meshing techniques to accommodate wing motion, including overset grids, immersed boundary methods, and Arbitrary Lagrangian–Eulerian

* Corresponding author

Received: 6 September 2024, Accepted: 10 June 2026, Published: 13 October 2026

formulations [9–14]. While these approaches have been successfully applied to tethered and hovering configurations, fully coupled free-flight simulations remain comparatively scarce, largely due to the high computational cost associated with the need to resolve both complex kinematics and flow features in strongly unsteady regimes.

Partitioned simulations generally rely on a coupling between a flow solver and a rigid-body dynamics (RBD) solver, whereby aerodynamic loads computed from the Navier–Stokes equations are used to advance the drone body motion. Although more advanced coupling strategies exist [15], weakly coupled approaches remain the most common in practice [16, 17], despite limited applicability to highly dynamic flights with strong added-mass effects. For low-dynamic flights, the flow equations are often solved around isolated wings and the resulting aerodynamic forces are applied to the body’s equations of motion [6, 17, 18]. While some studies do model the flow around both the drone body and its wings [19–22], they often neglect inertial coupling by ignoring the wings’ mass and inertia [23, 24]. The equations of motion are then those of a standard aircraft with external aerodynamic forces applied by the wings [25]. Given the large size of the flapping-wing drone investigated in this research, this assumption may not hold [23, 24]. The most complete approach is then to consider the multibody equations of the drone, solving simultaneously for the motion of both the drone body and its wings. Only a few works have coupled this dynamic model with flow solvers using commercial software [26, 27] or in-house developments [28, 29].

Consequently, these contributions offer limited benefit to the open-source community. In many cases, the underlying algorithms are insufficiently documented, making it difficult to assess their applicability or limitations. In this work, we address this gap by developing and thoroughly documenting a high-fidelity simulation environment within OpenFOAM® that integrates fluid and rigid body dynamics solvers.

In OpenFOAM®, two approaches exist to simulate the dynamics of a flapping-wing drone. The first one relies on the *sixDofRigidBodyMotion* library and the second one relies on the *rigidBodyDynamics* library. The six-degrees-of-freedom library solves the Newton–Euler equations for a single rigid body subjected to fluid dynamic loads and has been recently used to model floating structures [30, 31]. However, its single-body formulation does not support articulated systems, which severely bounds its applicability to flapping-wing drones. While the *rigidBodyDynamics* library implements the articulated-body algorithm (ABA) capable of simulating multibody systems, its current formulation is restricted to passive joints, i.e. joints driven by the system dynamics and external loads. Therefore, the library cannot be used to impose prescribed kinematics on the drone wings while simultaneously solving for the body motion.

This work addresses these limitations by extending the articulated-body algorithm in OpenFOAM® to support actively driven joints with prescribed kinematics, as inspired by [32–34]. The proposed extended articulated-body algorithm (eABA) combines forward dynamics for the free motion of the system with inverse dynamics to enforce user-defined joint positions, velocities, and accelerations. A new *imposedMotion* library is introduced to provide modular kinematic definitions without relying on torque-based control. The algorithm is tested in a high-fidelity CFD environment based on the overset method, a moving background grid, and a Large-Eddy Simulation (LES) model to simulate the flow around a drone equipped with rigid, semi-elliptical wings.

This article is organized as follows. Section 2 details the *rigidBodyDynamics* library’s approach to formulating and solving the equations of motion. Section 3 describes the new implementations in the *rigidBodyDynamics* library to actuate a subset of the degrees of freedom from a multibody system. Section 4 presents two test cases: a single-wing drone and a body-wing drone. The first test case is used in section 5 to verify the extended articulated-body algorithm and to perform a few parametric studies. Section 6 explores the dynamics of the open-loop, vertical ascending flight of the body-wing drone. Section 7 discusses the main contributions of this work and suggests further developments.

2. Theoretical background of the rigid body dynamics library

Three main solvers make up the CFD environment that computes the motion of a multibody system subjected to an external flow: (1) a rigid body dynamics solver to solve the equations of motion and predict the body motion, (2) a dynamic mesh solver to adapt the grid according to the body state and (3) a flow solver to compute the forces acting on the bodies. Figure 1 shows these three key components for the original implementation of the *overPimpleDyMFoam* solver within OpenFOAM® v2206. This section describes the governing equations of the rigid body dynamics solver, linking them to the *rigidBodyDynamics* library available in OpenFOAM®. Blocks (2) and (3) are briefly described at the end of the section and the reader is referred to [35, 36] for more details.

2.1. System definition. Before solving the dynamics of the multibody system, that is before the solver enters the first loop in Fig. 1, the *rigidBodyDynamics* library is used to build a multibody system with entries from the *dynamicMeshDict*. Figure 2 illustrates one such multibody system. It is defined as a set of n bodies (also called links and represented as spheres) connected through $n - 1$ joints (also called articulations and represented by rectangles or cylinders [37]) that allow the relative motion between the bodies. The bodies are slightly offset from the joints with a grey bar for clarity purposes. Each body i is connected with one joint to its parent body $p(i)$ and this article considers only tree-like systems, without kinematic loops formed by bodies.

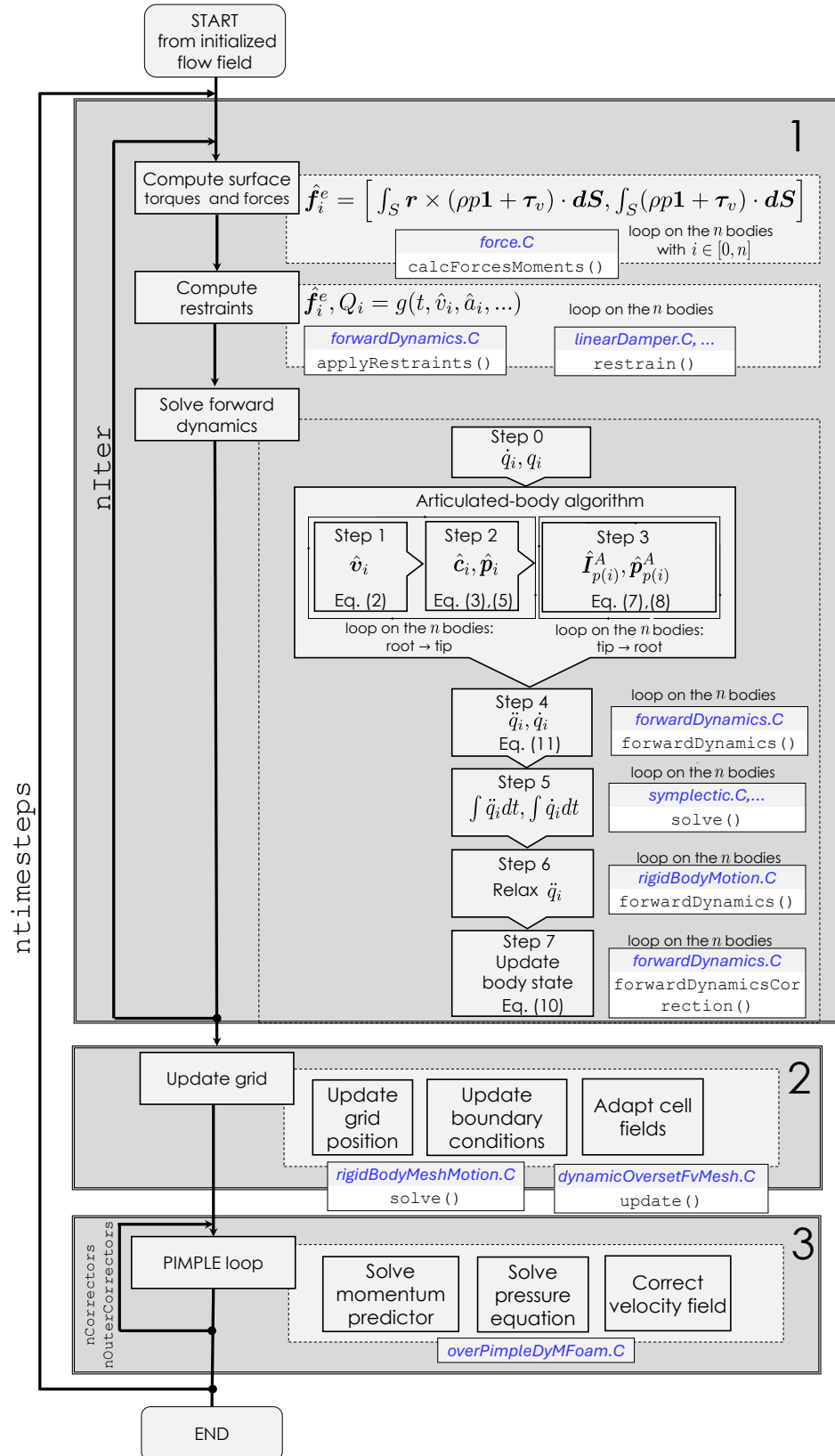


Figure 1. Diagram representing the main operations performed by (1) the rigid body dynamic solver relying on the ABA algorithm, (2) the dynamic mesh method relying on the overset method, and (3) the flow solver using pimple loops.

The first body ($i = 0$) is always a (static) *root* body followed by bodies predefined in the *rigidBodyDynamics* library (*sphere*, *cuboid*, etc). This article focuses on the most general body type, the *rigidbody* that has a point mass m , a center of mass position $\mathbf{x}_c$, an inertia matrix $\mathbf{I}$ (defined with respect to the origin) and a frame of reference defined by its joint.

The joints are either revolute or prismatic, and have a single degree of freedom along their axis of revolution or translation, respectively. Multiple joints can be chained between bodies to form a *composite* joint. In this case, mass-less bodies are inserted in between successive joints to conform with the Newton-Euler formalism of the equations of motion (see next section). The reader is referred to [38] for the complete functionalities of a multibody system in OpenFOAM®.

2.2. Forward dynamics. Based on the initial states of the multibody system and the forces applied to it, the joint accelerations are computed by solving the equations of motion. The main steps for this computation, known as the forward dynamics, are illustrated in the first block of Fig. 1 and rely on the articulated-body algorithm. The reader is referred to [34, 39] for a comprehensive overview of the mathematical framework. Here, we briefly recall the seven key steps to provide foundations for the implementations discussed in section 3.

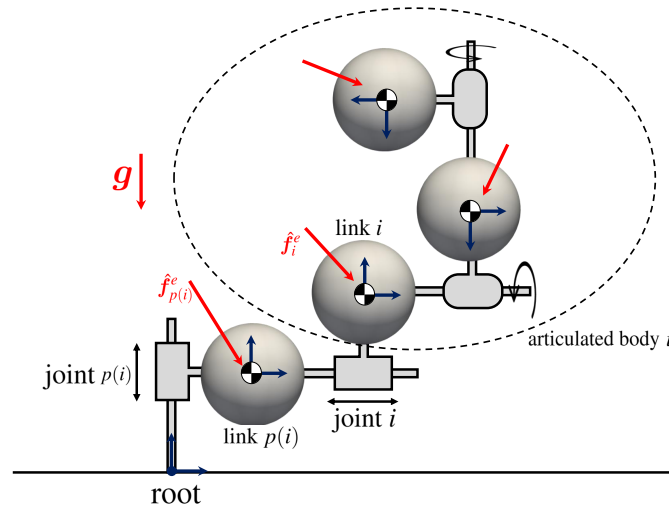


Figure 2. Schematic of the kinematic tree of a multibody system with four links and four joints, illustrating the articulated body concept.

Step 0: Computation of the external and internal forces and torques. At the beginning of each time step, a resultant torque and force vector is computed for each link i based on the pressure and shear stress that the flow applies on its surface. Those torques and forces are called **external** and they are gathered into a **spatial** vector $\hat{\mathbf{f}}_i^e \in \mathbb{R}^{6 \times 1}$ that stacks their three components:

$$\hat{\mathbf{f}}_i^e = \left[\int_S \mathbf{r} \times (\rho p \mathbf{1} + \boldsymbol{\tau}_v) \cdot d\mathbf{S}, \int_S (\rho p \mathbf{1} + \boldsymbol{\tau}_v) \cdot d\mathbf{S} \right], \quad (1)$$

where the hat notation indicates a spatial vector using Plücker coordinates and bases. p is the kinematic pressure, $\mathbf{1}$ is the identity tensor, $\boldsymbol{\tau}_v$ is the shear stress tensor (including fluid density multiplication), ρ is the fluid density, $\mathbf{r}$ is the centre position of the body patch faces taken with respect to the origin, $d\mathbf{S}$ is the area of the body patch faces multiplied by its unitary normal vector, and S is the body surface. The forces are applied to the body's center of gravity and defined in the inertial frame (static frame).

The spatial external force may be complemented by user-defined restraint forces $\hat{\mathbf{f}}_i^p$. For example, the user can attach a spring or a damper to a body to generate additional external forces. Similarly, the restraint library can be used to apply an **internal** torque, Q , along the joint axis. For example, the user could model articulations with friction.

Step 1: Computation of the link velocities. The ABA algorithm's first step aims to compute each link's velocity from the multibody system. The velocity of each link $\hat{\mathbf{v}}_i$ is defined by the velocity of the parent link $\hat{\mathbf{v}}_{p(i)}$ and the joint velocity $\dot{q}_i$, which models the relative velocity between the two bodies:

$$\hat{\mathbf{v}}_i = {}_i\hat{\mathbf{X}}_{p(i)} \hat{\mathbf{v}}_{p(i)} + \hat{\boldsymbol{\Phi}}_i \dot{q}_i, \quad (2)$$

where $\hat{\mathbf{v}}_i$ is a spatial vector with angular velocities in the first three entries and linear velocities in the last three. The velocities are expressed in the link i frame and ${}_i\hat{\mathbf{X}}_{p(i)} \in \mathbb{R}^{6 \times 6}$ is a spatial transformation matrix from the parent to the current link frame, $\hat{\Phi}_i \in \mathbb{R}^{6 \times 1}$ is a spatial vector that converts the joint velocity $\dot{q} \in \mathbb{R}$ into an angular and linear velocity for the link.

Equation (2) defines the first loop of the ABA algorithm: each link velocity is recursively defined spanning the multibody system from the root towards the tip body.

Step 2: Computation of the velocity-product acceleration, isolated inertia, and isolated bias forces. Step 2 involves pre-computing three link variables used in step 3. Firstly, the velocity-product acceleration is defined based on the joint and link velocities (step 1):

$$\hat{\mathbf{c}}_i = \hat{\mathbf{v}}_i \times \hat{\Phi}_i \dot{q}_i. \quad (3)$$

Secondly, the (constant) spatial isolated inertia matrix $\hat{\mathbf{I}}_i$ is defined with the mass and inertia matrix of the link as:

$$\hat{\mathbf{I}}_i = \begin{bmatrix} \mathbf{I}_i^c + m_i \mathbf{S}^T(\mathbf{x}_c) \mathbf{S}(\mathbf{x}_c) & m_i \mathbf{S}(\mathbf{x}_c) \\ m_i \mathbf{S}^T(\mathbf{x}_c) & m_i \mathbf{1} \end{bmatrix}, \quad (4)$$

where m_i is the link mass, $\mathbf{I}_i^c$ is the moment of inertia tensor about the center of mass and $\mathbf{S}(\mathbf{x}_c)$ is the skew matrix for the centre of mass position $\mathbf{x}_c$.

Thirdly, the isolated bias force gathers all the terms that are not a function of the link acceleration in the equation of motion:

$$\hat{\mathbf{p}}_i = \hat{\mathbf{v}}_i \times \hat{\mathbf{I}}_i \hat{\mathbf{v}}_i - {}_i\hat{\mathbf{X}}_0^{-T} \hat{\mathbf{f}}_i^e. \quad (5)$$

The first term is the inertia term due to the frame rotation, and the second term is the spatial vector of external forces (step 0) transformed in the link frame with the spatial matrix ${}_i\hat{\mathbf{X}}_0^{-T}$, with $^{-T}$ denoting the inverse of the transpose of a matrix.

This second step is performed during the first loop defined in step 1.

Step 3: Computation of the articulated inertia and bias forces. The Newton-Euler equations for each parent link $p(i)$ formulated at the center of mass can be written as (see [34]):

$$\hat{\mathbf{f}}_{p(i)} = \hat{\mathbf{I}}_{p(i)}^A \hat{\mathbf{a}}_{p(i)} + \hat{\mathbf{p}}_{p(i)}^A, \quad (6)$$

where $\hat{\mathbf{f}}_{p(i)}$ is the spatial force applied at the joint and transformed at the center of mass of a parent body. $\hat{\mathbf{I}}_{p(i)}^A$ and $\hat{\mathbf{p}}_{p(i)}^A$ are the inertia and bias forces of the articulated body which is the subtree that contains all the links from the body i to the tip body n (Fig. 2):

$$\hat{\mathbf{I}}_{p(i)}^A = \hat{\mathbf{I}}_{p(i)} + {}_{p(i)}\hat{\mathbf{X}}_i \left(\hat{\mathbf{I}}_i^A - \frac{\hat{\mathbf{I}}_i^A \hat{\Phi}_i \hat{\Phi}_i^T \hat{\mathbf{I}}_i^A}{\hat{\Phi}_i^T \hat{\mathbf{I}}_i^A \hat{\Phi}_i} \right) {}_i\hat{\mathbf{X}}_{p(i)}, \quad (7)$$

$$\hat{\mathbf{p}}_{p(i)}^A = \hat{\mathbf{p}}_{p(i)} + {}_{p(i)}\hat{\mathbf{X}}_i \left(\hat{\mathbf{p}}_i^A + \hat{\mathbf{I}}_i^A \hat{\mathbf{c}}_i + \frac{\hat{\mathbf{I}}_i^A \hat{\Phi}_i \left[Q_i - \hat{\Phi}_i^T (\hat{\mathbf{p}}_i^A + \hat{\mathbf{I}}_i^A \hat{\mathbf{c}}_i) \right]}{\hat{\Phi}_i^T \hat{\mathbf{I}}_i^A \hat{\Phi}_i} \right). \quad (8)$$

The articulated bias force depends on Q_i , the magnitude of the spatial joint force defined as:

$$Q_i = \hat{\Phi}_i^T \hat{\mathbf{f}}_i. \quad (9)$$

For an unconstrained joint, Q_i is zero so that the joint velocity $\dot{q}_i$ is only defined by the multibody system dynamics. The joint force can also be user-imposed in step 0.

Equations (7) and (8) are defined recursively based on the inertia and bias force of the next link from the kinematic chain. The ABA algorithm starts computing the articulated inertia of the tip body for which $\hat{\mathbf{I}}_i^A = \hat{\mathbf{I}}_i$ and then propagates it towards the parent bodies. Step 3 consists then in a tip-to-root loop.

Step 4: Computation of the joint and link acceleration. The time derivation of Eqn. (2) results in the following expression for the acceleration of link i :

$$\hat{\mathbf{a}}_i = {}_i\hat{\mathbf{X}}_{p(i)} \hat{\mathbf{a}}_{p(i)} + \hat{\mathbf{c}}_i + \hat{\Phi}_i \ddot{q}_i, \quad (10)$$

where $\ddot{q}_i$ can be formulated from Eqns. (6), (8) and (9):

$$\ddot{q}_i = \frac{Q_i - \hat{\Phi}_i^T \hat{\mathbf{I}}_i^A {}_i\hat{\mathbf{X}}_{p(i)} \hat{\mathbf{a}}_{p(i)} - \hat{\Phi}_i^T (\hat{\mathbf{p}}_i^A + \hat{\mathbf{I}}_i^A \hat{\mathbf{c}}_i)}{\hat{\Phi}_i^T \hat{\mathbf{I}}_i^A \hat{\Phi}_i}. \quad (11)$$

These equations form the final loop of the ABA algorithm. The tree is visited again from its root to its tip to compute each joint and link acceleration.

Steps 5, 6 and 7: Numerical integration and system correction. Following the computation of $\ddot{q}_i$ for time step t_k , the joint acceleration update can be relaxed based on the previous time step to enhance numerical stability. The resulting accelerations are integrated to compute the velocities at the next time step t_{k+1} (defining $\Delta t = t_{k+1} - t_k$). Similarly, the joint velocities are integrated to compute the joint positions at t_{k+1} and the system state $\dot{q}_i(t_{k+1}), q_i(t_{k+1})$ serves as the initial conditions for the equations of motion for the next iteration of the forward dynamics (step 0).

OpenFOAM® supports three second-order integration schemes: Newmark (implicit) [40], Symplectic (explicit) [41], and Crank–Nicolson (implicit) [42]. Since the Crank–Nicolson scheme is a specific case of the Newmark method for particular parameter choices, this work focuses on the Newmark and Symplectic schemes.

The Newmark method updates the joint velocities and positions as:

$$\dot{q}_i(t_{k+1}) = \dot{q}_i(t_k) + \Delta t (\gamma \ddot{q}_i(t_{k+1}) + (1 - \gamma) \ddot{q}_i(t_k)) \quad (12)$$

$$q_i(t_{k+1}) = q_i(t_k) + \Delta t \dot{q}_i(t_k) + \Delta t^2 (\beta \ddot{q}_i(t_{k+1}) + (1 - \beta) \ddot{q}_i(t_k)), \quad (13)$$

where γ, β are the hyperparameters set to 0.5 and 0.25 by default. The implicit nature of the method requires $nIter$ updates of positions and velocities within each time step by repeating steps 0 to 7 in Fig. 1. These body state updates enhance solver stability, which is particularly important for stiff and strongly coupled problems. Importantly, the flow equations are not solved during these $nIter$ iterations, keeping the flow forces constant (step 0). This approach reduces the computational cost but limits the algorithm's applicability to moderately coupled problems where added mass effects are weak. The drones investigated in this work have smooth-wing kinematics which generate only negligible added mass effects, especially in air.

The second integration scheme considered is the explicit Symplectic scheme. Only one integration is performed per time step and the time steps must be kept small to maintain numerical stability in strongly coupled problems. The Symplectic scheme uses the leapfrog method for which a half step is taken to compute the joint velocities and positions:

$$\dot{q}_i(t_{k+0.5}) = \dot{q}_i(t_k) + 0.5 \Delta t \ddot{q}_i(t_{k-1}) \quad (14)$$

$$q_i(t_{k+1}) = q_i(t_k) + \Delta t \dot{q}_i(t_{k+0.5}). \quad (15)$$

The system dynamics is then solved (steps 0 to 4) and another half step is taken to update the joint velocities:

$$\dot{q}_i(t_{k+1}) = \dot{q}_i(t_{k+0.5}) + 0.5 \Delta t \ddot{q}_i(t_k), \quad (16)$$

It is worth noting that no higher-order time integration schemes are considered in this study, analogously to similar work [17, 20, 43]. The time step is dictated by the flow solver through CFL stability constraints, resulting in values which are several orders of magnitude smaller than the characteristic flapping period (see Section 5.1). Under these conditions, the potential accuracy gains of higher-order schemes are reduced. Moreover, such schemes require multiple evaluations of the aerodynamic forces within a single time step, implying repeated calls to the flow solver and non-trivial modifications to the current OpenFOAM® framework.

Following the updates of the joint positions and velocities using Symplectic or Newmark integration, the link position, velocity, and acceleration are corrected in step 7. The grid must then be adapted to this new state (Fig. 1).

2.3. Grid adaptation and PIMPLE loop. Referring to the second block in Fig. 1, the body displacement is computed from the current position at t_{k+1} and the previous position at t_k . The displacement is then applied to the grid cells with the *pointDisplacement* field. The cell centers of the grid are adjusted and the boundary conditions are corrected accordingly.

This work uses the overset method to adapt the grid to the body's motion. A small *component* grid is fitted to each body of the system, overlapping with a large *background* grid. The *component* grids can also partially overlap with each other, and interpolations allow the grids to exchange velocity and pressure fields according to a role attributed to each cell.

Active cells are standard cells where the flow equations are solved. Hole cells are background grid cells that overlap with the body and must block the flow. Interpolated cells are either neighbouring cells from the holes or cells at the component grid interface. They receive flow variables from interpolation stencils formed by active cells from another grid. Identifying these donor cells and the holes is computationally intensive as it requires spanning the background and component grid cells. Moreover, this operation must be repeated at the beginning of each time step. In addition, the interpolation of flow variables across grids introduces local conservation errors and numerical diffusion. Conservation errors have been shown to be negligible for open domains [44], while they may be further reduced using local conservation-correction schemes [35]. Numerical diffusion is mitigated by ensuring overlap across a few grid cells, matching cell sizes across grid interfaces, and using inverse-distance interpolation [35],

which provides a good compromise between accuracy and stencil compactness for flapping-wing simulations. The reader is referred to [35, 45] for further details.

Once the grid is updated and the fluxes are corrected according to the grid's motion, the momentum and pressure equations are solved to compute a new pressure field (block three in Fig. 1). The surface forces (Eqn. (1)) are updated, and the full loop starts again with the rigid body dynamics solver.

3. Implementations in the rigid body dynamics library

The articulated-body algorithm, available in OpenFOAM® v2206, solves the equations of motion to predict the kinematics of the joints and links. However, various multibody systems have joints that are actively controlled by actuators. For example, an actuator controls the rudder of a boat, the elevation flaps of an airplane, and the wings of a flapping wing drone, while the rest of the system dynamics is governed by the equations of motion.

The *rigidBodyDynamics* library partially allows modeling those constrained motions through a *restraint* named *prescribedRotation*. The latter controls the rotational velocity ω of a given link i by adding a force $\hat{f}_i^p$ to its external force vector $\hat{f}_i^e$. This additional force is modeled as the output of a Proportional-Integral-Derivative (PID) controller which seeks to minimize the rotational velocity error $e_\omega(t_k) = \omega(t_k) - \omega_{set}(t_k)$:

$$\mathbf{f}^p(t_k) = \frac{I_i}{dt} \left(K_p e_\omega(t_k) + K_d (e_\omega(t_k) - e_\omega(t_{k-1})) + K_i \sum_{t_j=t_0}^{t_k} e_\omega(t_j) \right), \quad (17)$$

where K_p, K_d, K_i are constant PID parameters and dt is the simulation time step. The *prescribedRotation* restraint could be easily generalized to control the motion defined by any joint (hence not only the rotation velocity of links). Nevertheless, the use of a PID controller requires tuning three parameters for each control scenario. This calibration process is tedious and unrealistic for computationally expensive simulations.

This work proposes to address these limitations by directly enforcing the kinematics $q, \dot{q}, \ddot{q}$ of active joints from a multibody system, bypassing the need to compute unknown joint forces and torques. Inspired by the work in [32–34], the proposed implementations modify the equations of motion to make the multibody system dynamics consistent with the prescribed motion of a subset of joints. The implementations consist of three main steps in the *rigidBodyDynamics* library. (1) The position, velocity, and acceleration of the active joints are loaded from user-defined analytical expressions, (2) the position, velocity, and acceleration are imposed on the desired joints, and (3) the system equations of those joints are modified to model their constrained dynamics while the equations of motion for the passive joints remain unchanged.

Implementation 1. A new library named *imposedMotion* is implemented based on the existing *restraint* library. The *imposedMotion* class defines generic operations that child classes can leverage, with each child class specifying different motion parameterizations, as demonstrated in Lsts. 1 and 2 for *wangParametrization*.

The class defines private variables read from the *dynamicMeshDict* dictionary. `jointList_` contains the indices of the active joints and `motionType_` designates the parameterization subtype implemented within the `loadImposedMotion` member function. This function saves the positions, velocities, and accelerations in `imposedJoints` from user-defined analytical expressions for the joints with indices in `jointList_`. The analytical expression is a function of the current time and parameters defined in the header.

Implementation 2. After the integration of the acceleration and the velocity (e.g. in *Newmark.C*), a loop iterates on the `jointIndex` list and imposes the $q, \dot{q}$ and $\ddot{q}$ from `imposedJoints` to the corresponding joints. The same implementation is done for the three numerical integration schemes supported by OpenFOAM® (i.e. *Newmark.C*, *symplectic.C* and *CrankNicolson.C*).

```

1  (...)
2  void Foam::RBD::ImposedMotion::
3  wangParametrization::loadImposedMotion
4  (
5  Field<label>& jointIndex ,
6  Field<scalar>& imposedJoints
7  ) const
8  {
9  if (motionType_ == "midStart")
10 {
11     (...)
12     Field<scalar> varList(
13     jointList_.size()*3);
14
15     scalar q_phi = Aphi_/asin(Kphi_)*
16     asin(Kphi_*sin(2*M_PI*fphi_*ti));
17     scalar qDot_phi = (...);
18     scalar qDdot_phi = (...);
19
20     varList[0] = q_phi;
21     varList[1] = qDot_phi;
22     varList[2] = qDdot_phi;
23
24     jointIndex = jointList_;
25     imposedJoints = varList;
26 }
27 (...)

```

Listing 1. Snippet of *wang-Parametrization.C*.

```

1  class wangParametrization
2  : public imposedmotion
3  {
4  // Parameters read from dynamicMeshDict
5  labelList jointList_;
6  string motionType_;
7
8  scalar Aphi_;
9  scalar fphi_;
10 (...)
11 public:
12 (...)
13
14 // Compute the q,qd,qdd from joints in jointIndex and
15 // save it in imposedJoints
16 // [q0,qd0,qdd0,q1,qd1,qdd1,...]
17 virtual void loadImposedMotion
18 (
19 Field<label>& jointIndex ,
20 Field<scalar>& imposedJoints
21 ) const;
22
23 // Update properties from given dict
24 virtual bool read(const
25 dictionary& dict);
26
27 // Write
28 virtual void write(Ostream&) const;
29 (...)

```

Listing 2. Snippet of *wang-Parametrization.H*.

Implementation 3. The last implementation modifies the equations of motion to make the dynamics of the multi-body system consistent with the imposed motions. Changes take place in steps 3 and 4 of the ABA algorithm implemented in *forwardDynamics.C* (Fig. 1). For active joints, the acceleration $\ddot{q}$ in Eqn. (11) is known with the implementations 1 and 2. The equation can be manipulated to isolate the magnitude of the unknown joint force Q_i that drives the motion of the joint. The latter is inserted in Eqn. (8), which changes the expressions of the articulated inertia and the bias force of link $p(i)$ as:

$$\hat{I}_{p(i)}^A = \hat{I}_{p(i)} + {}_{p(i)}\hat{X}_i \hat{I}_i^A {}_{p(i)}\hat{X}_i, \quad (18)$$

$$\hat{p}_{p(i)}^A = \hat{p}_{p(i)} + {}_{p(i)}\hat{X}_i \left(\hat{p}_i^A + \hat{I}_i^A \hat{c}_i + \hat{I}_i^A \hat{\Phi}_i \ddot{q} \right). \quad (19)$$

The resulting extended articulated-body algorithm (eABA) is summarised with the Algorithm 1 for which the red colour highlights the new implementations compared to the original ABA (in black). Line 1 calculates external loads using Eqn. (1) and user-defined forces/torques specific to the multibody test case. User-defined joint forces Q_j are also calculated, except for the n_a active joints driven by analytical functions in the eABA implementation. Line 2 loads the n_a analytical expressions of the position, velocity and acceleration (solely in eABA). Lines 5–8 (common to both ABA and eABA) calculate link quantities needed for inertia and bias force computation (lines 9–14). A conditional statement is added in the eABA to use Eqns. (18) and (19) for active joints (Line 12) and use Eqns. (7) and (8) otherwise. Lines 15–18 compute joint and link accelerations. In the eABA algorithm, acceleration computation of the active joints (Eqn. (11)) is not performed (line 16) since the accelerations, velocities and positions are imposed later (line 22) using the expression loads in line 2. With these implementations, the free motion of multibody systems can be simulated in a flow while one of its bodies follows a known trajectory. This is verified for the flapping-wing drones defined in the next section.

4. Test case definition

Two test cases are defined in the following section. The first test case is a single-wing (SW) drone used to verify the rigid body dynamics CFD environment in Section 5. The second test case is a body and wing (BW) drone used to analyze the dynamics of an ascending flight in section 6. In both test cases, only one wing is considered, defining the degrees of freedom to ensure symmetry of the flapping motion to the (z_I, x_I) plane. To reproduce these test cases, the reader should download the TEST_CASE/SW and TEST_CASE/BW folders from the GitHub repository and use the eABA libraries described in the previous section, available in the src folder.

4.1. Single-wing (SW) drone. The SW drone consists of a massless body and rigid, semi-elliptical wing, as shown in Fig. 3. The drone assumes bilateral symmetry of the flapping wings, such that a single wing and its kinematics are mirrored about the sagittal plane. This assumption is relevant for the perfect vertical ascent, without any control input, as detailed hereafter. The wing weighs half of a typical hummingbird [1, 8, 46] with a centre of mass that coincides with the instantaneous centre of rotation. The centre of mass is then outside the wing geometry, in a position similar to that of a hummingbird. This SW drone assumes then a negligible inertial

Algorithm 1: Extended articulated-body algorithm using Newmark time integration and with changes compared to the ABA indicated in red

```

1 Compute the external loads and restraints in  $\hat{f}_j^e$  and  $Q_j$  for the  $n - n_a$  links
2 Load the indices of the active joints in jointIndex and the imposed motion in  $q^a, \dot{q}^a, \ddot{q}^a$ 
3 Set the root velocity to 0 and its acceleration to the gravity
4 Forward dynamics
5   for link  $i = \text{root to tip}$  do
6     Compute link velocities  $\hat{v}_i$ : Eqn. (2)
7     Compute velocity-product forces  $\hat{c}_i$ : Eqn. (3)
8     Compute isolated inertia  $\hat{I}_i$  and bias force  $\hat{p}_i$ : Eqns. (4) and (5)
9   for link  $i = \text{tip to root}$  do
10    Compute articulated-body inertia  $\hat{I}_i^A$  and bias forces  $\hat{p}_i^A$  with:
11    if link's joint in jointIndex then
12      Eqns. (18) and (19)
13    else
14      Eqns. (7) and (8)
15   for link  $i = \text{root to tip}$  do
16     if link's joint not in jointIndex then
17       Compute joint acceleration: Eqn. (11)
18     Compute link acceleration: Eqn. (10)
19 Time integration
20   for joint  $i = \text{root to tip}$  do
21     if joint  $i$  in jointIndex then
22       Impose joint kinematics  $q_i = q_i^a, \dot{q}_i = \dot{q}_i^a, \ddot{q}_i = \ddot{q}_i^a$ 
23     else
24        $q_i, \dot{q}_i = \text{Newmark}(\ddot{q}_i, \dot{q}_i)$ 

```

and aerodynamic body-wing coupling as discussed in the introduction and in [23, 24]. The SW drone hypotheses greatly simplify the verification analyses undertaken in section 5, but may result in unrealistic flight trajectories as verified with the BW drone test case. Realistic flights are analysed in Section 6 leveraging the verification work.

The wing geometry is defined according to our previous works [47] using the design from [48], inspired by large hovering flyers such as hummingbirds [49]. Table 1 gathers its main dimensions, and Fig. 3.a shows its kinematic chain. The chain begins with two translational joints (T_{z_I}, T_{x_I}) along the vertical and longitudinal directions defined in the inertial frame $(x, y, z)_I$. These are used to solve for the drone trajectory defined with the variables x and z . The prismatic joints are followed by two revolute joints (R_{z_I}, R_{y_w}) around z_I of the inertial frame and y_w of the wing frame $(x, y, z)_w$ that coincides with the wing's axis of symmetry. These are used to prescribe the wing kinematics. Thus, the full kinematic chain conveys four degrees of freedom to the wing: the translational positions are computed by solving the equations of motion, while the rotational angles are imposed to follow periodic motions. The popular "Wang" parametrization [50–53] is chosen since it allows to impose periodical waveforms to the revolute joints using only two parameters. It is formulated as:

$$\phi(t) = \frac{A_\phi}{\arcsin(K_\phi)} \arcsin[K_\phi \cos(2\pi ft)], \quad (20)$$

$$\alpha(t) = \frac{A_\alpha}{\tanh(K_\alpha)} \tanh[K_\alpha \sin(2\pi ft)], \quad (21)$$

where Eqn. (20) defines the flapping angle due to the R_z joint and Eqn. (21) defines the pitching angle due to the R_y joint. Figure 3 shows both angles in the stroke plane parallel to (x_I, y_I) and in which the tip of the wing always remains. A_ϕ, A_α are the maximum flapping and pitching angles, f is the frequency for both rotations and K_ϕ and K_α drive the flapping and pitching waveforms. This article considers sinusoidal waveforms with the A_ϕ, A_α and f from Tab. 1 that were inspired by experimental measurements [54, 55] and CFD simulations [8, 56, 57] on the hovering flight of natural species.

In OpenFOAM®, the degrees of freedom are defined in the *dynamicMeshDict* wherein the *imposedMotion* dictionary with the *wangParametrization* is called to enforce the rotational motions (see Section 3). The Symplectic

Wing geometry		
Span	R	5 cm
Offset	R_0	2.25 cm
Mean chord	$\bar{c}$	1.5 cm
Thickness	b	0.045 cm
Wing motion		
Maximum flapping angle	A_ϕ	75°
Maximum pitching angle	A_α	45°
Frequency	f	20 Hz
Flapping shape parameter	K_ϕ	0.01
Pitching shape parameter	K_α	0.01
Wing dynamic parameter		
Mass	m	1.5 g
Inertia	I_x, I_y, I_z	$6e-6 \text{ kg} \cdot \text{m}^2$
Center of mass	$\mathbf{x}$	(0,0,0) m

Table 1. Parameters of the single-wing drone test case.

scheme is used to integrate the equations of motion and compared with the Newmark solver in section 5.1. The integration is done once three times per time step without acceleration relaxation (see Fig. 1 and Section 5.1).

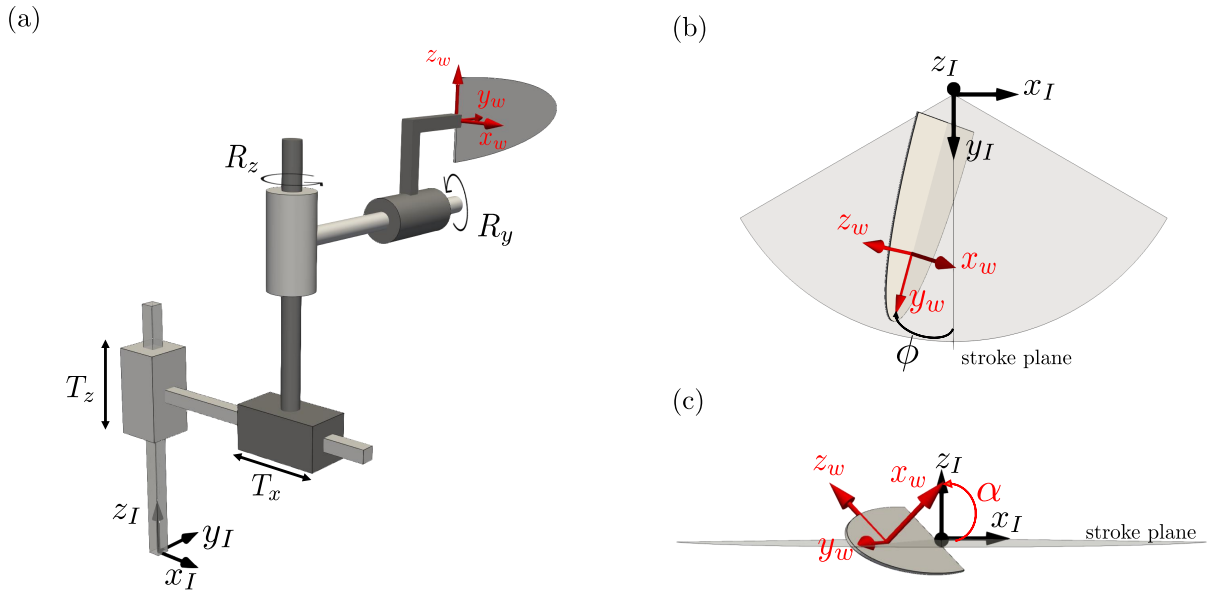


Figure 3. (a) Kinematic tree of the flapping-wing drone, showing its four degrees of freedom. (b) Top view and (c) side view of the wing, defining the flapping angle (ϕ) and pitching angle (α).

The grid is adapted to the wing motion using the overset method similar to other works focusing on tethered flights [6,9,58] and free flights [10,19,27]. Two overlapping grids are defined (see Fig. 2 from previous work [59]). Following the guidelines in [58], the background grid is a 20-chord cube with 500k (predominantly) hexahedral cells that are refined along the wing path to minimize inter-grid interpolation errors [60] and ensure a detailed capture of the coherent structures generated near the wing. The interpolation scheme used is the inverse distance.

The component grid is a C grid with 130k cells positioned within two chord lengths around the wing following literature standards [57,58] and previous works [47]. This grid follows the wing's translation and rotation, and the background grid follows the wing's translation. This moving background grid contrasts with most of the CFD works [60–62], which have used a fixed background grid and did not solve for the drone motion.

In the *dynamicMeshDict*, the cell zone containing the background grid cells is assigned to move with the centre of gravity of the second link from the kinematic tree by using the *drivenLinearMotion* library. For recall, there is a massless body between each joint, and so the second link is located just after T_x (Fig. 3).

The flow solver is *overPimpleDyMFoam* which is incompressible and transient. The solver uses the PISO algorithm with four pressure correction iterations for each momentum predictor step. The flow is on the limit of the transition regime ($Re \sim \mathcal{O}(10^3)$ when the drone hovers), and the LES model with the dynamic turbulent kinetic sub-grid model was shown to give similar aerodynamic forces to those from the *Fluent* simulation in [59].

Second-order central Gauss schemes are used for the spatial terms with limiters in the direction of the most severe gradients. A backward, second-order, implicit scheme is used for the time marching, and the simulations were performed with adaptive time steps capped by a maximum Courant number of 1. These schemes were already proven to give accurate results in [47].

Regarding the boundary conditions, the zero gradient condition is imposed on the boundaries of the background domain for the velocity field, together with a $0 \text{ m}^2/\text{s}^2$ fixed value for the pressure field. The wing patch is treated as a moving wall to impose a velocity distribution that results from its motion. The zero gradient condition is used on each domain patch for the *pointDisplacement* field and the *dynamicMeshDict* defines the parameters that govern the drone and the grid displacement.

4.2. Body and wing drone. The second test case adds a spherical body to the single-wing drone, introducing aerodynamic and inertial coupling between the wing and the body. Figure 4 shows the kinematic tree. Two translational joints (T_z, T_x) and one revolute joint (R_y) connect the spherical body to the inertial frame. The translational joints allow the horizontal x and vertical displacement z of the drone, and R_y allows the body to pitch with angle θ as defined between z_s of the body frame and z_I of the inertial frame. The spherical body is connected to the wing by the same revolute joints used for the single-wing drone (Section 4.1). The wing can execute both flapping and pitching motions relative to the body. Unlike the single-wing drone, the stroke plane is defined in $(x, y, z)_b$, which can then be tilted according to the pitching angle θ .

The wing motion is parameterized by the Eqns. (20) and (21) (Tab. 1). The sphere has a radius equal to the chord length, the wing mass is 10% of the sphere mass, which is slightly larger than that of typical hummingbirds and drones [1, 8]. The wing inertia is 1% of the sphere inertia, as approximated using the wing mass and dimensions, and Section 6.4 analyses the influence of this wing-to-body mass ratio. A component grid is generated around the sphere (160k cells) and the wing (130k cells) as shown in Figs. 5a and 5b. All the other numerical parameters are set as for the single-wing drone test case.

Finally, for the verifications of the CFD environment undertaken in section 5.2, the dimensionless aerodynamic forces of the wing are defined in the x_I and z_I direction of the inertial frame as:

$$C_{x_I, w} = \frac{\int_{S_w} (\rho p \mathbf{1} + \boldsymbol{\tau}_v) d\mathbf{S}_w \cdot \mathbf{x}_I}{0.5 \rho S_w U_{ref}^2}, \quad (22) \quad C_{z_I, w} = \frac{\int_{S_w} (\rho p \mathbf{1} + \boldsymbol{\tau}_v) d\mathbf{S}_w \cdot \mathbf{z}_I}{0.5 \rho S_w U_{ref}^2}, \quad (23)$$

and similarly for the drone's body:

$$C_{x_I, b} = \frac{\int_{S_b} (\rho p \mathbf{1} + \boldsymbol{\tau}_v) \cdot d\mathbf{S}_b \cdot \mathbf{x}_I}{0.5 \rho S_b U_{ref}^2}, \quad (24) \quad C_{z_I, b} = \frac{\int_{S_b} (\rho p \mathbf{1} + \boldsymbol{\tau}_v) \cdot d\mathbf{S}_b \cdot \mathbf{z}_I}{0.5 \rho S_b U_{ref}^2}, \quad (25)$$

where the reference velocity is defined as $U_{ref} = 4fA_\phi R_2$ at the radius of second moment of area $R_2 = \sqrt{\int_{R_0}^{R_0+R} c(r)r^2 dr / S_w}$.

5. Results from the single-wing drone

This first results section verifies the accuracy and performance of the rigid body dynamics CFD environment described in Section 3 using the single-wing drone test case. Section 5.1 presents two parametric studies: a grid independence study and a study on the integration scheme. Section 5.2 compares the drone forces and trajectories with those computed by a simplified environment developed in Python. Finally, Section 5.3 discusses the computational cost of typical simulations.

5.1. Parametric studies. Figure 6a tests the influence of the grid refinement on the lift force generated by the wing. For clarity purposes, only the last three flapping cycles of a 1 second flight are shown. The grids are refined by first increasing the cell count of the component grid and then adapting the grid refinement of the background grid to conserve similar cell sizes at the interface between the two grids. The coarse, medium, fine and fine+ grids contain 71k, 214k, 501k and 808k cells, respectively. Overall, the selected grids have a minor influence on the lift curves; all curves overlap and follow a similar sinusoidal trend imposed by the flapping kinematics. The largest discrepancies are observed at the lift peaks, where the finest grids result in slightly higher amplitudes on average.

Figure 6b further evaluates the influence of these force discrepancies on the x - z trajectory followed by the drone. The influence of the grid on the trajectory is small and assessed with the error $e = (||\mathbf{x}_f(i)||_2 - ||\mathbf{x}_f(i+1)||_2) / ||\mathbf{x}_f(i+1)||_2$ for which $i \in \{\text{Coarse}, \text{Medium}, \text{Fine}, \text{Fine+}\}$ and $\mathbf{x}_f = (x, z)$ is the final position taken when $t = 1 \text{ s}$. A 6.1%, 2.3% and 1.3 % error is computed going from the coarsest to the finest grid. One can assume that sufficient grid independence is reached with the fine grid. Similarly, Fig. 6c shows the influence of the step size Δt imposed through the CFL number. The average Δt is $7.7e - 06 \text{ s}$ for CFL = 0.5 and the double CFL = 1.

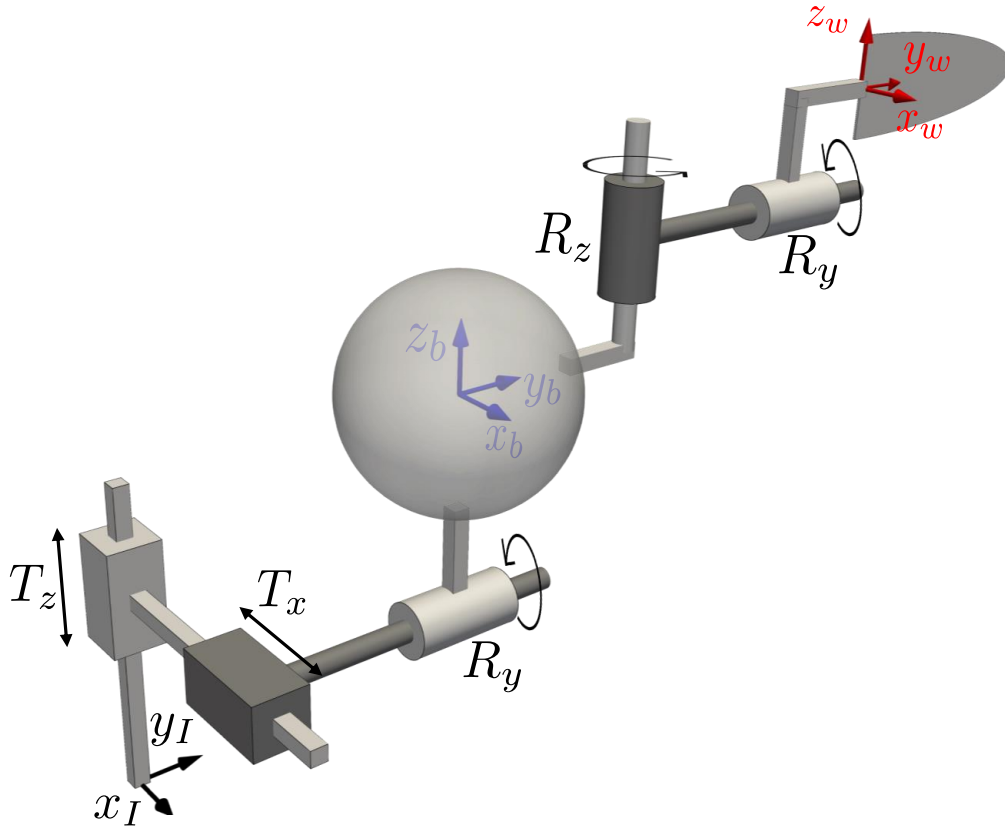


Figure 4. Kinematic tree of the drone consisting of a spherical body and a wing with 5 degrees of freedom.

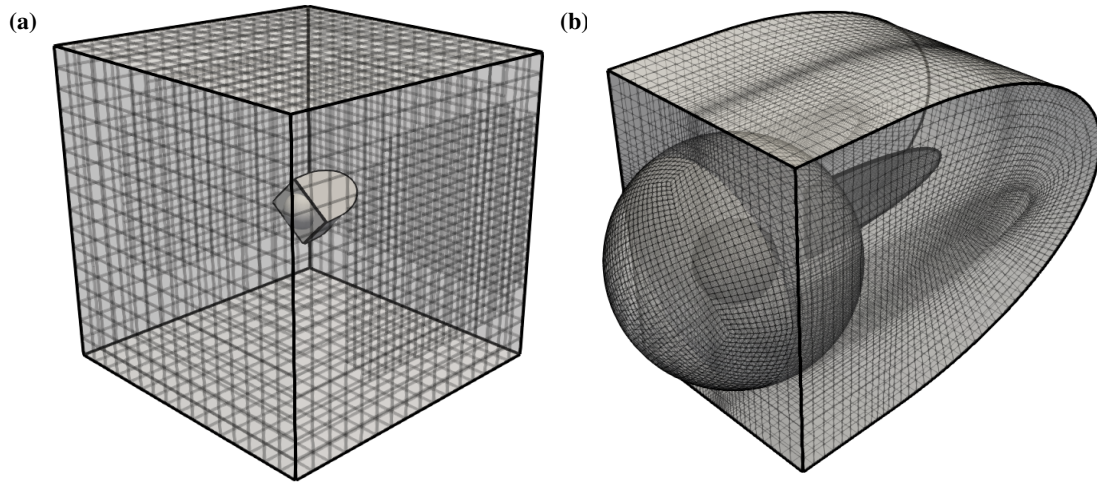


Figure 5. (a) Background grid shown in transparent mode with the component grid of the spherical body and the wing. (b) A closer view of the component grids.

The figure clearly shows that the step size with $CFL = 1$ is sufficiently small to result in trajectories that are almost overlapping with those with $CFL = 0.5$.

Figure 6d shows the influence of the numerical scheme used to integrate the equations of motion. The symplectic method (Eqns. (14) and (16)) is compared with the Newmark method (Eqn. (12)), evaluated with one and three passes of the rigid body dynamics solver per time step. Both methods rely on similar time steps determined by the adaptive time-stepping scheme of the flow solver.

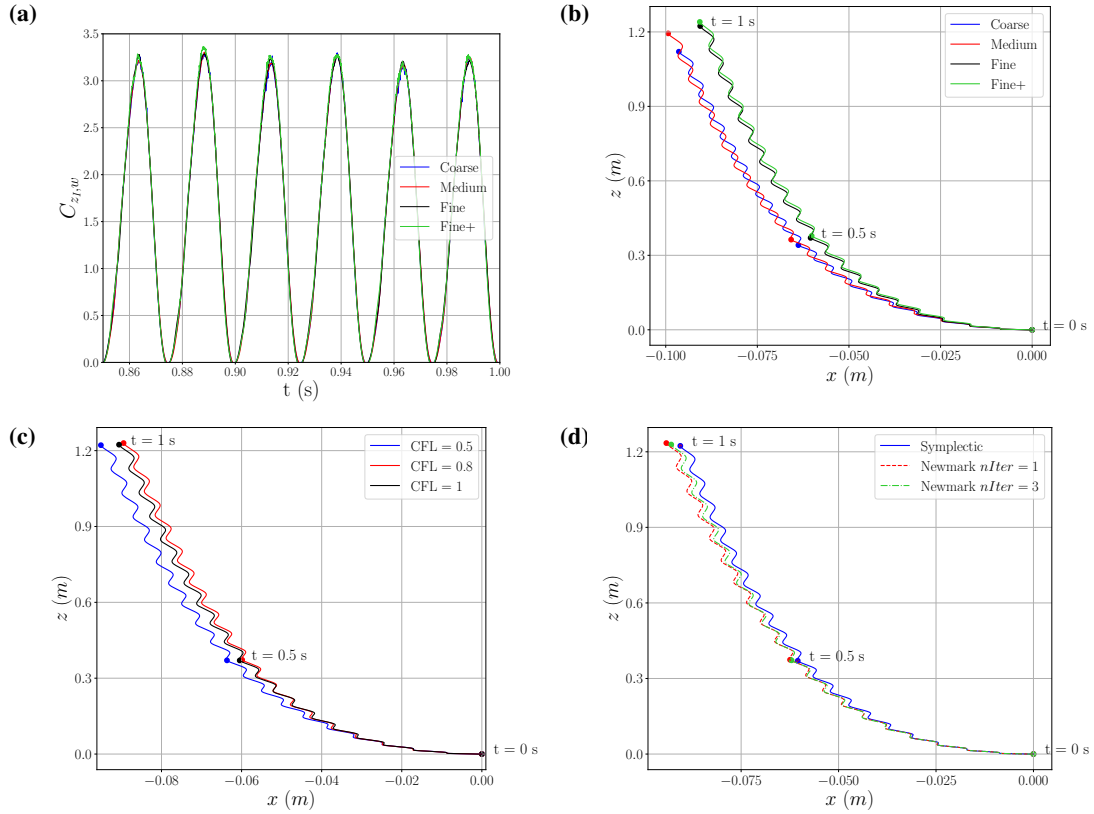


Figure 6. (a) Time evolution of the lift coefficient for coarse (71k cells), medium (214k cells), fine (501k cells), and fine+ (808k cells) grids during the last three cycles of the ascending trajectory shown in (b). (c) x-z trajectory for CFL numbers of 0.5, 0.8, and 1.0. (d) x-z trajectory obtained using the Newmark and Symplectic integration schemes.

The figure demonstrates that both solvers produce similar trajectories, with the final position differing by less than 1%. It also shows that increasing sub-iterations with the Newmark scheme brings the trajectory closer to that of the symplectic method. Notably, no stability issues were observed with either method.

5.2. Verification studies. Parametric studies from the previous section demonstrated the robustness of the CFD setup, showing that the results are not sensitive to numerical parameters. The next step in the validation process is to benchmark the CFD results, ideally with experimental data. However, flapping-wing drones are rare, and none offer sufficiently reproducible setups or detailed results. Similarly, numerical simulations of flapping-wing flights remain scarce, as noted in the introduction. Few studies have employed multibody formulations but they rely on complex setups, making them impractical for validation purposes. Given these limitations, the numerical environment is verified in three steps. First, the flow solver and rigid-body dynamics solver are validated independently against reference data, after which their coupling is verified using a simplified model under conditions where it is known to provide reliable predictions.

5.2.1. Flow solver validation. The flow solver was first validated without the body dynamics solver in a previous work [47] (using the overset method and LES). Harmonic motions of the wing were simulated in quiescent flow and the aerodynamic forces were found to match the ones from a Fluent simulation using the sliding mesh approach [48].

5.2.2. Rigid body dynamic solver validation. The rigid body dynamic solver (eABA) is validated independently from the flow solver using a 2D double pendulum in a vacuum. This multibody system is a standard benchmark, extensively analyzed in previous numerical studies [63–65]. Its non-linear and chaotic dynamical system is defined using Lagrangian mechanics and implemented in Python following [65]. The resulting equations of motion are time-integrated using a Runge-Kutta scheme to provide the benchmark data for comparison against the double pendulum simulated within the OpenFOAM® environment.

The double pendulum is shown in Fig. 7a. It consists of two massless rods with length $l_1 = 10$ cm and $l_2 = 20$ cm, each with a point mass $m = 0.5$ kg at its end. A revolute joint R_z connects the top of the first rod to the origin of

the inertial frame $(x, y)_I$. A second revolute joint R_z connects the tip of the first rod with the top of the second rod. The system motion is then characterized by the angles θ_1 and θ_2 . Figure 7a (right) illustrates the grid setup, where the component grid for the second rod is generated only around its second half. This design ensures a clearance of l_1 between the two component grids, preserving a sufficient number of cells between the pendulums and preventing interpolation issues [66]. Since no fluid forces act on the system, this computation grid setup does not influence the pendulum dynamics.

To test the eABA, the first joint of the pendulum is activated using the cosine function $f(t) = A \cos(2\pi ft) - A$. At each time step, this function drives the angle $\theta_1(t) = f(t)$, velocity $\dot{\theta}_1(t) = \dot{f}(t)$, and acceleration $\ddot{\theta}_1(t) = \ddot{f}(t)$, with $A = 57$ (deg) and $f = 2$ Hz. The resulting motion of the second rod is measured through θ_2 , and Fig. 7b compares its time evolution computed when using the eABA (implemented in OpenFOAM®) and the Lagrangian mechanics (implemented in Python). The figure shows that the angle θ_2 exhibits the same time evolution in both the eABA and the Lagrangian mechanics computation, thereby validating the implementation of eABA.

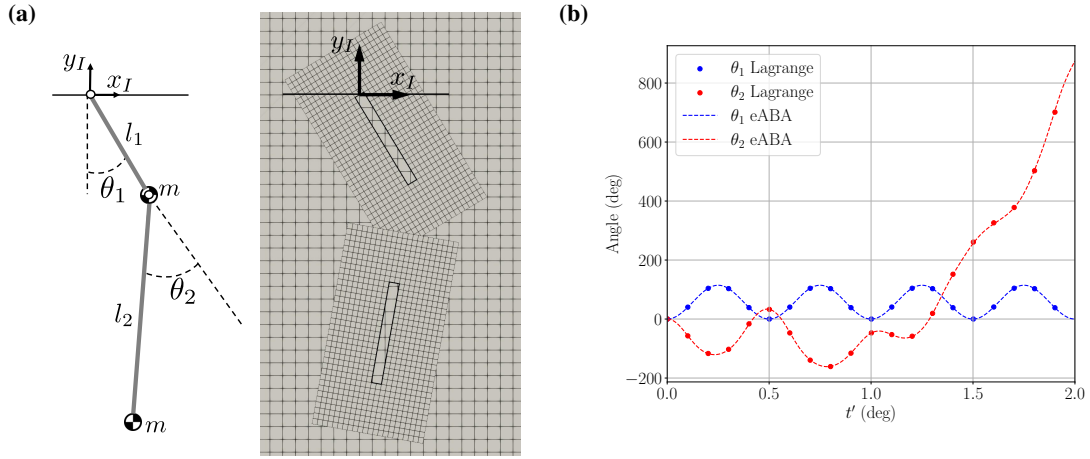


Figure 7. (a) Schematic of the double pendulum, its two component grids and the background grid and (b) Comparison of the pendulum angles θ_1 and θ_2 using the eABA and Lagrange mechanics.

5.2.3. Coupled solver verification. The coupling between the flow solver and the rigid-body dynamics solver is verified using a quasi-steady (QS) flight dynamics model commonly employed in the flapping-flight literature due to their good aerodynamic predictions for smooth wing kinematics [10, 48, 67]. A potential limitation of this verification approach is that the reduced-order QS model does not capture all unsteady flow effects and depends on coefficients calibrated with CFD data, which can introduce circular reasoning when comparing CFD predictions against such models. These risks are mitigated by limiting the comparison to smooth flapping kinematics, for which quasi-steady models are known to be accurate. In addition, the used reduced-order model [48] was calibrated using CFD results obtained with an independent solver and validated against experimental data.

The simplified model is implemented in Python and combines an aerodynamic model with the drone equations of motion. The aerodynamic model is the quasi-steady formulation derived in [48] which estimates the lift and drag forces along z_I and $x_w \cos \alpha$ (Fig. 3):

$$L_w(t) = \frac{1}{2} \rho C_{L,w}(\alpha) \int_{R_0}^R U_w^2(t, r) c(r) dr, \quad (26)$$

$$D_w(t) = \frac{1}{2} \rho C_{D,w}(\alpha) \int_{R_0}^R U_w^2(t, r) c(r) dr, \quad (27)$$

where $C_{L,w}$ and $C_{D,w}$ are the lift and drag coefficients that have been calibrated with *Fluent* simulations of flapping wings in still air [48]. The authors of [48] extensively validated their CFD simulations against experimental measurements and subsequently verified the resulting QS model under out-of-training conditions. This makes the model a widely used reference for smooth flapping-wing kinematics.

Equations (26) and (27) are modified from [48] to express the wing velocity U_w as a function of the flapping velocity $\dot{\phi}$, as well as the body linear and rotational velocities. The drone's equations of motion, derived from aircraft flight dynamics principles [25], describe the translational motion of the drone's center of mass. Specifically, they define the two translational degrees of freedom (x, z) (Fig. 3) resulting from the drone's weight (mg) and the aerodynamic forces generated by the wings ($F_{x,w}, F_{z,w}$):

$$m\ddot{x} = F_{x,w} \quad (28)$$

$$m\ddot{z} = F_{z,w} - mg \quad (29)$$

where the wing forces result from the projections of Eqns. (26) and (27) in the inertial frame. The translational trajectory of the single-wing drone is then obtained from the integration of system (28)-(29) using a Runge Kutta scheme. Implementation details on the simplified environment are given in [68] and the remainder of the section compares the two environments using the parameters of Tab. 1.

Figure 8 first shows the flapping (blue) and pitching (red) angle, velocity, and acceleration as a function of the travel time for the first six cycles. The figure demonstrates that the ABA algorithm implementation (section 3) enables the multibody system's joints to accurately track the kinematics prescribed by Eqns. (20) and (21). These equations and their derivatives are directly used in the quasi-steady model (labeled QS).

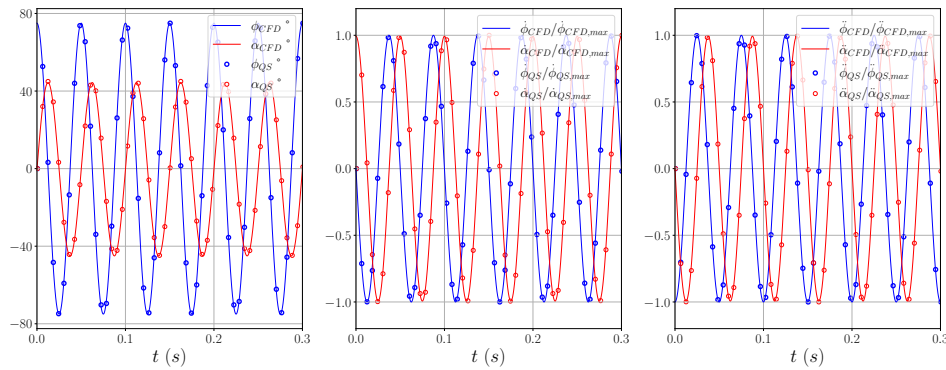


Figure 8. Comparison of the flapping and pitching angle (left), velocity (middle), and acceleration (right) between the analytical expression (20) and (21) used in the *QS* model and the motion from the R_z and R_y joints of the multibody drone.

Figure 9 compares the wing forces $C_{xI,w}$ and $C_{zI,w}$ (Eqns. (22) and (23)) in the CFD and QS environments. The first half of the cycle (downstroke) is left unshaded, and the second half (upstroke) is shaded in gray.

The forces from the CFD follow the same trend as the QS forces, verifying the correct implementation of the flow solver, the rigid body dynamics solver and their coupling within the CFD framework. The error is quantified using the normalized root-mean-square error (NRMSE) to avoid division by zero: it is 9.5% and 9.1% for the drag and the lift, respectively.

Discrepancies are primarily attributed to limitations of the simplified model. In particular, the QS formulation relies on semi-empirical lift C_{Lw} and drag C_{Dw} coefficients to model the influence of the LEV. These coefficients were obtained through regression on CFD simulations [48], which introduces (small) numerical errors and limits its use as an independent aerodynamic reference. Additionally, while the quasi-steady LEV dominates force generation during smooth wing motion, secondary unsteady effects still influence the force profiles to a minor extent [47]. At stroke reversal, wing-wake interactions introduce small oscillations in the CFD results, as discussed in [69, 70] and Section 6. This highly unsteady phenomenon can originate from four primary physical mechanisms [69], and occurs across a broad range of low Reynolds numbers ($10^1 - 10^4$) with an intensity that strongly depends on wing kinematics and geometry. For instance, wings following kinematics with small flapping amplitudes and rapid pitching experience strong wing-wake interactions [59], whereas wings with low aspect ratio or operating at higher Rossby numbers exhibit reduced interaction with the previously shed wake [69].

The used QS model also omits the Wagner effect, which governs the transient buildup of LEV circulation as the wing accelerates from rest [71]. Nevertheless, the imposed kinematics is smooth and characterized by small accelerations, so that only a slight phase lag between CFD and QS force is noted in Fig. 9. This effect is usually ignored in simplified models while it was included for some models relying on more aggressive maneuvers [71, 72]. For the same reasons, the influence of added mass is also negligible for the investigated flight conditions [48, 67]. The added-mass also depends on the small flapping and pitching accelerations which make this non-circulatory effects negligible compared to circulatory contributions. Conversely, various species and flapping-wing drones have large flapping and pitching accelerations at stroke reversal which result in large, short-duration force peaks [9, 53, 67, 73]. For the smooth kinematics considered in this work, including an added-mass term in the quasi-steady model (Eqns. (26)–(27)) would therefore only induce a slight increase in normal force during wing acceleration and a corresponding decrease during deceleration, without altering the overall and average force trends.

The slightly different wing forces in the CFD and QS lead to minor changes in the drone acceleration. This results in different relative velocities of the wing, which influence the wing force computation, which in turn influences the drone acceleration, etc. This feedback loop has light effects on the drone dynamics, as shown with Fig. 10, which presents the complete drone trajectory over time (NRMSE of 5.4%). The longitudinal (x) position shows a maximal discrepancy of 4 cm between the models, while the vertical position (z) has a maximal discrepancy of 10 cm.

Interestingly, Fig. 10 shows that the drone slowly drifts backwards. This is linked to the time evolution of $C_{xI,w}$. During the downstroke (unshaded area), $C_{xI,w}$ is negative, accelerating the drone backwards. In the upstroke (grey area), $C_{xI,w}$ is positive, decelerating the drone while the drone is still moving backwards. This explains the oscillations seen on the x -position. This drift phenomenon strongly depends on the initial state of the wings. When the wings start at mid-stroke at $t = 0$ s, negative and positive drag contributions cancel each other, resulting in a minimal drone drift. Any residual drift is attributed to the body velocity effect on the wing forces, as also visible in Fig. 10. At the end of the first cycle, the drone is nearly at rest, but it moves more and more forward from cycle end to cycle end.

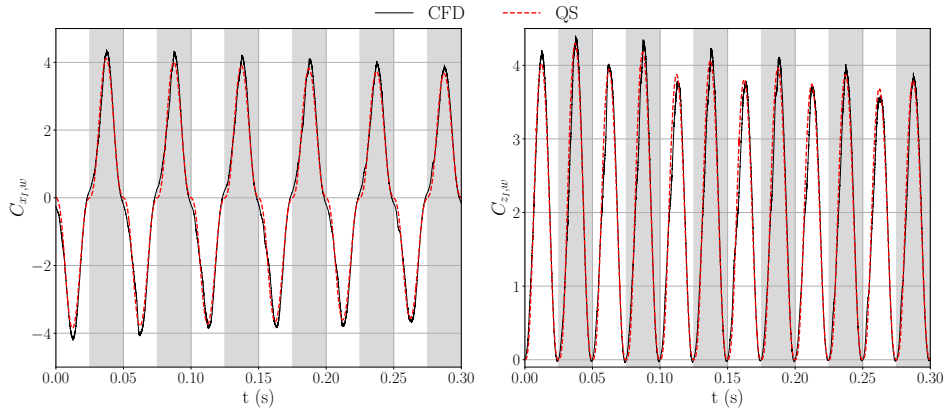


Figure 9. Comparison of the wing forces computed in the CFD environment and in the QS model for the longitudinal direction (left) and vertical direction (right) during 6 flapping cycles.

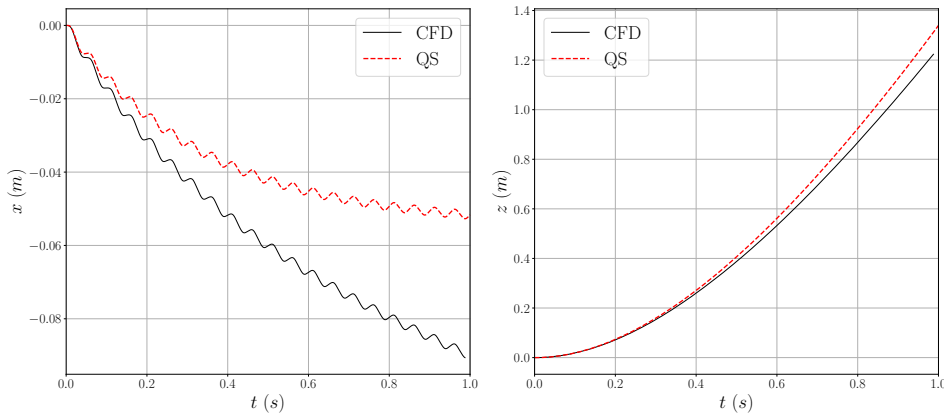


Figure 10. Comparison of the drone position computed with the CFD environment and with the QS model for the longitudinal direction (left) and vertical direction (right) during 20 flapping cycles.

5.3. Computational time characterization. This section analyses the computation time of the simulation described above. The total execution time is divided into three main operations: the rigid body dynamics (RBD), the overset method, and the pimple loop. The rigid body algorithm starts from the computation of the wing forces and ends with the update of the boundary conditions (Fig. 1), the pimple algorithm mainly solves the momentum and pressure equations and the overset method includes all the rest.

The computations are performed on the 501k cells grid (Fig. 6b) using 48 processors of the 64 present on one node from the cluster of the Von Karman Institute. The cluster has 12 nodes with Dual AMD Rome Epyc 7742 and 1TB of RAM in 32 DIMMs for 3200 MHz with 8 memory channels. Previous studies have shown that the Scotch decomposition results in a high computational cost when using the overset method due to excessive processor communication [74]. Therefore, three grid decomposition methods, namely Scotch, Hierarchical (zxy), and Simple are tested in Fig. 11. The figure shows the execution time percentages taken by the three operations during 5 flapping cycles. The simulation took around 5 h for the hierarchical decomposition, 8 h for the scotch decomposition, and 10 h for the simple decomposition. The execution time overhead added by the rigid body algorithm is negligible, unlike the overset method, which is extremely time demanding (63% of the total time taken by *overPimpleDyMFoam* with the scotch method). The overset method calls three main functions [45]: an `update` function that updates the roles of the cells according to the wing motion (*inverseDistanceCellCellStencil.C*), a `updateAddressing` function that excludes holes and includes interpolated cells in the equation matrices (*dynamicOversetFvMesh.C*), and an `interpolateFields` function in *cellCellStencilTemplates.C* that interpolates the fields from the component grid to the background grid and vice versa. The first operation involves updating the position of the hole cells and interpolated cells as well as searching for donor cells from which interpolation is performed. This last operation is especially time-consuming and strongly influenced by the total number of cells. Using the fine grid as a reference (501k cells), the execution time per flapping cycle is reduced by a factor of 4.7 on the coarse grid and by a factor of 2.5 on the medium grid, while it increases by a factor of 5.7 on the fine+ grid. These results were obtained using the same grid decomposition method (Scotch) and a comparable number of cells per processor ($\sim 12k$), such that the observed variations primarily reflect the scaling of overset-grid operations with grid resolution. These results confirm the huge impact on overall runtime as the grid is refined, and hence the need for the method optimisation if larger grids are considered [75].

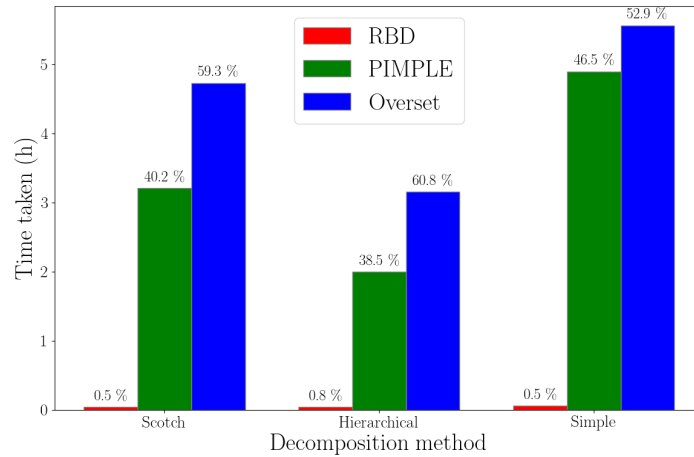


Figure 11. Bar chart of the execution time taken by the rigid body dynamics solver, the overset method, and the PIMPLE loop for the scotch, hierarchical, and simple grid decomposition methods.

6. Results from the body and wing (BW) drone

This section analyses the flight performance of the BW drone described in Section 4.2.

6.1. Longitudinal motion. Figure 12 (left) shows the time evolution of the horizontal position of the drone's centre of mass. The trajectory followed by the SW drone is also shown as a dashed line. Both drones experience backwards drift due to the influence of wing drag (as discussed in the previous section). The BW drone exhibits larger peak-to-peak oscillations due to its inertial coupling with the wing: as the wing flaps in one direction (e.g., toward $x < 0$), the body moves slightly in the opposite direction (e.g., toward $x > 0$). To isolate this effect, Fig. 12 (left) also shows the trajectory of the BW drone flying in a vacuum, where no aerodynamic forces are present (Eqn. (1)). These kinematically induced oscillations combine with the aerodynamically induced oscillations detailed in Section 5.2.3. One can also notice that at $t = 0.7$ s, the drone starts moving in the opposite direction. This is explained by the larger pitching angle θ as detailed in Section 6.5.

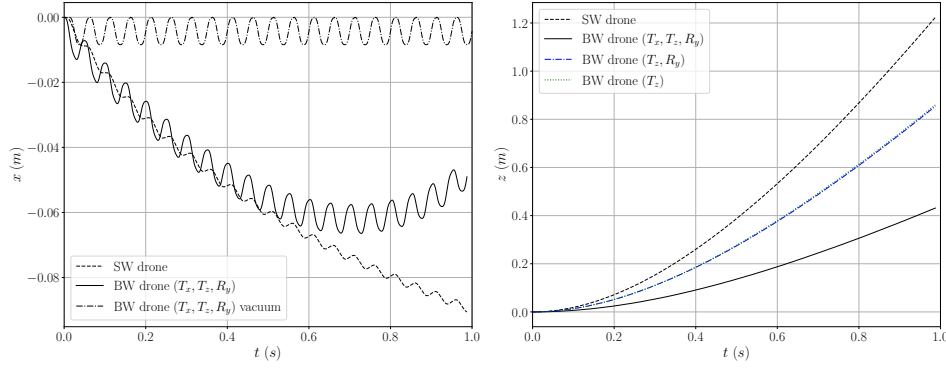


Figure 12. Horizontal (left) and vertical (right) position of the drone body as a function of time for 20 flapping periods.

6.2. Vertical motion. The SW and BW drones generate an average lift exceeding their weight, causing them to ascend (Fig. 12 right). However, the BW drone reaches a height ~ 0.7 m lower than the SW drone. To illustrate this difference, Fig. 12 (right) presents two additional trajectories obtained by simulating the drones with a smaller degree of freedom set. The figure shows (1) a BW drone with a body featuring two degrees of freedom (T_z, R_y), represented by the dash-dot blue line, and (2) a BW drone with a single degree of freedom body (T_z), shown as a dotted green line.

Comparing these trajectories reveals that the longitudinal oscillations of the full-DOF BW drone tend to slow its ascent. As the body moves opposite to the wing, the relative airflow over the wing decreases, reducing the vertical force F_z . Additionally, the body contributes a vertical drag force opposing the motion (Fig. 13, right), which becomes increasingly negative as the drone climbs. However, its impact is minor, being two orders of magnitude smaller than the wing-induced drag.

The BW drone also slightly pitches up and down which tilts the stroke plane, slightly increasing F_x and decreasing F_z . These small pitching angles do not explain the large z-position discrepancies, as confirmed by comparing the trajectories of the one dof body (T_z) and two dofs body (T_z, R_y). This differs from observations made in [18] where larger pitching motions were reported to strongly change insect trajectories.

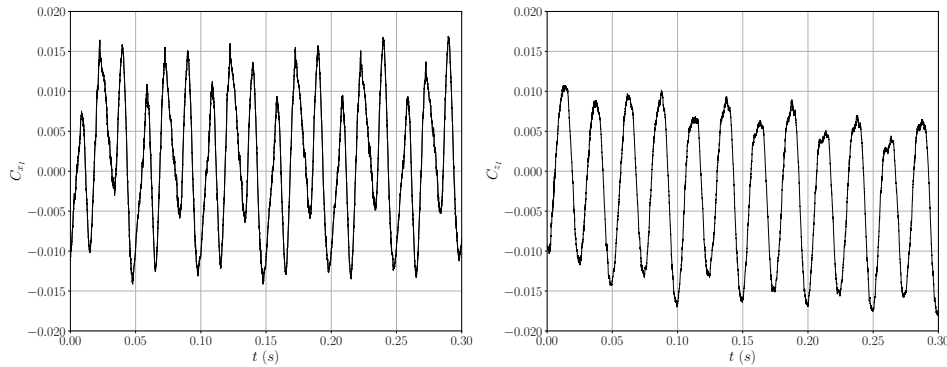


Figure 13. Horizontal (left) and vertical (right) forces of the drone body as a function of time for the 6 first flapping periods.

6.3. Pitching dynamics. Figure 14a shows the body pitch angle θ (left) and the pitching torques $C_{M_{yy,w}}$ induced by wing forces to the centre of mass of the body (right). The cycle is divided into four distinct phases, illustrated by different colours in Figs. 14a and 14b: downstroke start to midstroke (DM), midstroke to upstroke start (MU), upstroke start to midstroke (UM), and midstroke to downstroke start (MD). The body starts with $\theta = 0^\circ$, indicating that the stroke plane is parallel to the ground (Fig. 3). During the downstroke (DM and MU), θ predominantly decreases, while it increases during the upstroke (UM and MD). The first cycle ends with $\theta > 0^\circ$, indicating a nose-down body orientation and a tilted stroke plane. This small deviation from $\theta = 0^\circ$ carries over to subsequent cycles, after which the pitching angle gradually decreases throughout the simulation. The body orientation is therefore passively unstable, which was also observed in various flapping species [4]. This instability arises primarily

from the wing kinematics and aerodynamic forces. The wing kinematics influences the body pitching dynamics analogously to its influence on the longitudinal dynamics discussed in Section 6.1. Specifically, the wing pitching, combined with the flapping motion, induces a counter-pitching motion of the body due to a reaction torque at the body-wing joints. This torque gradually reduces the body's pitching angle over successive flapping cycles as shown in Fig. 14a (left) with the dash-dot line showing the trajectory of the BW drone simulated in a vacuum.

Nevertheless, the influence of the wing kinematics on the pitching dynamics is surpassed by the aerodynamic excitation of the wings. The latter is less trivial to understand. A first analysis could conclude in torque equilibrium as the lift should generate a torque during DM (UM) that is counter-balanced by the torque during MU (MD). Figure 14a (right) shows that the pitching torque has indeed four torque peaks during each part of the cycles but the peaks from DM and UM are smaller ($|C_{M_{yJ,w}}| \sim 6$) than the ones from MU and MD ($|C_{M_{yJ,w}}| \sim 8$). This asymmetry has three main origins, explained with the help of a schematic in Fig. 14b and the constitutive equation of the pitching moment:

$$M_y = r_{cz}F_x - r_{cx}F_z, \quad (30)$$

where $\mathbf{r}_c = [r_{cx}, r_{cy}, r_{cz}]$ is the lever arm, i.e., the distance between the center of pressure and the center of rotation (which is set to the center of mass). Firstly, the forces F_x, F_z are not equal during the downstroke (DM and MU) and upstroke (UM and MD) because they are functions of the body velocity. For example, the LEV contribution is a function of the relative wing-air velocity, which is itself a function of the body velocity. The wing-wake interaction contribution also varies with the drone motion. At the beginning of the flight, when the drone is almost hovering, the flow is not fully developed, and the wing faces different wake flows during the successive downstrokes and upstrokes. Afterwards, the wing encounters fewer wake structures because it continuously accelerates upwards.

Secondly, the amplitude of the forces is not symmetric within a stroke due to the Wagner effect. The time integral of the lift during the second stroke halves (MU and MD) is larger than during the first halves. Consequently, the pitching moment is increased during the second half of the strokes.

Thirdly, even if one ignores the previously described asymmetry sources, the pitching moment would still give two different amplitude peaks due to the center of pressure that is offset from the wing symmetry axis (i.e. $r_{cz} \neq 0$).

To clearly explain this, one can look at the different signs from the two contributions of Eqn. (30) during a flapping cycle. Looking at Fig. 14b, knowing that r_{cz} is negative, one can assume that the first contribution $r_{cz}F_x$ is positive during DM (negative during UM). The drag keeps its sign during a stroke, and so does $r_{cz}F_x$ during the second half of the stroke. On the other hand, for the second contribution, r_{cx} changes sign during one stroke (Fig. 14b), inducing a nose-up and nose-down motion during each half-stroke. The summation of the two terms leads to an increase or decrease of the pitching moment amplitude.

6.4. Parametric analyses. For the flapping wing drone analyzed in the previous subsections, the wing mass was set to 10% of the body mass and the wing moment of inertia was estimated based on textbook formulas. This section evaluates how the drone dynamics evolves as the wing mass is decreased and so as the wing-to-body mass ratio $\eta = m_w/m_b$ is reduced to 5% and 1%, which are representative of many insects and hummingbirds [54]. All other dynamic parameters are kept identical to those listed in Tab. 1.

Figure 15a presents the resulting x-z trajectories, while Fig. 15b reports the time evolution of the body pitch angle. Each case is shown over 0.7 s, corresponding to 14 flapping cycles. As expected, reducing η enhances vertical performance: the configuration with the lightest wings reaches the highest altitude. Simultaneously, longitudinal oscillations are markedly attenuated. This reduction follows directly from the weaker dynamic coupling between the wings and the body as wing mass decreases. The limit case of this trend was already observed in Fig. 12 for the SW configuration, where wing mass was neglected entirely.

Beyond influencing oscillation amplitudes, wing mass also substantially affects longitudinal drift. The case with $\eta = 1\%$ exhibits a monotonic drift toward the negative x-direction with the largest final amplitude. In contrast, the $\eta = 10\%$ drone initially moves in the negative x-direction before reversing its motion. The intermediate case, $\eta = 5\%$ displays a transitional behavior: a pronounced initial backward motion followed by a more moderate recovery. These distinct drift patterns highlight the sensitivity of translational dynamics to wing inertia and can be directly interpreted through the pitch dynamics shown in Fig. 15b.

The $\eta = 10\%$ drone possesses the largest wing-induced moment of inertia and therefore generates the strongest inertial pitching torque acting on the body. As discussed in the previous section, this inertial torque tends to reduce the body pitch angle. Conversely, when the wings are light ($\eta = 1\%$), the inertial pitching torque becomes negligible and the aerodynamic torque dominates. The resulting tilt of the stroke plane reorients the lift vector rearward, thereby promoting sustained motion in the negative x-direction.

6.5. Flowfield analysis. This section illustrates the aerodynamic phenomena driving the drone's motion through the time evolution of the pressure field, velocity field and vortical structures. Fig. 16 shows the $x - z$ trajectory overlaid with snapshots of the drone at the midpoints of the upstrokes across successive flapping cycles. As

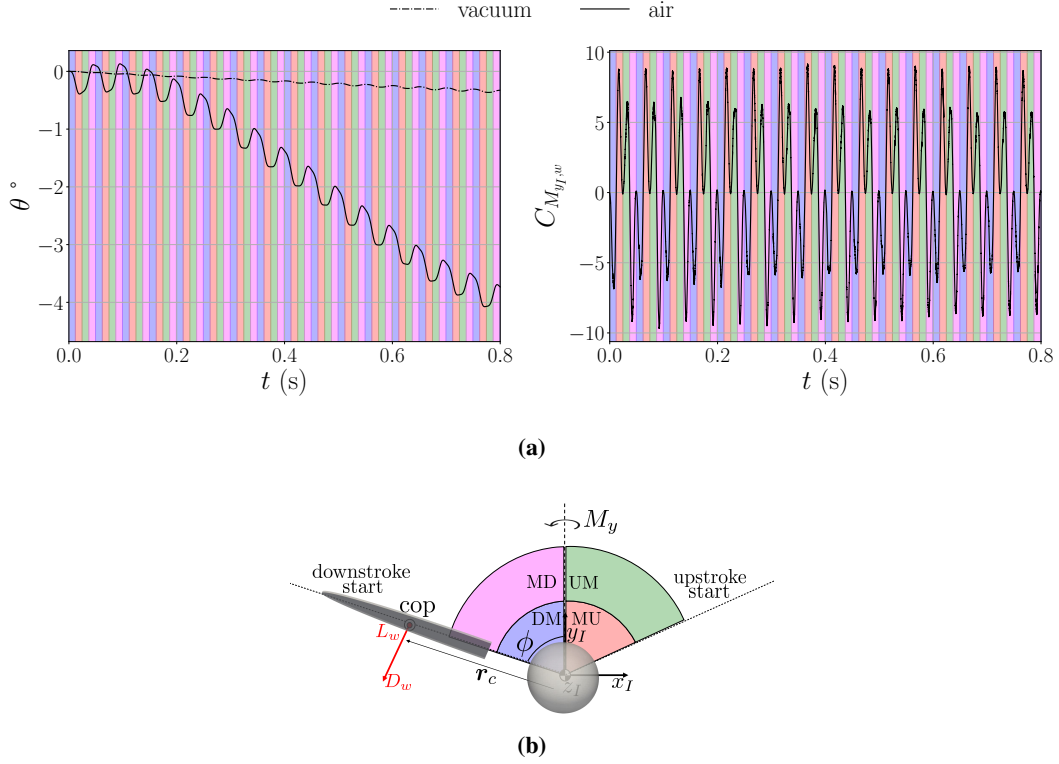


Figure 14. Body pitching dynamics shown with (a) the body pitching angle and the wing pitching torque, and (b) a schematic of the drone and the wing forces.

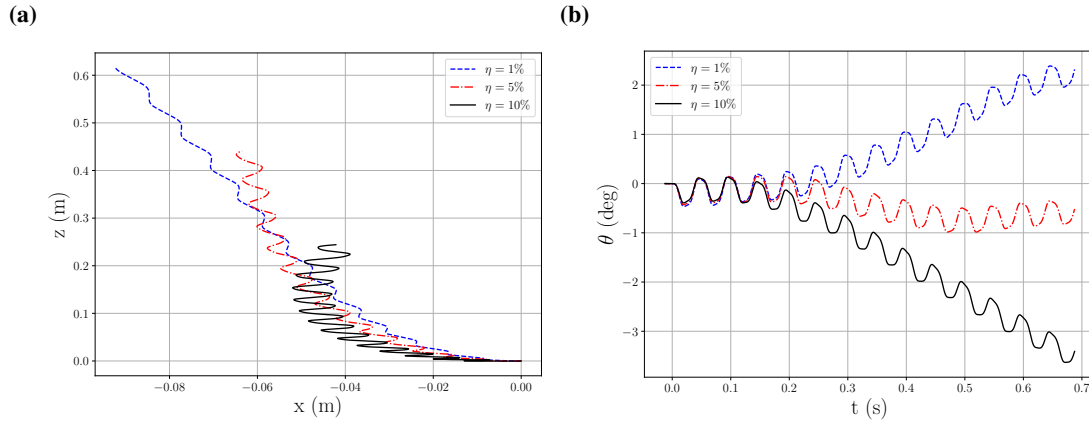


Figure 15. Influence on the wing to body mass ratio η on (a) the trajectory and (b) the time evolution of the body pitch angle.

explained in Section 4.1, the drone was simulated with only one wing and the second wing shown in Fig. 16 is its mirror image across the $(x, z)_I$ symmetry plane.

The figure first evidences the longitudinal oscillations of the body due to the wing-body inertial coupling and aerodynamic forces. While the tilting of the stroke plane is too small to be detected, its effects are seen on the $x - z$ trajectory. The drone initially moves towards the $x < 0$ ($z \sim \in [0, 0.1]$) due to the wing drag as discussed in Section 6.1. As the magnitude of the pitching angle and thus the stroke plane angle increases (see Fig. 14), the drone moves in the opposite direction due to the redirection of the lift force in the horizontal direction.

This effect is well captured by Fig. 17, which shows the vertical force during the second, fifth, tenth, and seventeenth flapping cycles. The forces are similar during the second and fifth cycles, with the fifth cycle showing more lift oscillations due to the wing-wake interaction effect arising when the flow is developing. However, from the fifth to the seventeenth cycle, the peak force increases in the downstroke and decreases in the upstroke. This results from the body pitching motion with time, setting the stroke plane more vertical and so increasing the angle

of attack during the downstrokes and decreasing it during the upstrokes. One can notice that this effect is stronger during the upstrokes.

Figure 16 also shows the pressure distribution on the drone's surface. The wings experience the strongest (negative) pressure on their suction side, confirming that this side drives the drone upwards. During the flight, the mean pressure slightly decreases while the pressure distributions remain similar. Figure 18 focuses on midpoints of the upstrokes corresponding to $t' = [0.75, 9.75, 18.75]$. Figure 18a shows the pressure distribution on the suction side, Fig. 18b on the pressure side and Fig. 18c relates the pressure distribution to the LEV using iso-contours of the Q-criterion ($Q > 0$) [76].

The pressure distributions on the pressure side are almost identical, with a larger pressure near the tip due to the highest wing velocity. On the suction side, the low-pressure region forms a triangular shape which is the fingerprint of the leading-edge vortex (LEV) as seen in Fig. 18c. The direct connection between the LEV and the pressure surface confirms the well-established understanding that, for smooth flapping motions, the LEV is the primary contributor to force generation and that its circulation drives the lift and drag evolution during a cycle [47, 48, 77].

The LEV is stably attached to the wing and grows along the span due to its dependence on the flapping velocity. The LEV core pressure decreases as the drone ascends due to the decrease of the effective angle of attack and the increase of the pitch angle magnitude. Figure 19 confirms this trend with the LEV (pseudo) circulation Γ computed as:

$$\Gamma = \int_{V': Q > 0 \cap \omega_{yw} \geq 0} \omega_{yw} dV' \quad \text{with} \quad \begin{cases} \omega_{yw} > 0 & \text{upstroke} \\ \omega_{yw} < 0 & \text{downstroke} \end{cases} \quad (31)$$

and shown as an averaged value for each cycle. As the drone ascends, the circulation decreases.

Figure 20 shows three clipped slices of the moving domain that are symmetry planes of the drone taken at the midstroke of the second, eleventh, and nineteenth cycles. The wake develops along the vertical direction and two patterns corresponding to the current and previous cycle can be identified. The previous wake structures are smeared out in the domain due to the largest cells used further away from the drone.

Finally, Fig. 21 shows the evolution of the vortex structures during the first cycle. The structures appear thicker than those shown in Fig. 18c as the contours of the Q-criterion were computed on the background grid to show the full length of the structures. In contrast to the previous analyses, the mid-position of the upstroke is taken as the beginning ($t' = 0$). Shortly after the start, the wings show three clear clockwise vortex structures forming a loop. The leading-edge vortex (LEV) sheds parts of its vorticity through a tip vortex (TV), and the body-wing clearance allows the development of a root vortex (RV). The trailing-edge vortex, shed shortly after the wing starts in the wake, connects these three structures. At stroke reversal, the wing decreases its pitching angle (i.e. wing becomes more vertical) and the vortex loop detaches from the wing. For the next stroke, the wing accelerates in the other direction and generates a new attached LEV, TV and RV forming a second vortex loop closed by the previously shed vortices.

7. Conclusion

This work characterizes and extends the rigid body dynamics solver in OpenFOAM® v2206 to enable high-fidelity simulations of flapping-wing drones. The solver is based on the Articulated-Body Algorithm (ABA), which integrates the equations of motion for multibody tree systems such as those found in flapping-wing configurations. To allow a subset of joints to follow prescribed motions while others evolve freely, we introduced active articulations by extending the ABA and modifying the governing equations of motion.

These developments are encapsulated in a new *imposedMotion* library, which supports hybrid multibody simulations with both imposed and free motions—capabilities that were previously unavailable in OpenFOAM®. While motivated by flapping-wing drones, the framework is broadly applicable to other multibody systems, including biological flyers, propeller-driven vehicles, and marine platforms.

The framework was first verified using a double pendulum benchmark derived from Lagrangian mechanics. It was then applied to two drone configurations: a single-wing drone and a body-wing drone with up to five degrees of freedom. Simulations of ascending flight were performed using the *overPimpleDyMFoam* solver with overset meshes, a moving background grid, and an LES turbulence model. Drone trajectories showed good agreement with quasi-steady predictions, and parametric studies confirmed the numerical robustness of the solver. A detailed profiling analysis identified the overset grid update as the main computational bottleneck.

Flow visualizations and force analyses revealed strong oscillatory behaviour in the body-wing drone, driven by the coupling between wing kinematics and body inertia. These results highlight the importance of resolving both aerodynamic and inertial effects in free-flight simulations.

The present work also opens several directions for future development. The computational cost associated with the overset method, in particular hole-cutting and donor-cell search operations, motivates further optimization

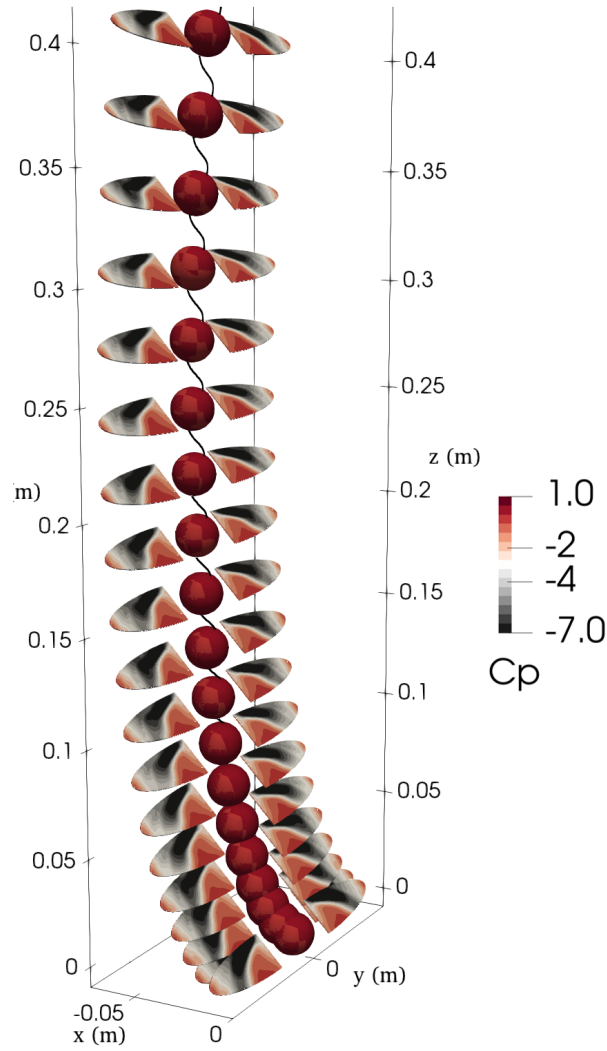


Figure 16. Pressure coefficient on the body-wing drone shown at midpoints of the upstrokes during its vertical ascent trajectory

efforts to enable simulations on larger grids and over longer time horizons. In addition, while the proposed framework has been verified using numerical benchmarks and simplified models, generating new experimental data for free-flight using existing flapping-wing drones would allow for more comprehensive validation. Finally, the flapping-wing drone model considered here remains a simplified representation of real platforms. Future work will (1) consider flexible wings by coupling the CFD framework with a structural solver from Kratos Multiphysics via the CoCoNuT code [78, 79], and (2) consider dynamic manoeuvres using wing kinematics optimised for various objectives with simplified models.

With these enhancements, the proposed simulation framework could serve as a digital companion to real flapping-wing drones—bridging the gap between biological flight performance and current engineering designs, which have so far relied on limited CFD modeling.

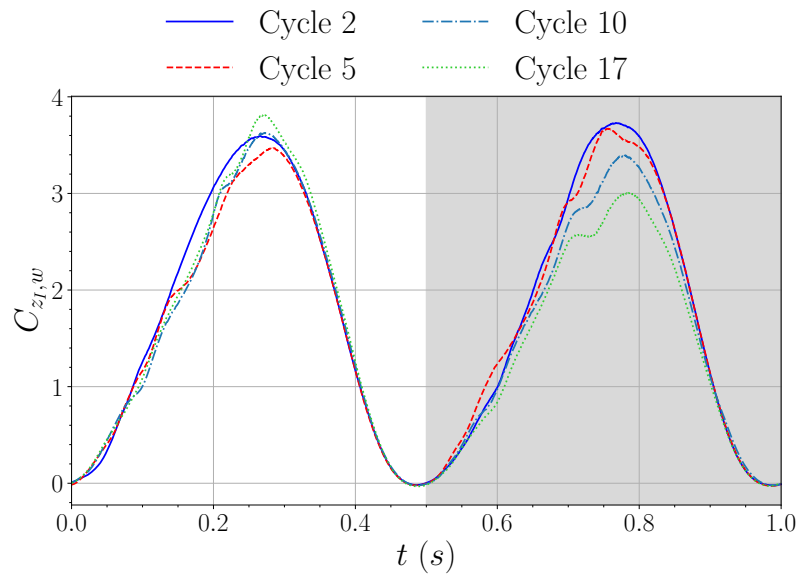


Figure 17. Lift force during the second, fifth, tenth, and seventeenth flapping cycles.

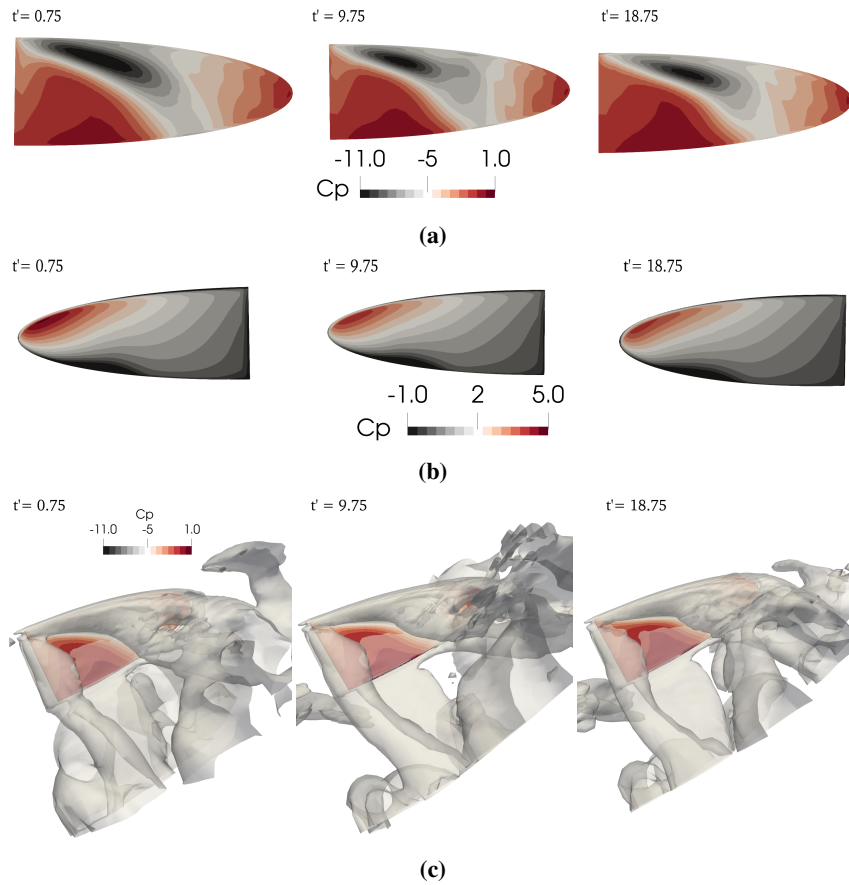


Figure 18. Pressure distribution on (a) the suction side and (b) the pressure side of the wing at midpoints of the upstrokes for the first, ninth and eighteenth cycles. The LEV is shown in (c) using the isocontours $Q > 0$ and $\omega_{yw} > 0$.

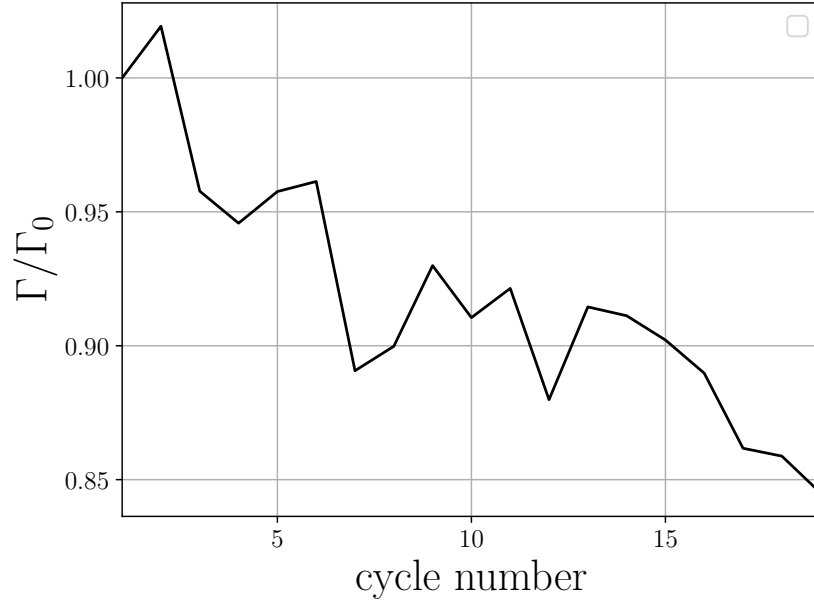


Figure 19. Evolution of the mean pseudo circulation during subsequent flapping cycles of the drone vertical ascent trajectory.

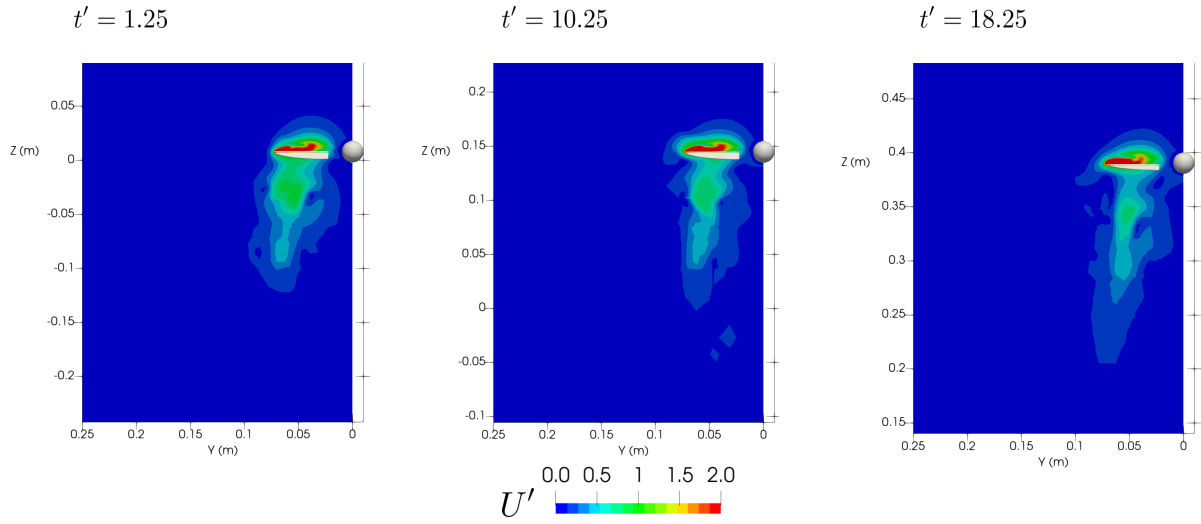


Figure 20. Wake formation evidenced by the dimensionless velocity $U' = U/U_{ref}$ field during the drone vertical motion for three time steps.

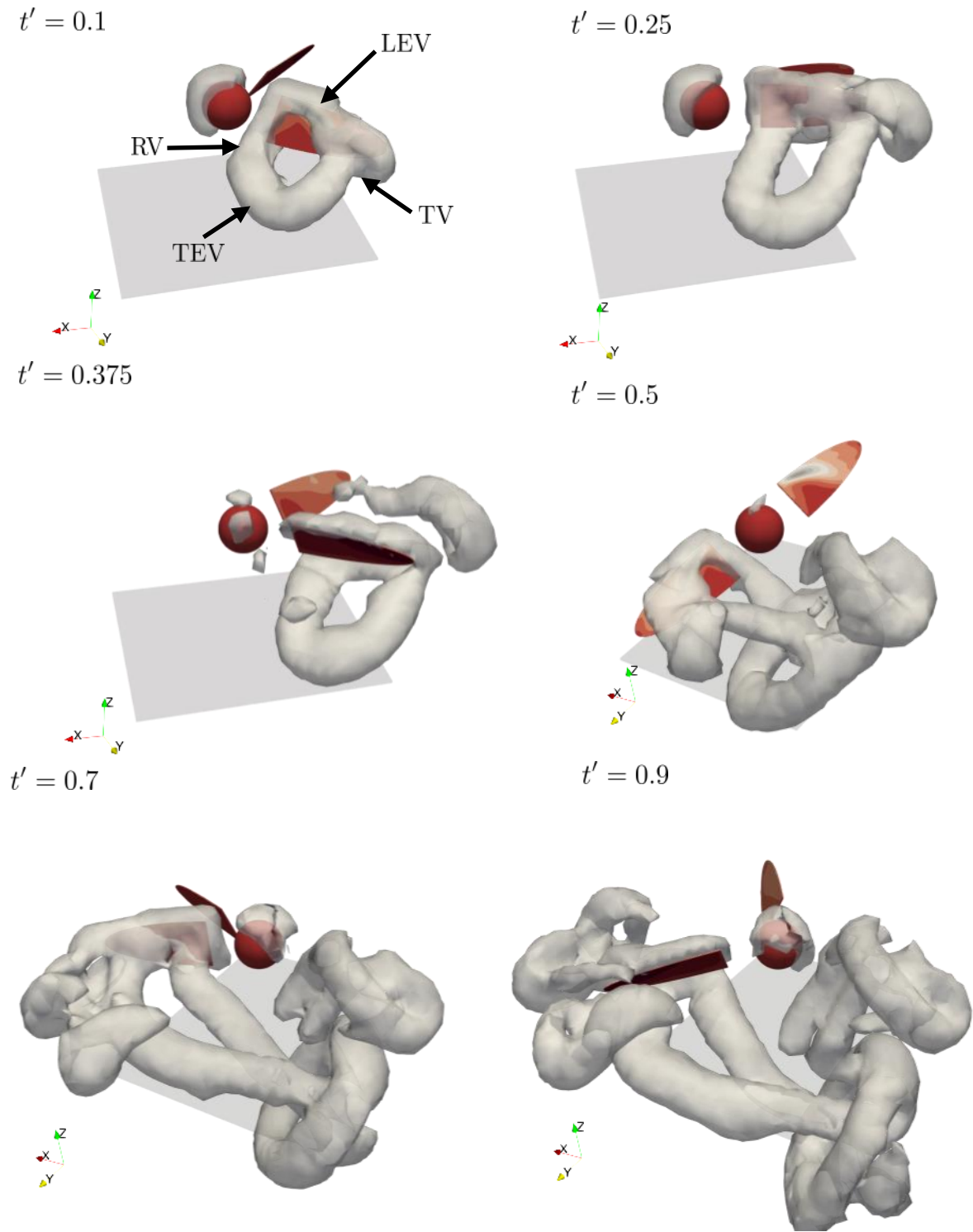


Figure 21. Vortex structures generated by the body-wing drone during the first cycle. LEV: leading-edge vortex, TEV: trailing-edge vortex, TV: tip vortex, RV: root vortex.

Acknowledgements

Romain Poletti was supported by Fonds Wetenschappelijk Onderzoek (FWO), Project No. 1SD7823N.

Author Contributions: Conceptualisation, R.P., E.B., L.K., J.D., M.A.M; methodology, R.P., E.B., L.K., J.D., M.A.M; software, R.P., E.B.; validation, R.P.; formal analysis, R.P.; investigation, R.P.; resources, L.K., J.D.; data curation, R.P.; writing—original draft preparation, R.P., L.K., J.D., M.A.M; writing—review and editing, R.P., L.K., J.D., M.A.M; visualisation, R.P.; supervision, L.K., J.D., M.A.M; project administration, J.D., M.A.M; funding acquisition, L.K., J.D., M.A.M. All authors have read and agreed to the published version of the manuscript.

References

- [1] B. Cheng, B. W. Tobalske, D. R. Powers, T. L. Hedrick, S. M. Wethington, G. T. Chiu, and X. Deng, “Flight mechanics and control of escape manoeuvres in hummingbirds. i. flight kinematics,” *Journal of Experimental Biology*, vol. 219, no. 22, pp. 3518–3531, 2016.
- [2] G. Taylor and A. Thomas, “Animal flight dynamics ii. longitudinal stability in flapping flight,” *Journal of theoretical biology*, vol. 214, no. 3, pp. 351–370, 2002.
- [3] M. Karásek and A. Preumont, “Flapping flight stability in hover: A comparison of various aerodynamic models,” *International Journal of Micro Air Vehicles*, vol. 4, no. 3, pp. 203–226, 2012.
- [4] L. Schenato, D. Campolo, and S. Sastry, “Controllability issues in flapping flight for biomimetic micro aerial vehicles (MAVs),” in *42nd IEEE International Conference on Decision and Control (IEEE Cat. no. 03CH37475)*, vol. 6. IEEE, 2003, pp. 6441–6447.
- [5] X. Deng, L. Schenato, and S. S. Sastry, “Flapping flight for biomimetic robotic insects: Part ii-flight control design,” *IEEE transactions on robotics*, vol. 22, no. 4, pp. 789–803, 2006.
- [6] H. Aono, F. Liang, and H. Liu, “Near-and far-field aerodynamics in insect hovering flight: an integrated computational study,” *Journal of Experimental Biology*, vol. 211, no. 2, pp. 239–257, 2008.
- [7] Y. Liu, A. D. Lozano, T. L. Hedrick, and C. Li, “Comparison of experimental and numerical studies on the flow structures of hovering hawkmoths,” *Journal of Fluids and Structures*, vol. 107, p. 103405, 2021.
- [8] J. Song, H. Luo, and T. L. Hedrick, “Three-dimensional flow and lift characteristics of a hovering ruby-throated hummingbird,” *Journal of The Royal Society Interface*, vol. 11, no. 98, p. 20140541, 2014.
- [9] R. J. Bomphrey, T. Nakata, N. Phillips, and S. M. Walker, “Smart wing rotation and trailing-edge vortices enable high frequency mosquito flight,” *Nature*, vol. 544, no. 7648, pp. 92–95, 2017.
- [10] X. Cai, D. Kolomenskiy, T. Nakata, and H. Liu, “A CFD data-driven aerodynamic model for fast and precise prediction of flapping aerodynamics in various flight velocities,” *Journal of Fluid Mechanics*, vol. 915, p. A114, 2021.
- [11] H. Luo, H. Dai, P. J. F. de Sousa, and B. Yin, “On the numerical oscillation of the direct-forcing immersed-boundary method for moving boundaries,” *Computers & Fluids*, vol. 56, pp. 61–76, 2012.
- [12] L. Zheng, T. Hedrick, and R. Mittal, “A comparative study of the hovering efficiency of flapping and revolving wings,” *Bioinspiration & biomimetics*, vol. 8, no. 3, p. 036001, 2013.
- [13] F. M. Bos, B. W. van Oudheusden, and H. Bijl, “Radial basis function based mesh deformation applied to simulation of flow around flapping wings,” *Computers & Fluids*, vol. 79, pp. 167–177, 2013.
- [14] S. Deng, T. Xiao, B. van Oudheusden, and H. Bijl, “A dynamic mesh strategy applied to the simulation of flapping wings,” *International Journal for Numerical Methods in Engineering*, vol. 106, no. 8, pp. 664–680, 2016.
- [15] N. Delaissé, T. Demeester, D. Fauconnier, and J. Degroote, “Comparison of different quasi-newton techniques for coupling of black box solvers,” in *14th World Congress on Computational Mechanics/8th European Congress on Computational Methods in Applied Sciences and Engineering*, 2021.
- [16] K. Yeo, S. Ang, and C. Shu, “Simulation of fish swimming and manoeuvring by an SVD-GFD method on a hybrid meshfree-Cartesian grid,” *Computers & Fluids*, vol. 39, no. 3, pp. 403–430, 2010.
- [17] C. Zhang, T. L. Hedrick, and R. Mittal, “An integrated study of the aeromechanics of hovering flight in perturbed flows,” *AIAA Journal*, vol. 57, no. 9, pp. 3753–3764, 2019.
- [18] J. H. Wu, Y. L. Zhang, and M. Sun, “Hovering of model insects: simulation by coupling equations of motion with Navier–Stokes equations,” *Journal of Experimental Biology*, vol. 212, no. 20, pp. 3313–3329, 2009.
- [19] H. Liu, T. Nakata, N. Gao, M. Maeda, H. Aono, and W. Shyy, “Micro air vehicle-motivated computational biomechanics in bio-flights: aerodynamics, flight dynamics and maneuvering stability,” *Acta Mechanica Sinica*, vol. 26, no. 6, pp. 863–879, 2010.
- [20] D. Wu, K. Yeo, and T. Lim, “A numerical study on the free hovering flight of a model insect at low Reynolds number,” *Computers & Fluids*, vol. 103, pp. 234–261, 2014.
- [21] Y. Yao and K. Yeo, “Longitudinal free flight of a model insect flyer at low Reynolds number,” *Computers & Fluids*, vol. 162, pp. 72–90, 2018.
- [22] J. Yao and K. Yeo, “Free hovering of hummingbird hawkmoth and effects of wing mass and wing elevation,” *Computers & Fluids*, vol. 186, pp. 99–127, 2019.
- [23] H. E. Taha, M. R. Hajj, and A. H. Nayfeh, “Flight dynamics and control of flapping-wing MAVs: a review,” *Nonlinear Dynamics*, vol. 70, pp. 907–939, 2012.
- [24] C. T. Orlowski and A. R. Girard, “Modeling and simulation of nonlinear dynamics of flapping wing micro air vehicles,” *AIAA journal*, vol. 49, no. 5, pp. 969–981, 2011.
- [25] B. Etkin and L. D. Reid, *Dynamics of flight: stability and control*. John Wiley & Sons, 1995.
- [26] M. Vanella, “A fluid structure interaction strategy with application to low Reynolds number flapping flight,” Ph.D. dissertation, University of Maryland, 2010.
- [27] K. Bakhshaei, H. MoradiMaryamnegari, S. SalavatiDezfouli, A. M. Khoshnood, and M. Fathali, “Multi-physics simulation of an insect with flapping wings,” *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 235, no. 10, pp. 1318–1339, 2021.
- [28] Y. Xue, X. Cai, R. Xu, and H. Liu, “Wing kinematics-based flight control strategy in insect-inspired flight systems: Deep reinforcement learning gives solutions and inspires controller design in flapping mavs,” *Biomimetics*, vol. 8, no. 3, p. 295, 2023.

- [29] S. Biswal, M. Mignolet, and A. A. Rodriguez, "Modeling and control of flapping wing micro aerial vehicles," *Bioinspiration & biomimetics*, vol. 14, no. 2, p. 026004, 2019.
- [30] H. Chen, T. A. Medina, and J. L. Cercos-Pita, "CFD simulation of multiple moored floating structures using openfoam: An open-access mooring restraints library," *Ocean Engineering*, vol. 303, p. 117697, 2024.
- [31] A. Karola, S. Tavakoli, T. Mikkola, J. Matusiak, and S. Hirdaris, "The influence of wave modelling on the motions of floating bodies," *Ocean Engineering*, vol. 306, p. 118067, 2024.
- [32] N. Docquier, A. Poncelet, and P. Fisette, "Robotran: a powerful symbolic generator of multibody models," *Mechanical Sciences*, vol. 4, no. 1, pp. 199–219, 2013.
- [33] W. Hu, D. W. Marhefka, and D. E. Orin, "Hybrid kinematic and dynamic simulation of running machines," *IEEE transactions on robotics*, vol. 21, no. 3, pp. 490–497, 2005.
- [34] R. Featherstone, *Rigid body dynamics algorithms*. Springer, 2014.
- [35] H. Hadzic, "Development and application of finite volume method for the computation of flows around moving bodies on unstructured, overlapping grids," Ph.D. dissertation, 2006.
- [36] J. H. Ferziger and M. Perić, *Computational methods for fluid dynamics*. Springer, 2002.
- [37] P. Flores, *Concepts and formulations for spatial multibody dynamics*. Springer, 2015.
- [38] "Rigid body dynamics library," https://www.openfoam.com/documentation/guides/latest/api/group__grpRigidBodyDynamics.html, [Online; accessed 15-June-2024].
- [39] B. V. Mirtich, *Impulse-based dynamic simulation of rigid body systems*. University of California, Berkeley, 1996.
- [40] N. M. Newmark, "A method of computation for structural dynamics," *Journal of the engineering mechanics division*, vol. 85, no. 3, pp. 67–94, 1959.
- [41] A. Dullweber, B. Leimkuhler, and R. McLachlan, "Symplectic splitting methods for rigid body molecular dynamics," *The Journal of chemical physics*, vol. 107, no. 15, pp. 5840–5851, 1997.
- [42] J. Crank and P. Nicolson, "A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type," in *Mathematical proceedings of the Cambridge philosophical society*, vol. 43, no. 1. Cambridge University Press, 1947, pp. 50–67.
- [43] J.-K. Kim and J.-H. Han, "A multibody approach for 6-DOF flight dynamics and stability analysis of the hawkmoth *Manduca sexta*," *Bioinspiration & biomimetics*, vol. 9, no. 1, p. 016011, 2014.
- [44] D. D. Chandar, B. Boppana, and V. Kumar, "A comparative study of different overset grid solvers between openfoam, starccm+ and ansys-fluent," in *2018 AIAA Aerospace Sciences Meeting*, 2018, p. 1564.
- [45] "OverPimpleDyMFoam," <https://openfoamwiki.net/index.php/OverPimpleDyMFoam>, 2022, [Online; accessed 1-May-2024].
- [46] D. L. Altshuler, E. M. Quicazán-Rubio, P. S. Segre, and K. M. Middleton, "Wingbeat kinematics and motor control of yaw turns in Anna's hummingbirds (*Calypte anna*)," *Journal of experimental biology*, vol. 215, no. 23, pp. 4070–4084, 2012.
- [47] R. Poletti, A. Calado, L. K. Koloszar, J. Degroote, and M. A. Mendez, "On the unsteady aerodynamics of flapping wings under dynamic hovering kinematics," *Physics of Fluids*, vol. 36, no. 8, 2024.
- [48] Y. Lee, K.-B. Lua, T. Lim, and K. Yeo, "A quasi-steady aerodynamic model for flapping flight with improved adaptability," *Bioinspiration & biomimetics*, vol. 11, no. 3, p. 036005, 2016.
- [49] D. D. Chin and D. Lentink, "Flapping wing aerodynamics: from insects to vertebrates," *Journal of Experimental Biology*, vol. 219, no. 7, pp. 920–932, 2016.
- [50] G. J. Berman and Z. J. Wang, "Energy-minimizing kinematics in hovering insect flight," *Journal of fluid mechanics*, vol. 582, pp. 153–168, 2007.
- [51] Z. Yan, H. E. Taha, and M. R. Hajj, "Effects of aerodynamic modeling on the optimal wing kinematics for hovering mavs," *Aerospace Science and Technology*, vol. 45, pp. 39–49, 2015.
- [52] Y. Bayiz, M. Ghanaatpishe, H. Fathy, and B. Cheng, "Hovering efficiency comparison of rotary and flapping flight for rigid rectangular wings via dimensionless multi-objective optimization," *Bioinspiration & biomimetics*, vol. 13, no. 4, p. 046002, 2018.
- [53] S. S. Bhat, J. Zhao, J. Sheridan, K. Hourigan, and M. C. Thompson, "Effects of flapping-motion profiles on insect-wing aerodynamics," *Journal of Fluid Mechanics*, vol. 884, p. A8, 2020.
- [54] C. Ellington, "The aerodynamics of hovering insect flight. iii. kinematics," *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, vol. 305, no. 1122, pp. 41–78, 1984.
- [55] J. W. Kruyt, E. M. Quicazán-Rubio, G. F. Van Heijst, D. L. Altshuler, and D. Lentink, "Hummingbird wing efficacy depends on aspect ratio and compares with helicopter rotors," *Journal of the Royal Society Interface*, vol. 11, no. 99, p. 20140585, 2014.
- [56] F. M. Bos, B. W. van Oudheusden, and H. Bijl, "Wing performance and 3-D vortical structure formation in flapping flight," *Journal of Fluids and Structures*, vol. 42, pp. 130–151, 2013.
- [57] T. Nakata, H. Liu, and R. J. Bomphrey, "A CFD-informed quasi-steady model of flapping-wing aerodynamics," *Journal of Fluid Mechanics*, vol. 783, pp. 323–343, 2015.
- [58] C. Badrya, "CFD/Quasi-Steady coupled trim analysis of diptera-type flapping wing MAV in steady flight," Ph.D. dissertation, University of Maryland, College Park, 2016.
- [59] A. Calado, R. Poletti, L. K. Koloszar, and M. A. Mendez, "A robust data-driven model for flapping aerodynamics under different hovering kinematics," *Physics of Fluids*, vol. 35, no. 4, 2023.
- [60] M. Alletto, "Comparison of overset mesh with morphing mesh: flow over a forced oscillating and freely oscillating 2d cylinder," *OpenFOAM® Journal*, vol. 2, pp. 13–30, 2022.
- [61] M. Coe and S. Gutschmidt, "Ika-flow: A flexible body overset mesh implementation for fish swimming," *OpenFOAM® Journal*, vol. 3, pp. 75–119, 2023.
- [62] K. Ahmed, A. A. Hamada, L. Chatellier, and M. Furth, "A Modified Overset Method in OpenFOAM for Simultaneous Motion and Deformation: A Case Study of a Flexible Flapping Foil," *OpenFOAM® Journal*, vol. 4, pp. 41–61, 2024.
- [63] T. Stachowiak and T. Okada, "A numerical analysis of chaos in the double pendulum," *Chaos, Solitons & Fractals*, vol. 29, no. 2, pp. 417–422, 2006.
- [64] A. Calvão and T. Penna, "The double pendulum: a numerical study," *European Journal of Physics*, vol. 36, no. 4, p. 045018, 2015.
- [65] S. Herho, F. Fajary, K. Herho, I. Anwar, R. Suwarman, and D. E. Irawan, "Reappraising double pendulum dynamics across multiple computational platforms," 2024.
- [66] S. Schubert and T. Rung, "Challenges and applications of overset grid coupling strategies," *PAMM*, vol. 17, no. 1, pp. 139–142, 2017.

- [67] S. P. Sane and M. H. Dickinson, “The aerodynamic effects of wing rotation and a revised quasi-steady model of flapping flight,” *Journal of experimental biology*, vol. 205, no. 8, pp. 1087–1096, 2002.
- [68] L. Schena, P. Marques, R. Poletti, S. Ahizi, J. V. d. Berghe, and M. A. Mendez, “Reinforcement twinning: from digital twins to model-based reinforcement learning,” *arXiv preprint arXiv:2311.03628*, 2023.
- [69] Y. Lee and K. B. Lua, “Wing–wake interaction: comparison of 2D and 3D flapping wings in hover flight,” *Bioinspiration & Biomimetics*, vol. 13, no. 6, p. 066003, 2018.
- [70] H. Li and M. R. Nabawy, “Capturing wake capture: a 2D numerical investigation into wing–wake interaction aerodynamics,” *Bioinspiration & Biomimetics*, vol. 17, no. 6, p. 066015, 2022.
- [71] W. G. Van Veen, J. L. Van Leeuwen, B. W. Van Oudheusden, and F. T. Muijres, “The unsteady aerodynamics of insect wings with rotational stroke accelerations, a systematic numerical study,” *Journal of Fluid Mechanics*, vol. 936, p. A3, 2022.
- [72] H. E. Taha, M. R. Hajj, and P. S. Beran, “State-space representation of the unsteady aerodynamics of flapping flight,” *Aerospace Science and Technology*, vol. 34, pp. 1–11, 2014.
- [73] L.-G. Liu, G. Du, and M. Sun, “Aerodynamic-force production mechanisms in hovering mosquitoes,” *Journal of Fluid Mechanics*, vol. 898, p. A19, 2020.
- [74] C. Windt, J. Davidson, B. Akram, and J. V. Ringwood, “Performance assessment of the overset grid method for numerical wave tank experiments in the openfoam environment,” in *International Conference on Offshore Mechanics and Arctic Engineering*, vol. 51319. American Society of Mechanical Engineers, 2018, p. V010T09A006.
- [75] “Overset implementation,” <https://github.com/louisgag/openFoam-Overset-SpeedUp>, 2022, [Online; accessed 15-June-2024].
- [76] J. Jeong and F. Hussain, “On the identification of a vortex,” *Journal of Fluid Mechanics*, vol. 285, pp. 69–94, 02 1995.
- [77] L. Chen, J. Wu, and B. Cheng, “Leading-edge vortex formation and transient lift generation on a revolving wing at low Reynolds number,” *Aerospace Science and Technology*, vol. 97, p. 105589, 2020.
- [78] “kratos,” <https://github.com/KratosMultiphysics/Kratos>, [Online; accessed 18-June-2024].
- [79] “coconut,” <https://pyfsi.github.io/coconut/>, [Online; accessed 18-June-2024].