

RAM

● ROBOTICS
AND
MECHATRONICS

A GEOMETRIC PORT-HAMILTONIAN MODELLING AND SIMULATION FRAMEWORK FOR MORPHING-WING UAVS

J. (Jorge) Raven Garcia

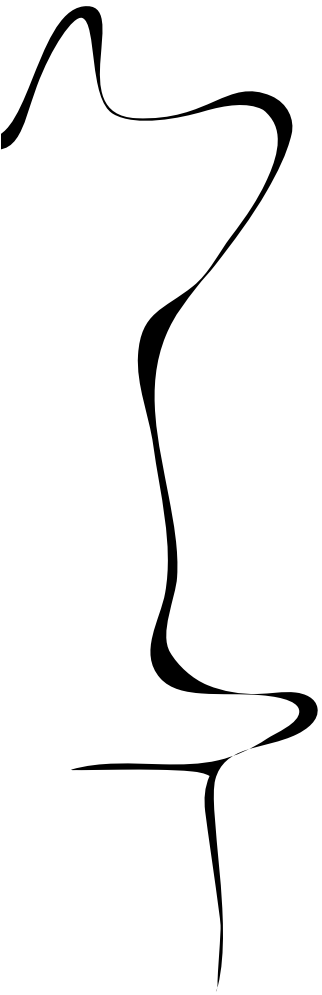
MSC ASSIGNMENT

Committee:

prof. dr. ir. S. Stramigioli
ir. R.S.M. Sneep
dr. ir. R.A.M. Rashad Hashem
dr. ir. R.G.K.M. Aarts

February, 2023

007RaM2023
Robotics and Mechatronics
EEMCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands



Summary

In the last decade the amount of UAVs has increased dramatically. Their usefulness has been greatly proven. Nevertheless, as long as tasks become more difficult, robots become more complex, including their interactions with the environment. In a similar way, flapping-wing robots have experienced a considerable growth since there is a big gap for them to fill. Even though flapping flight is still very little understood, these robots would be better suited for many tasks, as natural flyers have optimized their flapping flight during millions of years of evolution. However, developing flapping-wing robots would require a new modelling and simulation framework in order to keep up with the increasing complexity of these new aerial robot designs and dynamics.

Along this project a research of the state of the art of morphing-wing UAVs is realized. More exhaustively for the cases of flapping wings UAVs. A geometric port-Hamiltonian approach is developed to create a modular framework for the modelling and simulation of generic morphing-wing UAVs. A family-tree (parent-child) modelling method using port-Hamiltonian structures is designed and later-on applied for modelling and simulating a set of case studies morphing UAV present in the literature with satisfactory results. The framework consist on three basic elements only, a parent, a child and a joint, that can be even reduced to simply a parent and compact child (child-joint) for ease of implementation. Whats more, the approach is geometric, so the dynamics can be expressed in a coordinate free manner, which together with the port-Hamiltonian characteristics make the method highly modular. Additionally, it is energy-based, thus energy consistent, and uses power as common communication language, allowing for multi-domain modelling. After all, it is demonstrated how this method can improve others in terms of efficiency, energy consistency and speed of modelling.

Finally, a set of different literature models are selected and implemented on 20-Sim, representing a broad variety of case studies where the here presented framework can become of great help, showing in parallel the ease of use and the vast amount of possibilities the geometric port-Hamiltonian framework has to offer. Even though no numerical values are presented, as the models don't include air, it is clearly demonstrated the effect of the inertia forces generated by the morphing surfaces as well as the benefits of the framework which are exalted compared to current modelling techniques and tools. Besides, the model dynamics and the energy consistency of the framework subsystems are verified.

Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
CG	Center of Gravity
CM	Center of Mass
DOF	Degree Of Freedom
EoM	Equations of Motion
GRE	Global Relation Editor
ODE	Open Dynamics Engine
PDE	Partial Diferential Equation
R&D	Research and Development
UAV	Unmaned Aerial Vehicle

Nomenclature

$Ad_{H_i^j}$	Adjoint from i to j
$ad_{T_i^{i,j}}$	adjoint of twist $T_i^{i,j}$
θ	Angular position
ω	Angular velocity
ϕ_i	Coordinate frame i
r	Distance
e	effort
f	flow
F	Force
q	Generalized coordinate
g	gravity acceleration
A_{grv}^0	Gravity acceleration vector
H_i^j	Homogeneous matrix of i wrt j
I	Identity matrix
G^i	Inertia tensor of body i
i	Inertia
J_i^j	Joint matrix of i wrt j
E	Kinetic co-energy
v	Linear velocity
m	mass
p^i	Momentum of body i
o	Position
λ	Screw displacement
τ	Torque
$T_k^{i,l}$	Twist of k wrt l expressed in frame ϕ_i
$S_k^{i,l}$	Unit twist of k wrt l expressed in frame ϕ_i
W^i	Wrench expressed in frame ϕ_i

Contents

1	Introduction	1
1.1	Context	1
1.1.1	Defining a Morphing-Wing Robot	2
1.1.2	Why Flapping-Wing Robots	2
1.1.3	Motivation for a new modelling environment and framework	4
1.1.4	Why a Geometric Port-Hamiltonian framework	5
1.2	Related Work	5
1.3	Problem Statement & Research questions	8
1.4	Proposed Solution	8
1.5	Outline	9
2	State of the Art	10
2.1	Morphing Wings. State of the art.	10
2.2	Bio-inspired Morphing-Wing UAVs. State of the art.	12
2.3	Flapping-Wing UAVs (Ornithopters). State of the art.	14
2.4	Conclusion	16
3	Mathematical Background	17
3.1	Lie Group Theory	17
3.1.1	Dual Space: Twists & Wrenches	18
3.1.2	Adjoint & adjoint	19
3.1.3	Screw Theory	19
3.1.4	Twist's Exponential	20
3.1.5	Rigid Body Dynamics	21
3.2	Port-Hamiltonian Theory	21
3.2.1	Dirac Structures	22
3.2.2	Power-ports and Signal-ports	22
3.2.3	Power and Energy Balance	22
3.3	Conclusion	23
4	The Modelling and Simulation Framework	24
4.1	Framework Overview. The Parent-Child Paradigm.	24
4.2	Parent Subsystem	24
4.3	Child Subsystem	26
4.4	Joint Subsystem	29
4.5	Compact Child (joint + child)	35
4.6	Conclusion	35

5	Morphing-Wing Robots Case Studies	37
5.1	20-Sim Environment	37
5.1.1	Generalities of the 20-Sim models	37
5.2	Flapping-Wing Base Model. Subsystems validation.	39
5.3	Gull wing	42
5.4	Transformer aircraft (Flapping + prismatic wing)	45
5.5	Fully Adaptive UAV (Sweeping + Twisting + Span-morphing wing)	48
5.6	Sweeping-Wing UAV (Flapping + Sweeping)	50
5.7	Perching UAV (Flapping + Twisting)	53
5.8	LisHawk (Flapping + Sweeping wing + Tail Morphing)	55
5.9	Robird	58
5.10	Robird with Flexible Wing: Comparison to a Bond-graph Model with Flexible Wing	61
5.11	Robotic Bat.	66
5.12	Conclusion	71
6	Conclusion	73
6.1	Conclusions	73
6.2	Limitations and Future Work Recommendations	76
A	Appendix 1: 20-Sim Implementation	78
A.1	Before building a model	78
A.2	Building a basic model	79
A.2.1	Building a model with a Child with multiple children	82
A.2.2	Building a model with multi-DOF joints	83
A.2.3	Building a model with gravity compensation	83
A.3	Global Relation Editor (GRE)	85
A.4	Parent, Child and Joint 20-Sim blocks Implementation	86
A.4.1	Parent subsystem	86
A.4.1.1	GRE Parent	86
A.4.1.2	Parent parameters declaration (User definitions)	87
A.4.2	Compact Child + Joint	88
A.4.2.1	Child subsystem	88
A.4.2.2	Joint subsystem	89
A.4.2.3	GRE compact Child	90
A.4.2.4	Child parameters declaration (User definitions)	90
A.4.3	Child with multi-dof joint	91
A.4.3.1	GRE Child with multi-dof Joint	91
A.4.3.2	Child with multi-dof joint parameters declaration (User definitions)	92
A.4.4	World / Test-bed modelling	94

A.4.4.1	GRE World Attachment	96
A.5	The Numerical Differentiation and the Algebraic Loops Problem	97
A.6	Creation of the sub-models library	99
B	Appendix 2: Extra models for testing, validation and verification.	101
B.1	Pendulum model	101
B.2	Transformer Aircraft (Flapping + Prismatic) with gravity compensation	102
	Bibliography	105

1 Introduction

1.1 Context

Unmanned Aerial Vehicles (UAVs) have lately become a very widely used tool. Anywhere around the globe one can appreciate their unstoppable growth. This is perceivable everyday in the streets, the news or the social networks. We can see people flying their quad-copters in the park or in nature, capturing the beauty of the environment from the air, or even in movies, with a noticeable increase of aerial (UAV) takes. Furthermore, it is becoming more common to see UAVs performing complex technical operations, like electrical power lines surveillance and repairing, along with many other jobs like crop health control, spraying fertilizer, delivery services and, of course, military tasks, substituting humans in the accomplishment of these difficult duties. Shapes and configurations of the UAVs can vary vastly as well. There are multi-rotors, fully-actuated, under-actuated and with actuated arms, helicopters, fixed-wings or ornithopters, which encompass all those UAVs that propel using flapping wings. Figure 1.1 present some of the afore mentioned examples.

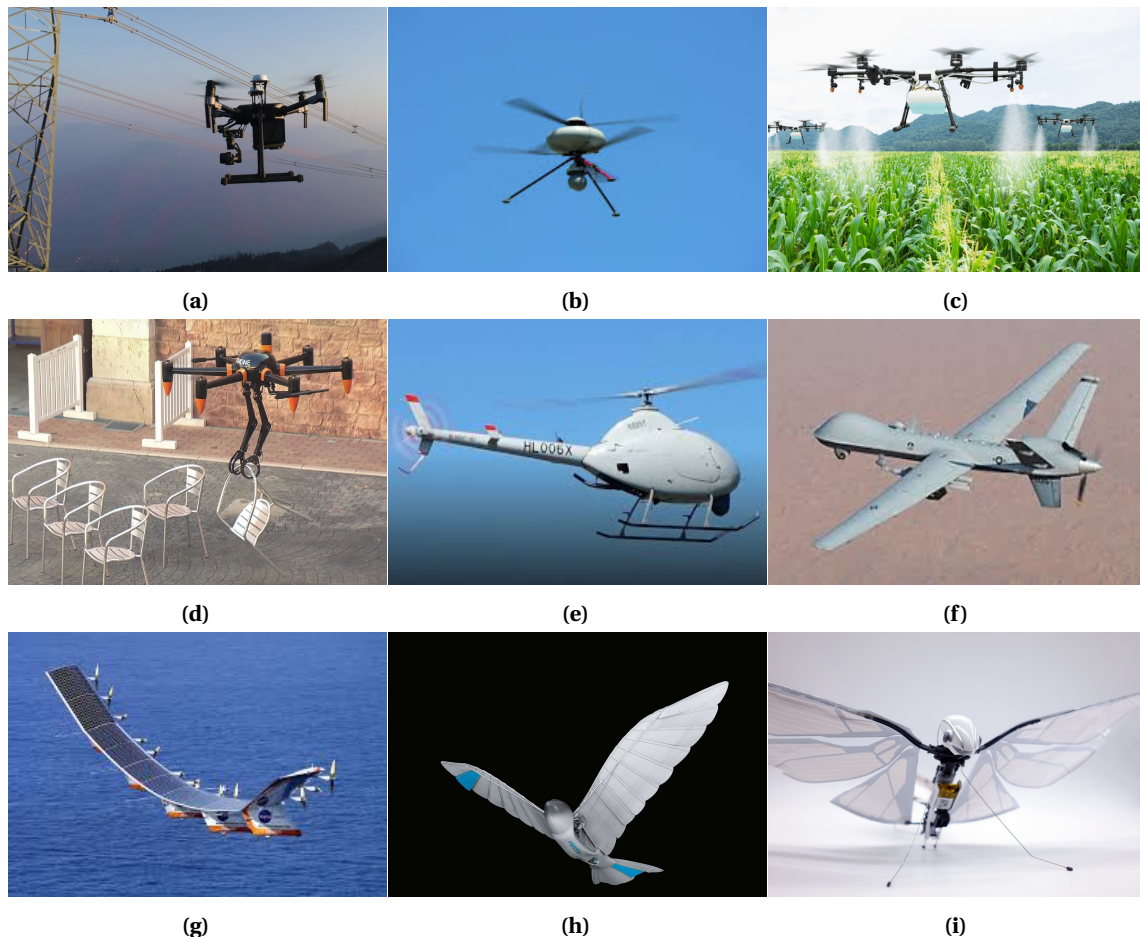


Figure 1.1: Existing UAV examples. a) Quadcopter power line inspection, b) Duocopter, c) Multirotor fertilizing, d) Multirotor with arms, e) Helicopter (military), f) Fixed Wing (military), g) Stratospheric, h) Ornithopter (bird), i) Ornithopter (insect).

Aerial robots are rapidly evolving towards more and more complex systems. The most standard solutions are the fixed-wing and the rotary-wing robots, that are easier to model and simulate thanks to the rather well understood aerodynamics of the propellers, fixed-wings and flaps. Nevertheless, problems arise with flapping wings, flexible wings and complex aerodynamic

surfaces in general. In fact, one of the main objectives of the PortWings Project [Stramigioli et al. (2018)] (ERC 2018 Advanced Grant from the Horizon 2020 research and innovation programme of the European Commission), of which this thesis is part of, is increasing our understanding of the flapping-flight. Ornithopters (flapping-wing) or morphing-wing robots comprehend very complicated aerodynamic effects, aerodynamic and inertial coupling and variable center of mass (CM). All these effects could be analyzed through experiments, but carrying out real-life experiments tend to be very costly, therefore the behaviour and performance of the aerial robots should be studied beforehand, by means of modelling and simulating digital models (digital twin). The main issue nowadays is that **developing accurate models for these complex aerial robots can be very challenging in current state of the art simulation tools, environments and frameworks**. Therefore, getting rid of this bottle-neck that is the difficulty in the development of accurate flapping-wing UAV models could help us increase our understanding on animal flight together with enabling the design of more optimal controls.

1.1.1 Defining a Morphing-Wing Robot

Morphing, in general, could be any variation in the configuration of an UAV. So, if for example we consider a fixed-wing UAV, with active flaps, the change in angle of those flaps could be considered morphing. Nonetheless, this configuration change can be neglected in terms of mass distribution variation and inertia moments when exerting the movement, because the mass of the flaps is relatively small compared to that of the whole plane. Also, the movement of one flap won't affect the aerodynamics on the other wing. Therefore, we won't consider this a morphing UAV, since we are interested in bigger morph changes that actually have a large impact in terms of either mass distribution, distributed forces, inertia momentum generation or fluid dynamics. So inside this category we would include UAVs like fixed-wing with large sweep, twist, dihedral or span morphing (see figure 1.2) in addition to ornithopters which fly using a combination of all these motions.

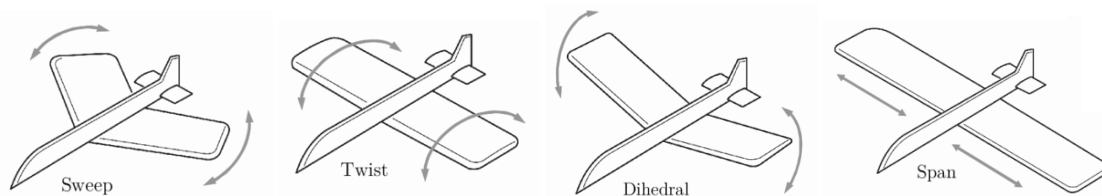


Figure 1.2: Fixed-Wing UAVs main types of morphing.

In this thesis though, we are particularly interested in developing a modelling and simulation framework to easily build ornithopter models. But why would we be interested in modelling ornithopters or flapping-wing robots in first place?

1.1.2 Why Flapping-Wing Robots

Multirotor UAVs are the most popular aerial robot solution used for civilian purposes. Nonetheless, these vehicles present some limitations, leaving some room for flapping-wing UAVs to fill.

For starters, birds are highly flexible and maneuverable animals. Even though multirotors are quite capable of maneuvering in small spaces, they are very sensitive to environment interactions, easily breaking under any slight contact. Their rigid fast rotating blades make them both vulnerable and hazardous. On the other hand, flapping-wing robots could be safer for interacting with humans and the environment in general, thanks to their lack of blades and their soft and elastic characteristics. Also, birds can fly more efficiently, as [Zheng et al. (2013)] suggest in their study, helped by their capability of gliding, which allows them to travel longer distances with a much lower energy consumption, together with their asynchronous flapping capabilities, that permits them to renounce to a vertical tail, consequently reducing air drag.

Propellers have very low efficiency, even less if the multirotor is fully-actuated due to their non-parallel configuration. Additionally, noise in propelled-UAVs is very inconvenient, contributing to increasing noise-pollution and making them annoying and uncomfortable to work or live nearby. Conversely, flapping birds are rather quiet and respectful with regards to the environment, making them also better suited for collaborating in near-human(-nature) surroundings. They are more kin to nature in general, plants and other birds, since they are flexible and light-weighted. Whats more, there are many reported bird attacks to propelled drones as mentioned by [Suzuki (2022)] and [Nunn (2021)], even though the second emphasize the potential use of UAVs for controlling and monitoring endangered bird colonies, a duty for which ornithopters would be better suited.

Flapping-wing UAVs offer a good trade-off between the capability of multirotor UAVs of taking-off vertically and the long range steady flight of fixed-wing UAVs. Actually, the blades can be considered again a disadvantage in UAVs with rotors for landing and taking-off in crowded areas for safety reasons. Birds, on the contrary, can use the already available infrastructures, like trees, buildings or lampposts where they can perch. An example of this are the UAVs developed by [Roderick et al. (2021)] and [Zufferey et al. (2022)] (see figure 1.3 a) that are a quadrotor and an ornithopter respectively, equipped with eagle-like legs for perching on branches in the same way birds would do.

Of course, there are some other more delicate applications for bird-mimicking robots, like employing them as spy drones. The use of this type of UAVs for military purposes could lead to unobtrusive surveillance. Whats more, there are already some on-going projects for using bird-like UAVs as spy drones for prosecuting the organized crime [Wilkinson (2022)] [Bisht (2022)]. Such robots would hide in plain sight.

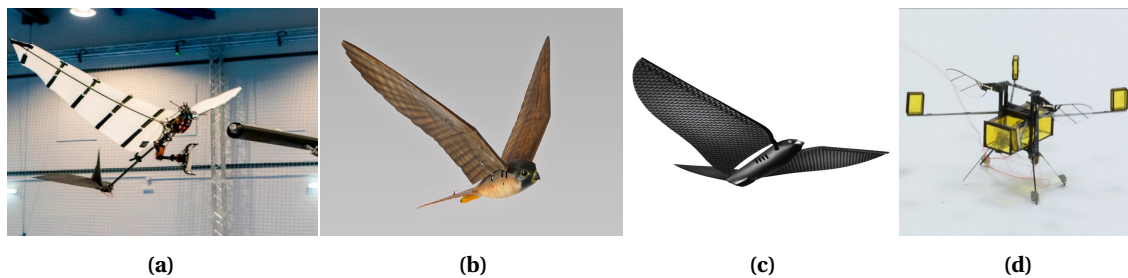


Figure 1.3: Aerial animal mimicking robot examples. a) Perching ornithopter. Zufferey et al. (2022), b) Bird pests control (Robird) Folkertsma et al. (2017), c) Spy UAV (Metabird) Bisht (2022), d) Insect UAV (Robotbee) Jafferis et al. (2019).

Insect-like UAVs could be of great use as well. These tiny robots could be used for tasks like inoculating vaccines in humans in rural areas or pollinating plantations. Actually, insects are responsible of pollinating 80% of the plants and crops [Ollerton et al. (2011)] and, as an example, bees population is decreasing dramatically [Nearman and Van Engelsdorp (2022)]. The actual problem of these tiny robots, as mentioned by [Jafferis et al. (2019)], is the too large amount of power they require for flying with such a small size (lift-power ratio), that they try to solve attaching small solar panels.

Another use for robotic birds is the control of bird pests. Nowadays airports use birds of pray to scare other birds and clear the sky near the runway to avoid aeroplane accidents. Such important task can be executed with robotic birds like The Drone Bird Company pretend to do with the Robird [Folkertsma et al. (2017)], a falcon-like robot.

Despite the just mentioned advantages of flapping-wing UAVs, there is a big drawback and a reason why these robots are not much developed yet, which is the complexity of their mechanical design, aerodynamics, flight dynamics and ultimately the complexity in the design of the

governing control system, which is why Festo claimed “Festo has succeeded in deciphering the flight of birds” [Festo AG & Co. KG (2011)] when they launched the SmartBird to the market back in 2011 (see figure 1.1h).

It is here where this thesis comes to help, to **contribute to the development of these complex aerial robots through suitable modelling and simulation tools.**

1.1.3 Motivation for a new modelling environment and framework

Flying animals present a very optimized, but rather complex to understand flapping wing motion. Not only flapping mechanics and aerodynamics are difficult to understand, but also it differs immensely from one type of flyer to another. To make it even harder, knowledge is not easily transferable, since even the dimension of the bird can imply large variations in the aerodynamic effects. For instance, insects’ flapping rate is much higher than that of bats and birds. As [Shyy et al. (2010)] suggests, the flapping frequency tend to decrease when the size of the wing (half span and chord) increases. Similarly, Reynold’s number increases dramatically with the size of the flyer, deriving in a huge difference in the fluid dynamics. Wing dynamics and kinematics are completely different as well, since insects wings are very light while birds and bats have heavier wings that morph to exploit inertial effects [Berg and Rayner (1995)]. All that is a problem for designing aerial robots, but it can be tackled or at least reduced by using faithful models simulations. Modelling should require a much lesser effort in money and time than building a physical prototype since software models are easier to modify and set up for new tests and simulations. Accurate but simple modelling can help reduce designing times considerably, as well as help us understand birds flapping flight dynamics.

Any standard modelling and simulation environment is composed by the next three elements (see figure 1.4):

- The 3D rendering for realistic visualization.
- The external APIs to interface with the model.
- **The physics engine.** Subdivided in:
 - **Robot Dynamic Model.**
 - **Robot Actuators Model.**
 - Environment Model.
 - Robot Sensors Model.

The work developed next lies under the third point. A physics engine which covers the **dynamic model of the robot’s components** and the environment, as well as the relations between one each other, with a focus on the **bodies dynamic model** and the **actuators model**, but keeping in mind that the main issue for most simulation tools available today for aerial robots lies within the environment physics model context, since most tools are poorly suited to simulate physics interaction with the air. Physics are in general described in a poorly consistent manner in the literature. It is very unlikely to find energy-based physics engines, not to mention that standard tools are based on coordinate-based treatments instead of using geometric descriptions.

Thus, this thesis seeks building an **energy consistent framework where to model those highly complex aerial robots in a geometric way**, that can experiment dynamic morphings, but without the implementation of the environment. This is, there is **no** intention of implementing the **air nor realistic aerodynamics**. A framework for ornithopters which wings’ mass compared to

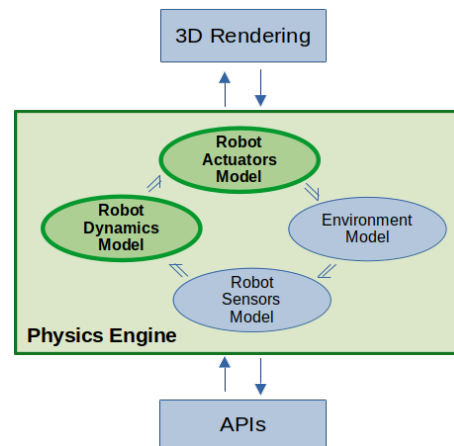


Figure 1.4: Simulator architecture.

that of the body is relatively high or, alternatively, flyers which Reynolds number is relatively high. This is, bats and birds, leaving very small birds or insects out of the equation.

1.1.4 Why a Geometric Port-Hamiltonian framework

Tools, mathematics and computational simulations have to evolve in order to be capable of dealing with the increasing robots complexity with multidisciplinary and coupled systems. New methods need to be investigated in order to make aerodynamics modelling generally applicable and UAV simulation physics more accurate.

The port-Hamiltonian theory was first introduced by Maschke and Van der Schaft (1992) as a combination of the port-based modelling approach and the geometric Hamiltonian mechanics. The port-Hamiltonian systems formulation provides a modular, physically oriented approach for modeling complex nonlinear systems, supported by the port-based modeling, where complex systems can be represented by means of simple blocks interconnected using bi-directional power-conserving energy ports. By using energy as a common language for interconnection, this approach allows the interaction of systems of different physical domains (mechanical, electrical, thermal, ...). The port-Hamiltonian formulation can also be extended to infinite-dimensional systems (PDEs), as beam and fluid equations [Cardoso Ribeiro (2016)]. Moreover, exact closed-form analytic derivatives of the dynamics can be easily obtained with respect to the configuration variables, what makes it possible to use Dirac structures. Whats more, dynamic equations can be easily transformed to coordinates of any reference frame, which make it possible to describe the robot model in a coordinate-free manner.

The motivation for using this formulation is that it is possible to describe each element of the system separately using the port-Hamiltonian systems formulation and the several subsystems can be then coupled, guaranteeing that the global system is also a port-Hamiltonian system. Additionally, energy-based methods can be used for control purposes, which are well suited for robot-environment interaction control [Duindam et al. (2009)], [Stramigioli (2015)].

Additionally, Geometric mechanics are in general better suited for modelling fluids like water or air than standard rigid-body mechanics, thus it can be better for modelling robots of which motion relies on their interaction with fluids.

1.2 Related Work

The recent works by [Hentati et al. (2018)], [Mairaj et al. (2019)] and [Gill et al. (2021)] expose the latest advances in UAV simulators. Their work enlighten the big effort put on developing environments for UAV networks simulation, computer vision and AI training and simulation, for drone pilot training or even entertainment, but more importantly for research and development of new flying robots. Inside this last group we can find many simulators that are only suited for either multirotors, fixed-wing or both, but only a few allow the modelling and simulation of more complex flying robots dynamic models.

The tendency for modelling flapping-wing UAVs in particular is to use very simplistic models of fixed-wing or multirotor UAVs as it is the case of Simbeeotic [Kate et al. (2012)], which is a simulator specifically designed for the Robotbee robot [Jafferis et al. (2019)](see figure 1.3d), which is a tiny ornithopter that doesn't require a complex modelling of its motion.

Nonetheless, there are other simulators that allow a completely unconstrained definition of the flying robot. We can limit these to Gazebo [Robotics (2002)], Webots [Cyberbotics Ltd. (1998)], CoppeliaSim [Rohmer et al. (2013)] and 20-Sim [Controllab Products (2008)]. Gazebo, Webots and CoppeliaSim can use the Open Dynamics Engine (ODE) physics engine, that employs a constraints-based modelling instead of force-based. Also Gazebo and CoppeliaSim have in common the possibility to use physics engines that represent the rigid bodies mechanics with the Featherstone's Articulated Body Algorithm (physics engines: Simbody, DART or Bullet) that

is a generalized coordinate approach that consists on an linear-time recursive articulated-body dynamics modelling method. Additionally, CoppeliaSim can run Newton, Vortex and MuJoCo physics engines. At last, 20-Sim was created for bong-graph (port-based) representations.

Simulator	Physics Engine(s)	Field of application
Webots [Cyberbotics Ltd. (1998)]	ODE	R&D and Education
CoppeliaSim [Rohmer et al. (2013)]	ODE, Bullet, Vortex, Newton, MuJoCo	R&D and Education
Gazebo [Robotics (2002)]	ODE, Bullet, Simbody, DART	Industry, R&D and Education
20-Sim [Controllab Products (2008)]	Port-Based	R&D and Education

Table 1.1: Aerial robots simulators state of the art.

As mentioned just before, ODE physics engine uses (velocity) constraint based models, that even though they are, in general, better than force based models (normally have lots of parameters to tune), they are mathematically more difficult to formulate and more expensive to compute. On the other hand, Featherstone's Articulated Body Algorithm [Baraff (1996)] [Mirtich (1996)] is good for rather simple chains. It is contrary to other methods that need a mass matrix for the system and must invert it to solve for joint accelerations at each time-step. Nevertheless it uses generalized coordinates, which can suffer a progressive drift as a consequence of numerical errors in the multipliers method due to the extra constraints compared to the system's DOF. Also generalized coordinates can become impractical if we desire to build a modular system or a complex system like it is our case. The here proposed method improves both ODE and Featherstone's physics for designing complex aerial robots, since mathematics become rather simple to implement once they are deciphered, a low amount of parameters are required, large chains can be defined, there is no need to invert complex mass matrices at each time-step, it allows the implementation of coupled systems and can be very computationally efficient if the amount of integrations is contained.

We can find a rather limited amount of large morphing and flapping-wing robots in the literature. Colorado et al. (2012) model a bat-like UAV using Featherstone's articulated rigid multi-body formalism, so the equations of motion are based on Newton-Euler equations of motion (EoM), representing the body as two serial chains, but it doesn't account for dissipation effects in the joints, nor includes a modelling of the motors and batteries. [Caetano et al. (2015)] perform a comparison between a single rigid-body flapping-UAV model and a rigid multi-body flapping-UAVs model, determining that even though the first is rather easy and fast to model its accuracy is very low and the aerodynamics are completely coupled to the body kinematics, while the multi-body approach is more complex but accurate and the aerodynamic forces can be modelled independently. Also they emphasize that the method avoids expensive differentiations of the energy-based methods. Nevertheless, the method proposed in this thesis avoid those differentiations computing them mathematically, greatly reducing their impact. [Serrani et al. (2010)] builds an under-actuated averaged flapping-wing model with rigid multi-body Kane's EoM for robust control studies. [Sibilski et al. (2004)] represent a flapping-wing UAV in generalized coordinates with Gibbs-Appel EoM, also for control studies. [Sanchez-Laulhe et al. (2022)] use very simple non-dimensional Newton-Euler EoM for a single rigid body representation. [Dietl et al. (2011)] model an ornithoptic blimp with Euler-Bernoulli equations of motion for rigid bodies, but reduced to the longitudinal dynamics only. [Obradovic and Subbarao (2011)] work out a non-standard rigid-body EOMs representation of a gull-wing aircraft dynamics, treating the morphing aircraft as a single body, with the difference that the inertia

tensor and the CM become an explicit function of time. Starting off Obradovic’s work, [Guo et al. (2016)] presents a decoupled quasi-rigid body modelling approach tested again on a gull-wing aircraft. Also [Neal et al. (2004)] designed a fully morphing UAV, with morphing fixed-wing, showing the difficulty of dealing with CG displacement (and aerodynamic center) when the vehicle experiences large structural morphings. But accounting for the inertia tensor and the CM being a function of time with multibody dynamics is very expensive computationally. On the other hand, the study developed by [Berg and Rayner (1995)] suggests an enormous importance of the wings inertia in birds and bats for their motion. Our method accounts for those inertias and CM drifting automatically and efficiently, without any added computation, improving the just mentioned methods, while also enabling the easy addition of dissipative elements.

[Pfeiffer et al. (2010)] designed a simulation framework for ornithopters modeled as flexible multi-bodies combining the software MSC.ADAMS for modelling the dynamics of the robot and ANSYS for the definition of the flexible wings. Nevertheless, they implement a rather simple model of the robot dynamics. Flexible Multi-body dynamics would become impractical for more complex UAVs. Conversely, the framework developed along this thesis allows for a very agile definition of the body elements and very complete dynamics of all those elements.

Alternatively, [Samuel et al. (2014)] demonstrate that bond-graphs are an efficient approach for complex and coupled dynamics systems modelling a hummingbird-like UAV using the Newton-Euler formalism with body fixed coordinates, but considering the mass and inertia of the wings negligible. Later [Jahanbin et al. (2015)] and [Jahanbin and Karimian (2018)] presented a Newton-Euler bond-graph model of a flapping-wing UAV with two flexible wings modelled as Euler-Bernoulli beams. The model is very detailed, including mechanical and electrical subsystems. [Abbasi et al. (2021)] extend this model adding a gust-mitigating mechanical system to it also in bond-graph notation.

Bond-graphs are visually very meaningful, can include dissipative elements, energy-based controls can be applied on them and can be connected to block diagrams. Nevertheless, bond-graphs require a very large amount of elements compared to the port-Hamiltonian approach proposed in this thesis (see figure 1.5), making them very difficult to read and understand. Our Dirac structures modelling paradigm sets an easy and readable framework as it will be demonstrated later on.

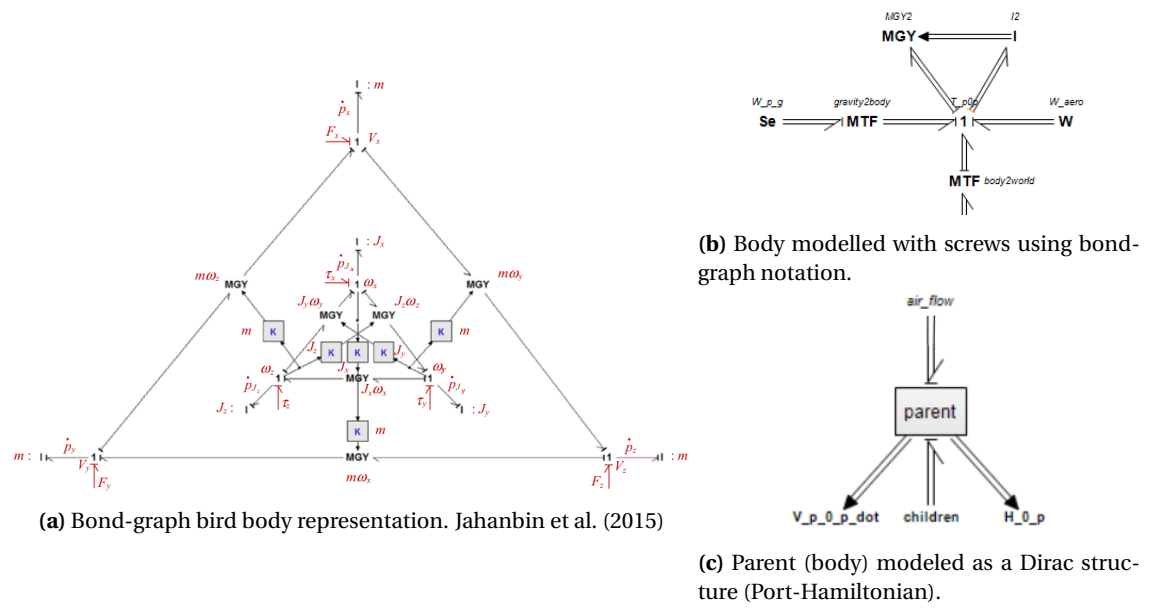


Figure 1.5: Comparison of Bond-graph modelling vs screws vs Port-Hamiltonian Dirac structures.

Recently, [Cardoso Ribeiro (2016)] developed port-Hamiltonian's theory based models of fluid systems and fluid-structure interactions, together with control algorithms for such systems. Similarly, [Rashad et al. (2021a)] (part I) and [Rashad et al. (2021b)] (part II) model the inviscid fluid flow with a port-Hamiltonian model in terms of Stokes-Dirac Structures. [Hong et al. (2021)] introduced a geometric formulation for generic multirotor UAVs. At last, [Abdelbadie (2021)] presented a port-Hamiltonian model of the Robird using Dirac structures, together with an aerodynamics model (strip theory approach).

1.3 Problem Statement & Research questions

The ultimate goal of this thesis is to **create a tool or environment to develop analytical models of case-study morphing-wing and flapping-wing robots**. Furthermore, the objective is to implement a new modelling of the bodies physics, substituting (and improving) the previously (commonly) used rigid-body, space-state or bond-graph models (among others) with geometric port-Hamiltonian models instead. It should set the first stone of an environment that would improve nowadays robotic simulators and allow for the future development of good flapping-wing surrogate models. It is part of a project to develop a generic framework for modelling, simulation and control of complex aerial robots, with focus on flapping-wing UAVs.

All that can be summed up under the following research questions:

- What's the state-of-the-art of the morphing-wing robots?
- What would be the energy-based atomic modules to support the geometric port-Hamiltonian modelling and simulation framework for morphing-wing aerial robots?
- How applicable is the framework for modelling and simulating literature example cases?

It's fundamental to understand that the focus of the thesis is on creating an environment or **framework for building models in a modular manner**. No air will be present in any model since that would add an immense complexity. This is, models will ignore aerodynamic effects. The thesis won't seek for numerical results, but ease of use and effectiveness of the method for building models imitating actual literature (state-of-the-art) morphing-wing robots, but in a geometric and energy-consistent way. In order to be able to do so, first a literature research and its corresponding documentation took place.

1.4 Proposed Solution

The presented modelling method is a continuation of the work developed by [Hong et al. (2021)], whose framework allow designing multirotor UAVs with a flexible number of rotors with any orientation in a rather modular way. Along this thesis their framework will be generalized for wing morphing-UAVs and implemented in 20-Sim.

Modularity and compactness is of great importance since we want to create a **generic framework for a wide range of complex aerial robots** that should be **easy to use and fast to model**. That's why, this thesis embrace a **parent-child philosophy**, where bodies are either a parent, a child or a parent-child element, as shown in figure 1.6. Models are built like a family tree, starting from a main body called *parent*, other bodies called *children* are attached to it like emerging branches that at the same time can lead to more branches underneath (*children of children*). All body elements are interconnected by means of *joints* that establish the degrees of freedom (if any) between each other.

In most modelling paradigms it is quite difficult to automate the modelling of robots, but the port-Hamiltonian method simplifies the definition of the frames where forces and moments are applied, since each body contains its own constraints and dynamics and the frame transfor-

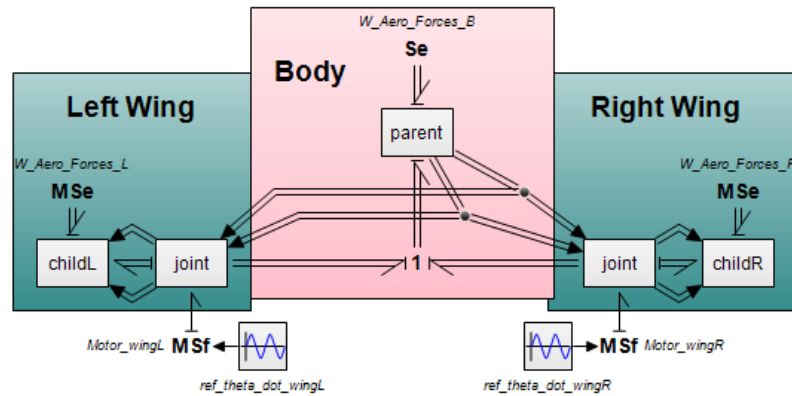


Figure 1.6: Parent-joint-Child basic model example. The model represents a main body with two wings.

mations are computed automatically. What's more, this method takes inertias into consideration and manages the changes in the CM by itself.

The fact that subsystems can be designed independently and then joined together very straightforwardly into a whole makes this method very advantageous. Moreover, the port-based structure makes models fully interconnectable with any other bond-graph element. It results in a modular, structured, and re-usable software framework where bodies, actuators and external elements are very simple to interconnect.

Additionally, port-Hamiltonian systems are "energy aware", this is, the exchange of energy is always 1:1 between bodies and environment, being faithful to physics, which means that models can be a lot more realistic and the designed control system can be more precise and robust afterwards.

1.5 Outline

An introduction was already displayed along this section 1, where it is exposed the problem and the proposed solution after analysing the related work. Next is presented the order of the following thesis chapters.

- On chapter 2 an analysis of the state of the art of the morphing-wing robots and more specifically of the flapping-wing robots will be performed.
- Chapter 3 will deal with the theoretical background. The mathematics supporting the modelling method will be expanded.
- Along chapter 4 the definition of the generic modelling and simulation framework is developed. The design of a parent, child and joint subsystems is closely analysed. These subsystems will be the base of all the models.
- Several experiments imitating literature models using the port-Hamiltonian parent-child modelling approach and their respective results are presented on chapter ??.
- Finally a discussion of the new modelling method will be held on chapter ??, followed by a conclusion on chapter 6.

2 State of the Art

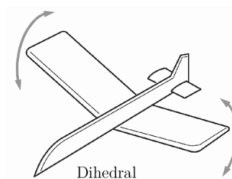
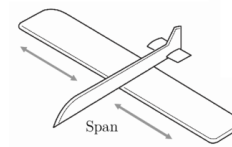
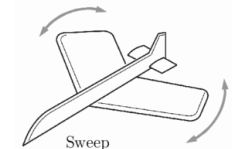
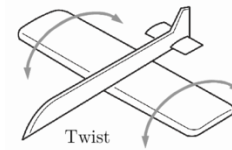
2.1 Morphing Wings. State of the art.

Morphing wings is a topic in aerospace since the very beginning of aviation, with the invention of the aeroplane in 1903 by the Wright brothers. Flaps and rudders are moving surfaces that are present since then and could be, to some extent, considered as morphing elements.

Wing morphing is a quite ambiguous term as it was already introduced in section 1.1.1. One could consider the movement of an aeroplane's flaps, small changes in its wing's camber shape, wing tip modifications or deformations in the leading and trailing edges as morphing, as many researchers do [Vos and Abdalla (2010)], [Smith et al. (2014)], [Pankonien and Inman (2013)], [Vasista et al. (2017)], [Rodrigue et al. (2016)], [Meguid et al. (2017)], [Martinez et al. (2017)], [Previtali et al. (2014)]. But for most of those cases, the moving masses are negligible compared to the mass of the whole vehicle, thus they don't really mean a substantial change in the structural characteristics of the plane nor cause any relevant inertial effect. Those type of morphings will be considered "small morphings", while the focus will be put on what can be referred as "dynamic morphings".

A morph will be considered "dynamic" when there is a considerable displacement of the CG or large inertia momentum or forces are generated. Furthermore, it is possible to divide the morphings, as introduced previously in section 1.1.1, in:

- **TWIST MORPHING:** It is a change in the angle of attack of the wings. Rotation around the wingspan axis. Such change of twist can be exerted from the root of the wing so the whole wing twist together or from the tip if the wing is flexible, with the base of the wing remaining in the original position.
- **SWEEP MORPHING:** Modifying the sweep is being used in military planes for a long time, with the F111 as the first extensively used bombardier that could sweep its wings while flying. Open wings are more stable, less power consuming, higher lift-drag ratio, so can lift-off moving slower, while back-swept wings allow for supersonic speeds and more maneuverability. Sweeping wings bring best of both worlds.
- **SPAN MORPHING:** Can be described as a change in the distance between wingtips. It can either be due to flexibility in the wing or due to some extraction and retraction of the wing. The span also changes together with other types of morphing like sweeping, but those won't be accounted as span morphing.
- **DIHEDRAL MORPHING:** The dihedral angle is the angle that forms the wing with the body. Dihedral morphing is the main morphing present in all flapping UAVs.
- **GULL WING MORPHING:** Can be considered a type of dihedral morphing, only with two bends per wing.



Aerial vehicles are normally designed to be the most efficient for one complete task, compromising optimal configuration for the different stages of the flight. For instance, a commercial

flight with passengers is required to be very steady and stable, compromising maneuverability and speed, while a fighter has to be fast and maneuverable, no matter (that much) the fuel consumption. But being able to fly in several regimes would be of great use, since some configurations are more convenient for the lift-off for example, while very inefficient during steady flight and vice-versa. The bigger the span the greater lift/drag ratio, the slower the vehicle, but the more energy efficient. The span can be modified extending-contracting the wings, but also sweeping them. In the same manner, maneuverability depends on the location of the center of gravity and the aerodynamic center, which can be modified sweeping the wings. Camber morphing and wing twisting also involve a change in the lift/drag ratio.

[Barbarino et al. (2011)] summarizes the state of the art of morphing-wing aircrafts and UAVs since 1903 to 2010, put together in image 2.1. All of them share one common thing, their morphing frequency is very low, this is, they do not imply any important inertial effect, but mostly large CG displacements and mass distribution changes. Another observation that can be made here is that the most common morphing used is the sweep, as it has an enormous influence in the vehicle's maneuverability, its speed and its lift/drag ratio. Similarly [Da Ronch et al. (2018)] analyse the progress of morphing-wings from 1903 to 2017 and conclude that the morphings with a bigger impact in the dynamics are the sweeping, the span and the folding/flapping, followed by the twisting.
















1903	1931	1931	1932	1937	1947	1951
						
Wright Flyer <i>Twist</i>	Pterodactyl IV <i>Sweep</i>	MAK-10 <i>Span</i>	IS-1 <i>Bi-to monoplane</i>	LIG-7 <i>Chord</i>	MAK-123 <i>Span</i>	X 5 <i>Sweep</i>
1952	1964	1964	1966	1967	1967	1969
						
XF10F <i>Sweep</i>	F 111 <i>Sweep</i>	XB 70 <i>Span bending</i>	Su 17 IG <i>Sweep</i>	MIG 23 <i>Sweep</i>	SU 24 <i>Sweep</i>	Tu 22 M <i>Sweep</i>
1970	1972	1974	1974	1979	1981	1985
						
F 14 <i>Sweep</i>	FS 29 <i>Span</i>	B 1 <i>Sweep</i>	Tornado <i>Sweep</i>	AD 1 <i>Obliquing</i>	Tu 160 <i>Sweep</i>	AFTI/F 111 <i>M.A.W.</i>
1993	1994	2001	2002	2003	2004	2005
						
FLYRT <i>Span</i>	MOTHRA <i>Camber</i>	AAL <i>Pitch</i>	F/A 18 <i>A.A.W.</i>	Virginia Tech <i>Span</i>	Univ. of Florida <i>Twist</i>	Univ. of Florida <i>Gull</i>
2006	2006	2007	2007	2007	2008	2010
						
MFX 1 <i>Sweep & Span</i>	Univ. of Florida <i>Sweep</i>	Virginia Tech <i>Camber</i>	Univ. of Florida <i>Folding</i>	MFX 2 <i>Sweep & span</i>	Delft Univ. <i>Sweep</i>	Virignia tech <i>Camber</i>

Figure 2.1: Morphing flying vehicles until 2010. [Barbarino et al. (2011)].

Apart from the morphing vehicles presented in figure 2.1, there are few others worth mentioning, included in table 2.1. [Ajaj and Jankee (2018)] designed a fully adaptive UAV, capable of modifying its sweep, span (wing length) and even twist a part of the wings simultaneously. [Obradovic and Subbarao (2011)] carried out a study of the modelling and simulation of all possible gull-wing configurations, for steady flight and turning without needing ailerons. Finally, the UAV presented by [Ajaj and Jankee (2018)] is capable of morphing both symmetrically and asymmetrically to also exploit rolling moments to turn without ailerons.

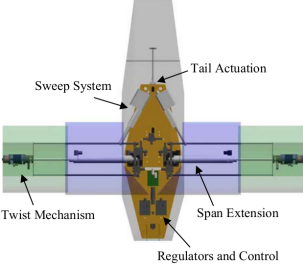
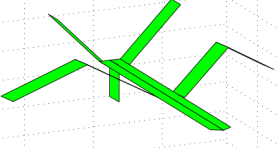

	UAV	Morphing Type	Author, Year	Purpose
	Fully Adaptive UAV	Sweep, Twist and Span Morphing	Neal et al. (2004)	Multi-mission
	Bird UAV	Gull Morphing-Wing	Obradovic and Subbarao (2011)	R&D
	Transformer aircraft	Prismatic-Wing	Ajaj and Jankee (2018)	Multi-mission

Table 2.1: Additional Morphing-wing UAVs in the state of the art.

2.2 Bio-inspired Morphing-Wing UAVs. State of the art.

Nevertheless, this thesis focuses on the bio-mimicking morphing-wing and more importantly flapping-wing UAVs, this is, bats and birds sizes and high morphing frequency. Even though the overall flying dynamics are very different and much more difficult in birds and bats than in propelled airplanes and UAVs, some of their flying principles apply. Birds sweep back their wings when attacking, this is, when they require speed and maneuverability and they extend their wings (large span) while gliding, using very low energy. Additionally, they twist the wings for perching or taking-off. Altogether, the overall flapping motion include several of these morphings simultaneously, which makes the flapping flight so complex. This section takes a closer look into these propelled but highly morphing bio-inspired UAVs.

Table 2.2 show some literature bio-inspired UAVs actuated by propellers. The most important examples of bio-inspired UAVs up to 2010 are the vehicles designed by the University of Florida, that started in 2004 experimenting with the twisting of the wings and later with the gull wing and the sweeping wing. The UAVs presented by [Abdulrahim and Lind (2004)] and [Grant et al. (2006)] (University of Florida) are an initial approach towards flapping-wing robots, both inspired by a seagull. The first consist on a gull-wing (dihedral) morphing UAV, while the second is comprised by multiple joints in each wing independently actuated that allow the UAV to sweep its wings asymmetrically, adopting many different configurations. [Delft and Wageningen (2007)] designed an asynchronous sweeping-wing UAV imitating a swift bird, that

can fly for long periods thanks to the efficiency the morphing capabilities provide. Another very unique bio-inspired (Sulidae birds family) UAV is the AquaMAV developed by [Siddall et al. (2017)], consisting on an UAV that can dive into the water to gather data and come back out flying again.


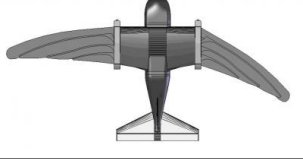
	UAV	Morphing Type	Author, Year	Purpose
	Gull-Wing UAV	Gull-Wing Morphing	Abdulrahim and Lind (2004)	R&D
	Multiple-Joint Wing Sweep	Sweeping-Wing	Grant et al. (2006)	R&D
	RoboSwift	Feathered Sweeping-Wing	Delft and Wageningen (2007)	Surveillance
	AquaMAV	Air-Water Sweeping-Wing	Siddall et al. (2017)	Gather Data
	Pigeon UAV	Feathered Morphing-Wing	Matloff et al. (2020)	R&D
	LisHawk	Feathered Morphing-Wing and Tail	Ajanic et al. (2020)	R&D

Table 2.2: Bio-inspired Morphing-Wing UAVs in the state of the art.

More recently [Matloff et al. (2020)] and [Ajanic et al. (2020)] presented two feathered UAVs inspired in a pigeon and a northern goshawk respectively. Both equipped with real feathers that overlap when the wings sweep. Whats more, [Matloff et al. (2020)] demonstrate how their mechanism make the UAV more robust against turbulences. On the other hand, [Ajanic et al. (2020)] go further and include a retractable feathered tail, actuated by an universal joint that enable the tail to be moved in any direction.

But, as already mentioned, all these robots are blade propelled, which make them easier to model and design. The next section 2.3 dives into the literature of the flapping-wing robots in particular. A field that has experimented a considerable growth in the last decade.

2.3 Flapping-Wing UAVs (Ornithopters). State of the art.

Ornithopter is a term that embraces any flapping-wing vehicle, from insects to large birds or even aeroplanes (only seen in movies). This section though, comprises a research of the state of the art of bat and bird size ornithopters. Insects are left aside, since their wings are so light and move so fast that their inertial effects can be neglected besides the displacements of the CG, leading to simplified models.

The most important flapping-wing robots present in the literature are summarized in tables 2.3 and 2.4. Starting from the Nano Hummingbird designed by [Karásek and Preumont (2011)] for surveillance purposes. Even though it is a bird in real life, in robotics it is closer to insects, since its motion is more similar to those, as the wings have very high flapping frequency. The first actual flapping-wing robot was Festo's Smartbird [Send et al. (2012)], a ground (air) breaking robot at that time. Smartbird achieved a high degree of realism in its motion, being able to orientate itself moving its tail accordingly.

Another interesting project is Robo Raven [Gerdes et al. (2014)]. It is a solar powered flapping UAV with asynchronous wings, which allows it to fly longer and execute complex maneuvers. Additionally, it can be equipped with a camera for data gathering or surveillance.

Continuing with the robotic birds, Robird is a falcon mimicking flapping-wing UAV developed by [Folkertsma et al. (2017)]. Its motion is very realistic, together with its appearance, which make it be perceived as a prey bird by other birds. This fact means that birds will fly away from it, allowing the robot to be used for bird pests control, for instance in airports, where falcons are normally used to scare the birds away from the runway. This robot would ideally replace the falconer that is normally required in every airport.

In a more simplistic way, Dove was presented by [Yang et al. (2018)] as a flapping-wing UAV equipped with a surveillance camera in its chest.

On the other hand, E-Flap [Zufferey et al. (2021)] is a high-payload flapping-wing robot. It is designed with the idea of carrying objects. For that, the research group has developed the robot with a falcon-like prehensile leg that can be used for either perching on tree branches or grabbing the objects. So far they have achieved to perch in a very controlled environment.

[Colorado et al. (2012)] developed a flapping bat robot that would exploit the inertia moments of the wings for lifting and maneuvering. They perform several experiments for studying the inertial forces on a test bench, gathering very relevant data. The robot has three actuated joints in each wing, one for the shoulder, one for the elbow and the last for the wrist. Also the wing's membrane is very flexible, similar to that of bats, so wing's configurations are almost limitless. [Ramezani et al. (2016)] go one step further with their Bat Bot B2, adding actuators in the legs for further control and maneuverability. The flapping is synchronous, but the contraction/expansion of the wings and the up/down movement of the legs are all asynchronous. The authors reported that the bat is able to perform autonomous flight.

Half-way between a bird and a bat is Robo-Falcon, presented by [Chen et al. (2021)]. It consists in a robotic bird of prey robot with bat-like wing articulation and membrane. It's flapping mechanism is designed so the wings are fully extended during the downstroke and tucked during the upstroke. Whats more, it is possible to decouple the wings to actuate them asynchronously for achieving faster rolling (turns). Still, the robot is built with flaps on its tail for maneuvering.


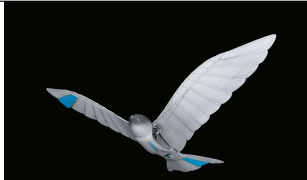
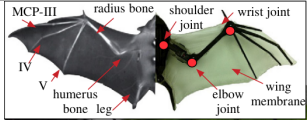

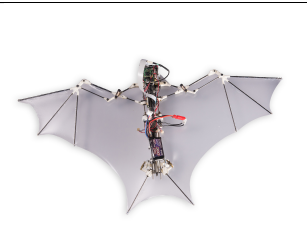



	UAV	Morphing Type	Author, Year	Purpose
	Nano Hammingbird	Flapping-Wing	Karásek and Preumont (2011)	Surveillance
	SmartBird	Flapping-Wing	Send et al. (2012)	R&D
	Bat	Flapping-Wing (Bat)	Colorado et al. (2012)	R&D
	Robo Raven	Flapping-Wing	Gerdes et al. (2014)	Data gathering / R&D
	Bat Bot B2	Flapping-Wing (Bat)	Ramezani et al. (2016)	R&D
	Robird	Flapping-Wing	Folkertsma et al. (2017)	Bird pests control / R&D
	Dove	Flapping-Wing	Yang et al. (2018)	Surveillance
	E-Flap	Flapping-Wing	Zufferey et al. (2021)	R&D

Table 2.3: Flapping wing UAVs state of the art. (I)



	UAV	Morphing Type	Author, Year	Purpose
	Robo-Falcon	Flapping-Wing (Bat-like)	Chen et al. (2021)	R&D
	MetaBird	Flapping-Wing	Edwin Van Ruymbeke (-)	Entertainment

Table 2.4: Flapping wing UAVs state of the art. (II)

Differently from the rest of robotic birds that tend to be used for R&D purposes, MetaBird is a small ornithopters oriented for entertainment that can be controlled from a smartphone. Curiously, the french police is working on incorporating it to their forces as spy drones [Bisht (2022)].

2.4 Conclusion

Along this section a research is presented on the state of the art of morphing-wing UAVs with an emphasis on flapping-wing robots. It was first briefly review the wing morphing aerial vehicles, highlighting some interesting cases like [Neal et al. (2004)] and [Obradovic and Subbarao (2011)]. Next it was commented the state of the art of bird-mimicking non-flapping (blade propelled) morphing-wing UAVs, among which stands out [Grant et al. (2006)] and [Ajanic et al. (2020)]. Before finally diving into the, not exactly vast amount of flapping-wing robots. From which we can distinguish [Send et al. (2012)], [Colorado et al. (2012)], [Ramezani et al. (2016)], [Folkertsma et al. (2017)] and [Zufferey et al. (2021)].

No doubt designing flapping-wing UAVs is still very difficult, but it is also clear that the interest in these bio-inspired robots is growing and that is reflected in the last years developments. There are almost no bat-like nor bird-like ornithopters present in the literature till 2011, but only blade propelled aerial robots. Nevertheless, use-cases for these UAVs are increasing and the designs will become more and more complex besides their control laws when looking for efficiency, payload and mimicry.

It is in fact the increasing complexity and the increasing use-cases of these flapping-wing robots what makes this study necessary and the consequent development of a framework for modelling and simulating generic morphing-wing UAVs so relevant.

3 Mathematical Background

Along this chapter the mathematics that support the *parent-child* modelling approach are expanded, starting with an explanation of the key Lie-group theory, Screw theory and port-Hamiltonian theory.

3.1 Lie Group Theory

The here presented modelling and simulation framework is supported on its very base by mainly elements of the Lie-group theory as these are the twists, wrenches, the Adjoint and the adjoint. Next it is exposed an introduction to these mathematical concepts, but starting by explaining what a Lie Group is.

A Lie Group is a smooth, differentiable, manifold group. To every Lie-group we can associate a Lie Algebra that is the tangent space of the Lie Group at the identity element and which completely captures the local structure of the group. The Lie Algebra is a common space thanks to the Lie Group structure, that can be for example the common space of rotational matrices or the common space of homogeneous matrices.

A Euclidean Space, ϵ , is a space of free vectors plus an inner product which can tell us about orthogonality of vectors and the absolute length of those. Such space can be used for describing motions of objects. It allows for the use of coordinate systems and the mapping between different coordinate systems. As an example, a rotational matrix R would be a map between two coordinate systems that share the same origin, but are rotated one wrt the other.

A square (rotation) matrix $R \in \mathbb{R}^{3 \times 3}$ such that $R^{-1} = R^T$ is called orthonormal. The group of orthonormal matrices with determinant 1 is called Special Orthonormal group, $SO(3)$, of \mathbb{R}^3 .

If $R(t) \in SO(3)$ defines the configuration of a body along time, and it is a continuous and differentiable function of time then its derivatives $\dot{R}R^T$ and $R^T\dot{R}$ are skew-symmetric and belonging to $so(3)$, that fulfills the definition a Lie Algebra.

A skew-symmetric matrix is of the form:

$$\text{IF } x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \rightarrow \text{In skew-symmetric form: } \tilde{x} = \begin{pmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{pmatrix}.$$

So, going back to the Lie Algebra $so(3)$, the angular velocity $\tilde{\omega}$ belong to the Lie Algebra $so(3)$ ($\tilde{\omega}_1, \tilde{\omega}_2 \in so(3)$) since $\tilde{\omega}_1 = \dot{R}R^T$ and $\tilde{\omega}_2 = R^T\dot{R}$. This implies that the angular velocities can be expressed independently from the configuration of the body (R).

Same as with rotations, we can expand this to general motions, that use instead of rotation matrices homogeneous matrices,

$$H = \begin{pmatrix} R & o \\ 0 & 1 \end{pmatrix},$$

where R is the rotation matrix and o the translation vector. And, as with R , $H(t) \in SE(3)$ is a continuous and differentiable function of time and its derivatives $\dot{H}H^{-1}$ and $H^{-1}\dot{H}$ belong to $se(3)$ (Lie Algebra). So in this **common space se(3)** appears a new element, the **twist T**, described as

$$T = \begin{pmatrix} \omega \\ v \end{pmatrix},$$

or in tilde form

$$\tilde{T} = \begin{pmatrix} \tilde{\omega} & v \\ 0 & 0 \end{pmatrix},$$

that is equivalent to ω , but for general motions, this is, it is the generalization of rigid body's velocities.

Additionally, a one degree of freedom kinematic-pair or joint constraints the relative motion of two objects a, b with a **unique and constant "unit twist"**, $S_a^{j,b}$, as long as Ψ_j (reference frame j) is fixed to either a or b ,

$$T_a^{j,b} = \theta S_a^{j,b}, \text{ with } \theta \in \mathbb{R}.$$

The twist and unit twist nomenclature just introduced represents the motion of body a wrt body b as seen from frame j , so that θ controls the degree of freedom between the two bodies a, b . Unit twists will be used constantly along the thesis.

Lie Group is ultimately so important because it **permits us to talk about motion (\tilde{T}) without knowing the configuration (H) of the object**. We can get rid of any dependency from configuration, which is essential to talk about interconnection through power ports of bodies with different configurations.

3.1.1 Dual Space: Twists & Wrenches

For any finite dimensional vector space we can define the space of linear operators from that space to power (dimensionless). This linear operators are referred as co-vectors and belong to the dual space of the vector. To visualize it better, for the case of the vector space of angular velocities $\omega \in so(3)$ there is a dual space (co-vector space / linear operator), torque $\tau \in so^*(3)$, such that $\tau * \omega = P \in \mathbb{R}$, where P represents power.



Figure 3.1: Dual Space. [Stramigioli (2021)]

The very same can be extrapolated to the twist, that is a vector. There is a co-vector that is the linear operator from twist to power,

$$WT = P,$$

where the co-vector W is composed by a torque τ and a linear force f , and looks like

$$W = (\tau \quad f),$$

that in tilde form is

$$\tilde{W} = \begin{pmatrix} \tilde{f} & \tau^T \\ 0 & 0 \end{pmatrix}.$$

Such co-vectors, belonging to the dual space $se^*(3)$, are called **wrenches**, and can be seen as the generalization of forces for rigid bodies.

This duality characteristic is the base of the power-ports theory and twists and wrenches in particular are the basic elements of the Screw Theory, that is at the same time the base upon which the parent-child structures are supported.

3.1.2 Adjoint & adjoint

The Adjoint (Ad) is a matrix for twists and wrenches to change its coordinates frame system. Changing from the twist of body k wrt body l expressed in frame ϕ_i , $T_k^{i,l}$, to the same twist but expressed in coordinate frame ϕ_j , $T_k^{j,l}$, would look like:

$$T_k^{j,l} = Ad_{H_i^j} T_k^{i,l}, \text{ where } \rightarrow Ad_{H_i^j} = \begin{pmatrix} R_i^j & 0 \\ \tilde{o}_i^j R_i^j & R_i^j \end{pmatrix}$$

This is, the coordinate frame transformation of the twist (and wrench) from coordinate frame ϕ_i to coordinate frame ϕ_j is done using the Adjoint of the homogeneous matrix that relates those two frames H_i^j . In the case of wrenches the transformation occurs the other way around, as:

$$(W^i)^T = Ad_{H_i^j}^T (W^j)^T$$

What in port-theory is interpreted as a change of coordinates (MTF) can be "simplified" introducing the Adjoint directly inside the dynamic equations of the body, enabling a simplification of the modelling. See this equivalence in figure 3.2. It is a key element of the modelling and simulation approach.

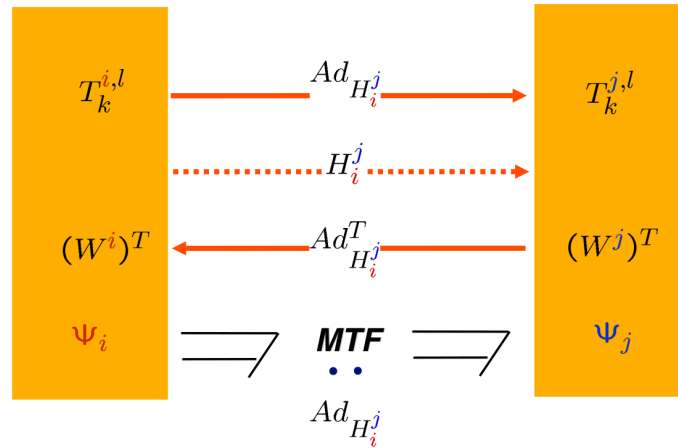


Figure 3.2: Change of coordinates of Screws. Adjoint. [Stramigioli (2021)]

Additionally, the adjoint results from the time derivative of the Adjoint. This is an element needed for defining the dynamics of bodies, as it will be shown later in section 3.1.5, and it is obtained as:

$$(\dot{Ad}_{H_i^j}) = Ad_{H_i^j} ad_{T_i^{i,j}}, \text{ where } \rightarrow ad_{T_i^{i,j}} = \begin{pmatrix} \tilde{\omega}_i^{i,j} & 0 \\ \tilde{v}_i^{i,j} & \tilde{\omega}_i^{i,j} \end{pmatrix}$$

3.1.3 Screw Theory

This thesis has one of its focuses on the ease of use and the benefits of the so called, screw theory, that is, in few words, an algebraic calculation of pairs of vectors and co-vectors (twists and wrenches) that define a body's motion. The screw theory allows a **geometric definition of the rigid body dynamics independent of the configuration** of such body and allows the interpretation of the interconnections between bodies by means of power ports (port-Hamiltonian).

These characteristics make the screw theory a rather powerful tool for modelling any type of robotic system, and in particular, UAVs and flapping-wing robots.

Geometric: The essence of geometric dynamics is the fact that the motion of the whole system can be identified with the motion of a certain virtual point along a geodesic (generalization of the notion of a "straight line") in a Riemannian manifold (real, smooth manifold equipped with a positive-definite inner product on the tangent space at each point). The geometric quality of the framework is essential for compactly describing the robot's model in a coordinate-free manner using Lie Group theory.

The screw theory is built on top of the Lie Group theory, thus it embraces all the attributes of that theory.

With all that said, the screw theory can be finally explained via two dual theorems. The first, Mozzi's Theorem, apply for twists:

Mozzi's Theorem (1763), Chasles Theorem (1830): Any rigid body motion can be expressed as a rotation around an axis and a translation along the same axis.

$$\begin{pmatrix} \omega \\ v \end{pmatrix} = \begin{pmatrix} \omega \\ r \wedge \omega \end{pmatrix} + \lambda \begin{pmatrix} 0 \\ \omega \end{pmatrix} \quad (3.1)$$

While the second, Poincot's Theorem, apply for wrenches:

Poincot's Theorem: Any system of forces can be expressed as a pure linear force along a line plus a pure moment around it.

$$\begin{pmatrix} \tau \\ F \end{pmatrix} = \begin{pmatrix} r \wedge F \\ F \end{pmatrix} + \lambda \begin{pmatrix} F \\ 0 \end{pmatrix} \quad (3.2)$$

3.1.4 Twist's Exponential

The relative configuration of two bodies can be obtained as a function of time, t , as:

$$H_i^j(t) = e^{\hat{S}_i^{j,j} t} \cdot H_i^j(0) \quad (3.3)$$

Or, equivalently, as a function of the coordinate q , as:

$$H_i^j(q) = e^{\hat{S}_i^{j,j} q} \cdot H_i^j(0) \quad (3.4)$$

Where the exponential of the unit twist multiplied by q would represent the actual state of the body, while $H_i^j(0)$ is the initial configuration.

To compute the exponential of the unit twist the Rodriguez's formula is used, defined as:

For any rotation $\hat{\omega} \in so(3)$ with $\|\omega\| = 1$ ($\hat{\omega}$):

$$e^{\hat{\omega}\theta} = I + \hat{\omega} \sin \theta + \hat{\omega}^2 (1 - \cos \theta) \quad (3.5)$$

Same can be expanded for twists, resulting in:

$$e \begin{pmatrix} \hat{\omega} & \hat{v} \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} e^{\hat{\omega}} & (I - e^{\hat{\omega}})r + \lambda \hat{\omega} \\ 0 & 1 \end{pmatrix} \quad (3.6)$$

In this last equation appears the previously defined Rodriguez's formula for rotations ($e^{\hat{\omega}}$), r , that is the distance of the joint to the parent and $\lambda\hat{\omega}$ that is a term only present in screw joints, this is, not in pure rotations or translations. In our particular case the model is designed for pure rotations mainly, but also for translations, resulting in the following exponentials:

1. Screw:

$$e \begin{pmatrix} \hat{\omega} & \hat{v} \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} e^{\hat{\omega}} & (I - e^{\hat{\omega}})r + \lambda\hat{\omega} \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} e^{\hat{\omega}} & (I - e^{\hat{\omega}})\hat{\omega}\hat{v} + \hat{\omega}^T\hat{v}\hat{\omega} \\ 0 & 1 \end{pmatrix} \quad (3.7)$$

Where $r = \frac{\omega \wedge v}{\|\omega\|^2}$ and since $\hat{\omega} \rightarrow \|\omega\| = 1$ then $r = \hat{\omega} \wedge \hat{v}$ or $r = \hat{\omega}\hat{v}$. Similarly, $\lambda = \frac{\omega^T v}{\|\omega\|^2}$, so $\lambda = \hat{\omega}^T \hat{v}$.

2. Rotational:

$$e \begin{pmatrix} \hat{\omega} & \hat{v} \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} e^{\hat{\omega}} & (I - e^{\hat{\omega}})r \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} e^{\hat{\omega}} & (I - e^{\hat{\omega}})\hat{\omega}\hat{v} \\ 0 & 1 \end{pmatrix} \quad (3.8)$$

3. Translational:

$$e \begin{pmatrix} I & \hat{v} \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} I & \hat{v} \\ 0 & 1 \end{pmatrix} \quad (3.9)$$

3.1.5 Rigid Body Dynamics

If we take the Euler equations for bodies, then the momenta P of a rigid body can be expressed as

$$(P^i)^T = G^i T_i^{i,0} \quad (3.10)$$

and Newtons law generalization for rigid bodies can be similarly written as

$$\dot{P}^{0,i} = W^{0,i} \quad (3.11)$$

that in body coordinates result in

$$(\dot{P}^i)^T = ad_{T_i^{i,0}}^T (P^i)^T + (W^i)^T \quad (3.12)$$

And joining equations (3.10) into (3.12) the dynamic equation of the body is obtained as

$$G^i \dot{T}_i^{i,0} = ad_{T_i^{i,0}}^T (G^i T_i^{i,0}) + (W^i)^T \quad (3.13)$$

Where G^i is the inertia tensor that defines the shape of the body. Note that this last equation is **independent from configuration**.

3.2 Port-Hamiltonian Theory

The key concept in the formulation of port-based models of physical systems as port-Hamiltonian systems is the geometric notion of a Dirac structure [Duindam et al. (2009)], so we can directly dive into them.

3.2.1 Dirac Structures

Dirac structures are a great tool in port-Hamiltonian systems, as they make possible the creation of systems by means of physically interconnected subsystems [Yoshimura and Marsden (2006)]. This is thanks to its most important feature, which is that any system conformed by several subsystems, in Dirac structure form, give place to another Dirac structure [Van der Schaft and Cervera (2002)]. This is, Dirac structures joined together generate new Dirac Structures, enabling at the same time the possibility to create modular systems. The Dirac structures themselves establish the dynamical constraints of the system and subsystems and they are power conservative.

A Dirac structure is defined as a subspace D of the vector's space (V) of flows f and co-vector's space (V^*) of efforts e , $D \subset V \times V^*$ such that $D = D^\perp$, so the Dirac structure satisfies:

- $\langle e|f \rangle = 0$, for all $(f, e) \in D$, which means that for every pair (f, e) in the Dirac structure the power exf is equal to zero.
- $\dim D = \dim V$, the subspace has maximal dimension with respect to this property.

As presented by Van der Schaft and Cervera (2002), there are different matrix representations for the Dirac structures, but for our particular case we are interested only in the Hybrid input output representation: Let D be given by square matrices E (Efforts) and F (Flows) such

- $EF^T + FE^T = 0$,
- $\text{rank}[F;E] = n$.

Suppose $\text{rank } F = m (\leq n)$. Select m independent columns of F , and group them into a matrix F_1 . Write $F = [F_1; F_2]$, $E = [E_1; E_2]$, $f = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$, $e = \begin{bmatrix} e_1 \\ e_2 \end{bmatrix}$. Then $[F_1; E_2]$ is invertible and

$$D = \left\{ \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}, \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \mid \begin{bmatrix} f_1 \\ e_2 \end{bmatrix} = J \begin{bmatrix} e_1 \\ f_2 \end{bmatrix} \right\} \quad (3.14)$$

with $J := -[F_1; E_2]^{-1} [F_2; E_1]$ skew-symmetric.

All future work in this project is based on these structures, since all the atomic subsystems will be represented in such manner.

3.2.2 Power-ports and Signal-ports

Differentiating between power ports and signal ports can be crucial for the correct understanding of the modelling and simulation methodology, since they have very different physical meanings.

Power ports consist of two dual components called flow, f , and effort, e , that multiplied result in power. They are used to create physically meaningful and physically ideal connections between elements. For instance, flow can be a twist and its dual effort would be a wrench. When multiplied together results in power, that is a common language between systems of any physical domain. It could be said that the power ports transmit energy.

On the other hand, signal ports don't have a physical meaning, but they are just a simple way to transmit any data.

3.2.3 Power and Energy Balance

The total sum of both power and energy in the Dirac structures should be equal to 0.

On the one hand, the power of the rigid body can be defined as:

$$Power = (T_i^{i,0})^T W^i = (T_i^{i,0})^T \dot{P}^i = (T_i^{i,0})^T G^i \dot{T}_i^{i,0} \quad (3.15)$$

On the other hand, the kinetic co-energy of the rigid body can be expressed as:

$$E^* = \frac{1}{2} (T_i^{i,0})^T G^i T_i^{i,0} \quad (3.16)$$

If the body is a Dirac structure, thus, energy conservative and the power balance is fulfilled, then the next equality should be fulfilled:

$$E_{Balance} = E^* - \int Power = \frac{1}{2} (T_i^{i,0})^T G^i T_i^{i,0} - \int (T_i^{i,0})^T G^i \dot{T}_i^{i,0} = 0 \quad (3.17)$$

3.3 Conclusion

Along this chapter the main mathematical tools required for understanding the modelling and simulation framework are explained.

The chapter starts with the definition of the Lie group theory, that includes the explanation of the duality twists and wrenches and the coordinate frame transformations with the Adjoint and adjoint. Next the Screw theory was introduced, which allows for a geometric definition of the bodies. Followed by a solution for calculating the twist exponential and the derivation of the body dynamic equation in an independent from configuration manner. Finally a definition of the Dirac structures representation is explained . Additionally, it is exposed the difference between a power port and a signal port.

All this mathematical background become necessary tools for understanding the definition of the geometric port-Hamiltonian modelling and simulation framework introduced next in chapter 4, as it is completely supported by the concepts introduced along this chapter.

For more information about the screw theory and port-Hamiltonian modelling the reader is encouraged to read Duindam et al. (2009) and Merzouki et al. (2013b).

4 The Modelling and Simulation Framework

4.1 Framework Overview. The Parent-Child Paradigm.

The parent-child modelling method consist on defining each body as either a parent or a child, that are represented as Dirac structures with all power ports pointing inwards and interconnected via joints. The joints, which are also Dirac structures, define the relative movement between bodies and acts as a translator or interpreter of the twists and wrenches expressed in one body's coordinate frame into the other body's coordinate frame. The framework is port-Hamiltonian based, so all interconnections between bodies take place by means of either power ports or signal ports, which is why all elements of the framework will be illustrated as blocks with ports. The use of power power ports, that are physically meaningful instead of simple signals is what makes the method energy consistent. The paradigm was previously developed by [Hong et al. (2021)] for modelling multirotors.

The intention behind this methodology for modelling flapping-wing UAV's is to create a base for simulating many different systems, thus the framework should be as modular as possible. In this line of thinking, *parent*, *child* and *joint* are designed in the most generic way possible. Seeking requiring the least interconnections and parameters possible for building any model.

The following figure 4.8 show the parent p and child c interconnected through the joint (J_c^p). From now on indices 'p' and 'c' will refer to parent and child respectively.

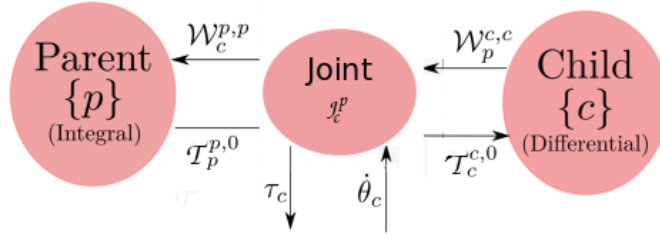


Figure 4.1: Parent-Joint-Child diagram. Modified from [Hong et al. (2021)].

Next it is explained the obtaining of the equations governing parent, child and joint as Dirac structures.

4.2 Parent Subsystem

The *parent* represents the main body of any model and it is the reference body to which all other elements, like the wings in the case of a flapping-UAV are attached. The *parent* subsystem contains the main body dynamics equation in integral causality, that can be directly extracted from eq.(3.13) and expressed as

$$G_p^p \dot{T}_p^{p,0} = [ad_{T_p^{p,0}}]^T G_p^p T_p^{p,0} + W_{ext}^{p,p} + \sum_{\alpha=1}^m W_{\alpha}^{p,p} \quad (4.1)$$

Here W_{α} represent the (m) children, whose wrenches are summed up. $W_{ext}^{p,p}$ is the external wrench, representing the gravity wrench, $W_{grv}^{p,p}$, plus the aerodynamics wrench, $W_{aer}^{p,p}$ ($W_{ext}^{p,p} = W_{grv}^{p,p} + W_{aer}^{p,p}$). About the second we are not interested for this studio, so won't be expanded, but the gravity wrench can be further defined as

$$W_{grv}^{p,p} = G_p^p [Ad_{H_0^p}] A_{grv}^0 \quad (4.2)$$

Where $A_{grv}^0 = (0, a^0) \in \mathbb{R}^6$ with $a^0 \in \mathbb{R}^3$. The gravity wrench is applied in the CG, which explains the presence of the inertia tensor G_p^p .

In a similar manner to equation eq. (4.1), the parent external interactions are also illustrated in figure 4.2.

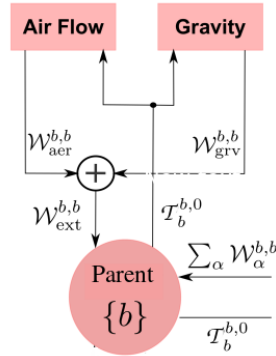


Figure 4.2: Parent subsystem block diagram. Modified from [Hong et al. (2021)].

Parent interconnection ports:

As it was explained along the theory chapter 3, the *parent* subsystem is treated as a Dirac structure, with all power ports looking inwards (as inputs). In figure 4.3 there are two power ports. The so called "children" is the connection to all the children's through intermediate joints. "children" represent the input wrench sum of all *children's* exerted wrenches into the main body. On the other side, the "air_flow" power port represent the external forces, that in the case of a flapping-UAV would be the air, this is, the aerodynamic forces of the body moving with respect to the air. If another external force was applied to the main body this could be directly summed up to the aerodynamic forces.

The other two signal ports present in 4.3 are signals required by the *joints* to compute the absolute positions of the *children* with respect to the world ($H_c^0 = H_p^0 H_c^p$), as well as the dynamics of the *child*, as it requires the rate of change of the *parent's* twist $\dot{V}_p^{p,0}$ to obtain the rate of change of the *children* without using numerical differentiation (equation (4.9)). This is very important, because numerical differentiation can be very costly computationally and add inaccuracies, so it should be avoided. All parent ports are related with their represented variables in table 4.1.

Port Name	Variable	In/Out	Type
air_flow	$W_{aer}^{p,p} / T_p^{p,0}$	Input/Output	Power-port
children	$W_c^{p,p} / T_p^{p,0}$	Input/Output	Power-port
V_p_0_p_dot	$\dot{V}_p^{p,0}$	Output	Signal-port
H_0_p	H_p^0	Output	Signal-port

Table 4.1: Parent subsystem ports.

Lastly, one of the terms taken into account in equation (4.1) is the parent's gravity. This can be directly included inside the code of the *parent* block, so no external ports are required.

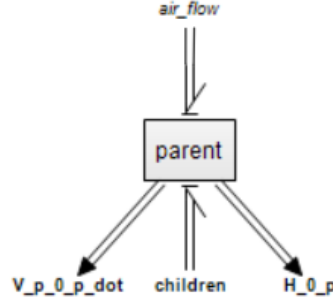


Figure 4.3: Parent subsystem

Parent parameters:

The only parameters the *parent* requires to be specified are:

- the **initial homogeneous matrix**, that defines the initial position and orientation of the parent with respect to the world $H_0^P(0)$,
- the **generalized inertia matrix** G^P of the parent, that is dependent on the mass, m , and shape, i (inertia), of the body, and
- a **gravity acceleration vector** A_{grv} , that in most cases will simply have a negative z acceleration ($-g$). This last acceleration term is the one included in equation 4.2.

Parameter	Size	Shape
$H_0^P(0)$	4x4	$\begin{pmatrix} R_0^P(0) & o_0^P(0) \\ 0 & 1 \end{pmatrix}$
G^P	6x6	$\begin{pmatrix} i_x & 0 & 0 & 0 & 0 & 0 \\ 0 & i_y & 0 & 0 & 0 & 0 \\ 0 & 0 & i_z & 0 & 0 & 0 \\ 0 & 0 & 0 & m & 0 & 0 \\ 0 & 0 & 0 & 0 & m & 0 \\ 0 & 0 & 0 & 0 & 0 & m \end{pmatrix}$
A_{grv}	6x1	$[0, 0, 0, 0, 0, -g]^T$

Table 4.2: Parent subsystem tuneable parameters

4.3 Child Subsystem

The *child* subsystem is used to represent any body different from the main body (*parent*), that in the case of a flapping wing UAV would be the different sections of the wings, tails, legs, etc. The *child's* dynamic equation is in differential causality form, this is, it is expressed as the reaction wrench of the *parent* exerted on the *child's* body expressed in the *child's* frame c , and results in:

$$W_p^{c,c} = G_c^c \dot{T}_c^{c,0} - [ad_{T_c^{c,0}}]^T G_c^c T_c^{c,0} - W_{ext}^{c,c} - \sum W_j^{c,c} \quad (4.3)$$

Where, similar to the case of the *parent* dynamic equation, the wrench $W_{ext}^{c,c}$ represent any external wrench acting on the *child*, this is, in the case of a flapping bird (*child* could be for

instance a wing), the gravity force $W_{grv}^{c,c}$ and the aerodynamic forces $W_{aer}^{c,c}$ ($W_{ext}^{c,c} = W_{grv}^{c,c} + W_{aer}^{c,c}$). Again, the aerodynamic forces won't be derived further, since that's not a target of this thesis, but the gravity force acquires the same shape as exposed in equation (4.2), only this time w.r.t. the *child* and equally applied on its CG, so

$$W_{grv}^{c,c} = G_c^c [Ad_{H_0^c}] A_{grv}^0 \quad (4.4)$$

On the other hand, the term $\sum W_j^{c,c}$ refers to the sum of the wrenches of the j children the *child* might have. If it is the case that the *child* don't have any other child connected underneath it, then this last term of eq. (4.3) will be ignored (equal to 0).

At last, in equation (4.3) appears the twist of the child $T_c^{c,0}$ and the rate of change of the child's twist $\dot{T}_c^{c,0}$ that are two inputs from the joint.

The equivalent block diagram of the *child* subsystem is shown in figure 4.4 with slightly different nomenclature, with 'b' referring to the parent (main body) and ' α ' referring to the child.

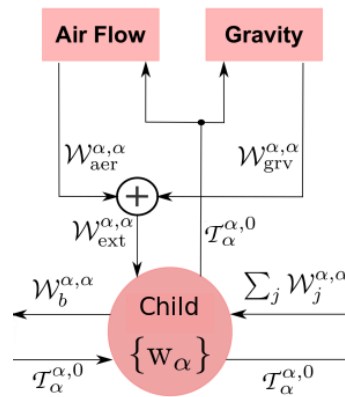


Figure 4.4: Child subsystem block diagrams. Modified from [Hong et al. (2021)].

Child interconnection ports:

The *child* subsystem, shown in figure 4.5, requires three power ports, equivalent to the pairs of signals shown in figure 4.4 and two additional signals.

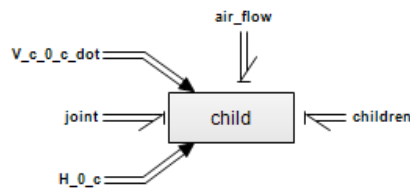


Figure 4.5: Child subsystem

Most *child* power ports and signals are very similar to those present in the parent but opposite in the case of the signals. The most important power port is the one coming from the *parent* and transformed by the *joint* to the *child*'s frame, named "joint" in figure 4.5. This port represents the exerted force by the *child* on the *parent* expressed in the *child*'s coordinate frame.

Next, the "air flow" port represents, as in the *parent*'s case, the external forces, that for an UAV are the aerodynamic forces of that body (e.g. a wing). Other external forces can be summed up with the aerodynamic effort and input through the "air flow" port if necessary.

The right-most power port, the "children" port, represent the external wrench the next children in the chain/branch exert on the *child*. This wrench adds up to the other external wrenches inside the block like "air flow". There are two case scenarios for the *child*. If the *child* is the last element of the branch, a leaf, meaning that there are no other subsystems connected after it. Then there is no need for the "children" port, or a 0-wrench source has to be connected to it.

As in the *parent*, the gravity term, which appears in the diagram of figure 4.4, is taken into account inside the code of the *child* block, so no external ports are required.

In what respects to signals, the *child* receives the rate of change of its own twist, $\dot{V}_c^{c,0}$, and its actual configuration with respect to the world, H_c^0 , from the *joint*, this is, the computation of these variables is done inside the *joint* and not the *child*.

All ports are related with their represented variables in table 4.3.

Port Name	Variable	In/Out	Type
air_flow	$W_{aer}^{c,c} / T_c^{c,0}$	Input/Output	Power-port
joint	$W_p^{c,c} / T_c^{c,0}$	Input/Output	Power-port
children	$W_j^{c,c} / T_c^{c,0}$	Input/Output	Power-port
V_c_0_c_dot	$\dot{V}_c^{c,0}$	Input	Signal-port
H_0_c	H_c^0	Input	Signal-port

Table 4.3: Child subsystem ports.

Child parameters:

The *Child* subsystem require the specification of only two parameters:

- **the generalized inertia matrix** G^c , that is dependent on the mass, m , and shape, i (inertia), of the body, and
- a **gravity acceleration vector** A_{grv} , that in most cases can be set by default as a negative gravity acceleration in Z.

Parameter	Size	Shape
G^c	6x6	$\begin{pmatrix} i_x & 0 & 0 & 0 & 0 & 0 \\ 0 & i_y & 0 & 0 & 0 & 0 \\ 0 & 0 & i_z & 0 & 0 & 0 \\ 0 & 0 & 0 & m & 0 & 0 \\ 0 & 0 & 0 & 0 & m & 0 \\ 0 & 0 & 0 & 0 & 0 & m \end{pmatrix}$
A_{grv}	6x1	$[0, 0, 0, 0, 0, -g]^T$

Table 4.4: Child subsystem tuneable parameters

4.4 Joint Subsystem

The *joint* is the element through which *parent* and *child* are attached together and communicate or transmit the energy with each other. It transforms the efforts and flows from one body coordinate frame to another coordinate frame.

As it has been mentioned many times already, the *joint* is interpreted as a Dirac structure. Figure 4.6 depicts the derivation from a regular bond-graph representation of a basic *joint* with two outputs into a Dirac structure representation. The *joint* can therefore be expressed as a matrix that transforms the twists and wrenches from one frame to another and which sum of powers must be always equal to 0.

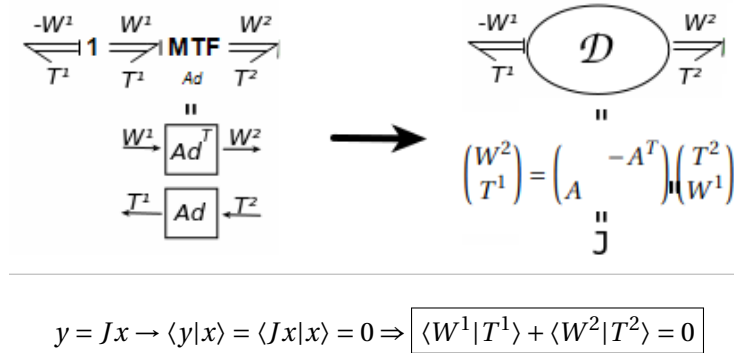


Figure 4.6: From bond-graphs to Dirac structure.

Furthermore, in the presence of additional external forces (dual τ and ω) the Dirac structure would look as shown in figure 4.7 and again, the sum of input/output power is equal to 0.

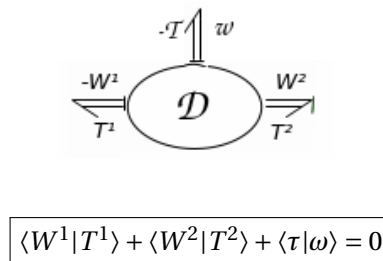


Figure 4.7: Dirac structure with external forces.

This last construction corresponds to the *joint's* configuration used along this modelling and simulation framework, where the left and right ports are connected to the *parent* and *child* and the third is the actuator's input. The *joint* matrix in this case acquires the following shape:

$$J_c^p(\theta_c) := \begin{bmatrix} 0 & -[Ad_{H_p^c}]^T & 0 \\ [Ad_{H_p^c}] & 0 & [Ad_{H_p^c}]S_c^{p,p} \\ 0 & (S_c^{p,p})^T [Ad_{H_p^c}]^T & 0 \end{bmatrix} \quad (4.5)$$

This *joint* matrix contains two distinctive elements. The unit twist $S_c^{p,p}$, that defines the degree of freedom of the joint, and the Adjoint $Ad_{H_p^c}$, that transforms the *child's* wrench in the *child's* frame into the *parent's* frame.

The *joint* matrix describes the interconnection between bodies (*parent* and *child*) and the actuator and it can be depicted in a diagram form as shown in figure 4.8.

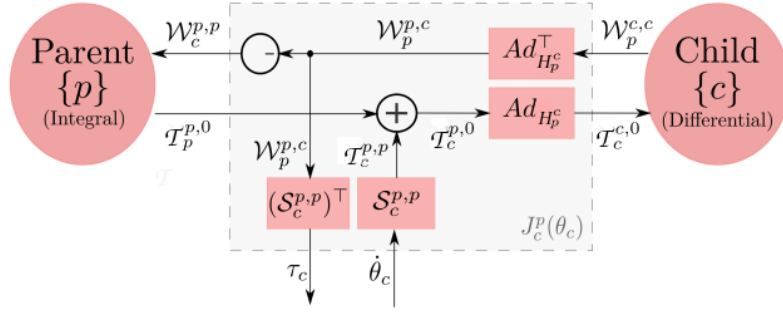


Figure 4.8: Parent-Joint-Child diagram. Modified from [Hong et al. (2021)].

The two basic possibilities for the *joint* unit twists are depending on the desired type of motion (from Mozzi's theorem eq.(3.1)):

1. Rotational (and screw):

$$S_c^{p,p} = \begin{pmatrix} \hat{\omega} \\ r \wedge \hat{\omega} \end{pmatrix} \quad (4.6)$$

2. Translational:

$$S_c^{p,p} = \begin{pmatrix} 0 \\ \hat{v} \end{pmatrix} \quad (4.7)$$

These not only are necessary to construct the *joint* matrix, but also are used to compute the configuration of the *child* wrt the *parent* at each time step employing Brocket's exponential formula (**Direct kinematics**) as:

$$H_c^p(q) = e^{\hat{S}_c^{p,p} q} \cdot H_c^p(0) \quad (4.8)$$

Where the exponential of the unit twist multiplied by the coordinate q would represent the actual state of the body and $H_c^p(0)$ is the initial configuration.

The updated configuration of the *child* wrt the *parent*, $H_c^p(q)$, is in turn needed for computing the Adjoint $Ad_{H_p^c}$ required to finish building the *joint* matrix of eq.(4.5).

Additionally, inside the joint subsystem the rate of change of the child's twist $\dot{T}_c^{c,0}$ is obtained mathematically avoiding the use of numerical differentiation by applying:

$$\dot{T}_c^{c,0} = [ad_{T_c^{c,0}}] S_c^{c,p} \dot{\theta}_c + [Ad_{H_p^c}] \dot{T}_p^{p,0} + S_c^{c,p} \ddot{\theta}_c \quad (4.9)$$

This is a very important and convenient equation, since numerical differentiation tend to be one of the biggest problems for modelling computing-efficient systems with energy-based methods. Such computation of $\dot{T}_c^{c,0}$ requires $S_c^{c,p}$ that can be obtained from $S_c^{p,p}$ with a simple transformation

$$S_c^{c,p} = Ad_{H_p^c(0)} \cdot S_c^{p,p} \quad (4.10)$$

Finally, the configuration of the child wrt the world is obtained inside the joint as well as:

$$H_c^0 = H_p^0 \cdot H_c^p \quad (4.11)$$

Moreover, the *joint* is the element that determines the relation between *parent* and *child* through the *actuator*, that inserts a relative velocity term as follows:

$$T_c^{c,0} = T_p^{p,0} + T_c^{c,p} = [Ad_{H_p^c}](T_p^{p,0} + T_c^{p,p}) = [Ad_{H_p^c}](T_p^{p,0} + S_c^{p,p} \dot{\theta}_c) \quad (4.12)$$

If $\dot{\theta}_c = 0$ then the child is fixed to the parent.

Nevertheless, when considering the flapping of most insects, bats and birds, it is clear that the motion is a combination of several rotations, in a similar way to a shoulder, that can rotate around X, Y and Z. Therefore many systems will require the modelling and simulation of *joints* with more than one dof. It is in this context that appears the concept of a joint with multiple degrees of freedom, that from now on will be referred as **multi-dof joint**.

Multi-degrees of freedom joint

The *multi-degrees of freedom joint* is meant to simplify the construction of some models, as it permits to avoid the implementation of several independent *joints* one on top of the other. This *multi-dof joint* comprises six simple *joints*, each one representing one dof, all together in one single subsystem. It can actually be interpreted as a concatenation of *joints*, so the relative pose of the child wrt the parent can still be computed with Brockett's exponential formula as done before for the simple *joint* in eq.(4.8), but with six coordinates instead $(q_1, q_2, q_3, q_4, q_5, q_6)$, as follows:

$$H_6^0(q_1, q_2, q_3, q_4, q_5, q_6) = e^{\tilde{S}_1^{0,0} q_1} \cdot e^{\tilde{S}_2^{0,1} q_2} \cdot e^{\tilde{S}_3^{0,2} q_3} \cdot e^{\tilde{S}_4^{0,3} q_4} \cdot e^{\tilde{S}_5^{0,4} q_5} \cdot e^{\tilde{S}_6^{0,5} q_6} \cdot H_6^0(0) \quad (4.13)$$

Where the exponential of the unit twists can be computed using Rodriguez's formula (eq.(3.5) and eq.(3.6)). The exponential of the rotational unit twists can be obtained from eq.(3.8), while the exponential of the translational unit twists can be obtained from eq.(3.9).

Each unit twist in tilde form, $\tilde{S}_c^{p,p}$, represents each one of the "pure" degrees of freedom $(\theta_x, \theta_y, \theta_z, o_x, o_y, o_z)$ of the *multi-dof joint*. Thus, the main change compared to the simple *joint* is the definition of such unit twists, since the six unit twists are now put together consecutively as the columns of a 6x6 matrix.

$$S_c^{p,p} = (S_1^{0,0} \quad S_2^{0,1} \quad S_3^{0,2} \quad S_4^{0,3} \quad S_5^{0,4} \quad S_6^{0,5})$$

$$S_c^{p,p} = \begin{pmatrix} \tilde{\omega}_x & \tilde{\omega}_y & \tilde{\omega}_z & \begin{pmatrix} 0 \\ \tilde{v}_x \end{pmatrix} & \begin{pmatrix} 0 \\ \tilde{v}_y \end{pmatrix} & \begin{pmatrix} 0 \\ \tilde{v}_z \end{pmatrix} \\ r \wedge \tilde{\omega}_x & r \wedge \tilde{\omega}_y & r \wedge \tilde{\omega}_z & & & \end{pmatrix}$$

$$S_c^{p,p} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ r_{joint}^p \wedge \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} & r_{joint}^p \wedge \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} & r_{joint}^p \wedge \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

where $r_{joint}^p = [r_x, r_y, r_z]^T$ is the relative position of the joint with respect to the parent as depicted in figure 4.9.

Additionally, all translations can be put together when computing the exponential of the translational dof's, since

$$\begin{cases} \hat{v}_{C_{translationX}}^{p,p} = (1, 0, 0)^T \\ \hat{v}_{C_{translationY}}^{p,p} = (0, 1, 0)^T \\ \hat{v}_{C_{translationZ}}^{p,p} = (0, 0, 1)^T \end{cases}$$

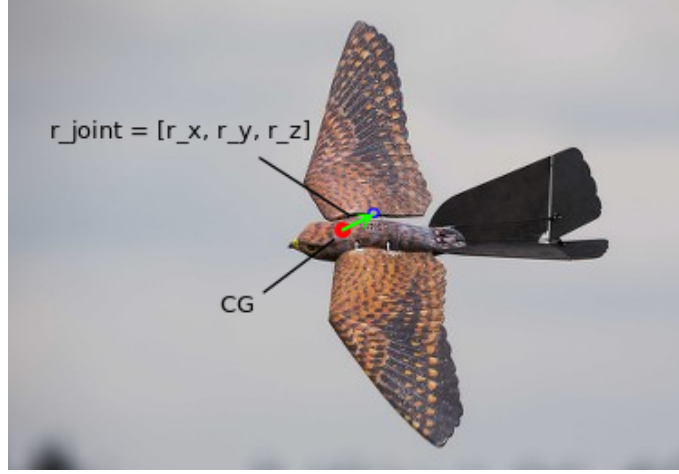


Figure 4.9: Example of $r_{joint}^{p,p}$ vector, in green. CG of the main body (parent) in red. The wing's joint location is indicated in blue.

so the homogeneous matrix of the exponentials of these translations is obtained using eq.(3.9) and results in:

$$H_c^p(q)_{translation} = \begin{pmatrix} I & \hat{v}_{c_{transX}}^{p,p} O_x \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} I & \hat{v}_{c_{transY}}^{p,p} O_y \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} I & \hat{v}_{c_{transZ}}^{p,p} O_z \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} I & \begin{pmatrix} O_x \\ O_y \\ O_z \end{pmatrix} \\ 0 & 1 \end{pmatrix} \quad (4.14)$$

Finally, the definitive configuration of the child with respect to the parent can be expressed as:

$$H_c^p(q) = \begin{pmatrix} I & \begin{pmatrix} O_x \\ O_y \\ O_z \end{pmatrix} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} e^{\tilde{\omega}\theta_z} & (I - e^{\tilde{\omega}\theta_z})r \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} e^{\tilde{\omega}\theta_y} & (I - e^{\tilde{\omega}\theta_y})r \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} e^{\tilde{\omega}\theta_x} & (I - e^{\tilde{\omega}\theta_x})r \\ 0 & 1 \end{pmatrix} \cdot H_c^p(0) \quad (4.15)$$

All the equations describing the interactions between parent and child previously explained for the simple *joint*, eq.(4.5), eq.(4.9), eq.(4.10) and eq.(4.12), will be exactly the same and apply similarly for the *multi-dof joint*, but taking into account that the unit twist $S_c^{p,p}$ element is now a 6x6 matrix and the input from the actuator becomes a 6x1 array (each input actuates one degree of freedom). Thus, the result of multiplying the matrix of unit twists with the input array will be the sum of the effects of each input.

Joint interconnection ports

All the ports of the *joint* subsystem are illustrated on figure 4.10 and table 4.5. The three energy ports represent, as already mentioned, the *parent*, the *child* and the *actuator*. From the picture it becomes also clear that the *parent* is in integral causality form while the *child* is in differential causality form. The other four signals are the twists rates of change of the *parent* $\dot{V}_p^{p,0}$ as input and the *child* $\dot{V}_c^{c,0}$ as output, frame transformed by the joint, and the absolute configurations of the *parent* H_p^0 and the *child* H_c^0 , also frame transformed by the joint.

The only difference in ports between the *joint* and the *multi-dof joint* is the actuator, that instead of being a single power port twist input in the case of the *multi-dof joint* it would become a 6x1 power port array representing six velocity inputs $[\omega_x, \omega_y, \omega_z, v_x, v_y, v_z]$, where each input actuate a degree of freedom, three rotations and three translations.

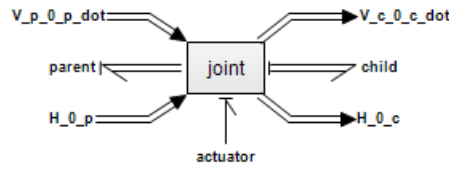


Figure 4.10: Joint subsystem.

Port Name	Variable	In/Out	Type
parent	$W_c^{p,p} / T_p^{p,0}$	Output / Input	Power-port
child	$W_p^{c,c} / T_c^{c,0}$	Input / Output	Power-port
actuator	$\tau_c / \dot{\theta}_c$	Output / Input	Power-port
V_p_0_p_dot	$\dot{V}_p^{p,0}$	Input	Signal-port
V_c_0_c_dot	$\dot{V}_c^{c,0}$	Output	Signal-port
H_0_p	H_p^0	Input	Signal-port
H_0_c	H_c^0	Output	Signal-port

Table 4.5: Joint and multi-dof joint subsystems ports.

There are three types of joints that can be represented with the *joint* subsystem, a revolute joint, a screw joint and a prismatic joint, shown in figure 4.11. The differentiation between them is intrinsic in the definition of the unit twist, that specifies the dof between bodies. If the unit twist has both, a linear \hat{v} and a rotational $\hat{\omega}$ velocity components, then the joint can be either a screw joint or a revolute joint, but if there is only a linear velocity \hat{v} component, it will be a prismatic joint.

$$\begin{cases} \text{Screw or Revolute joint} \rightarrow S_c^{p,p} = (\hat{\omega}, \hat{\omega} \wedge \hat{v})^T \\ \text{Prismatic joint} \rightarrow S_c^{p,p} = (0, 0, 0, \hat{v})^T \end{cases}$$

On the other hand the *multi-dof joint* subsystem allow the representation of those three one dof joints plus the cylindrical, universal and spherical joints, shown in figure 4.12.

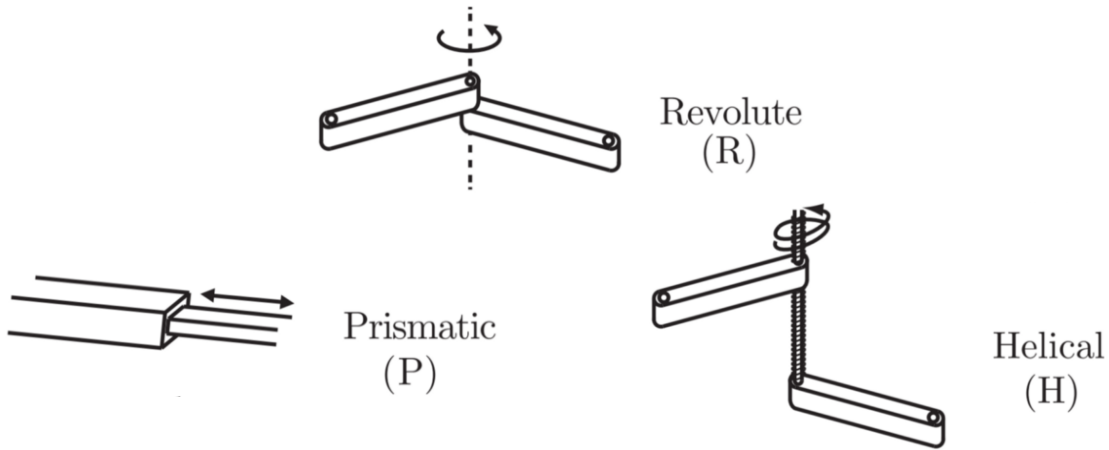


Figure 4.11: One DOF joint types are revolute (rotational), prismatic (translational) and helical (screw).

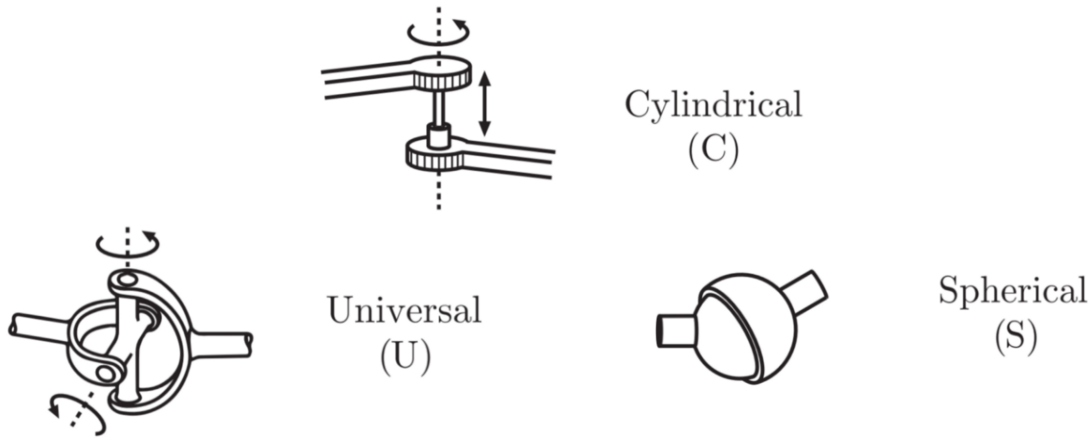


Figure 4.12: Multi-DOF joint types are cylindrical (one rotational DOF + one translational DOF), universal (three rotational DOF) and spherical (three rotational DOF).

Joint parameters:

The parameters required for the *joint* subsystem to work are:

- the **initial homogeneous matrix**, that defines the initial position and orientation of the child wrt parent $H_c^p(0)$,
- the **unit twist of the child wrt parent**, $S_c^{p,p}$, that defines the degree of freedom of the joint.
- (optional) $q(0)$ or q_{init} and $\ddot{q}(0)$ or $q_{init_dot_dot}$ are the initial values of the joint's position and acceleration.

For the case of a multi-dof joint the configuration parameters are mostly the same as for the regular joint subsystem with exception of the unit twist. The previously required unit twist, that already included the distance from the parent to the joint in its definition, is substituted by simply that distance from the parent to the joint:

- the **position of the joint with respect to the parent frame**, r_{joint}^p .

$$S_c^{p,p} = \begin{pmatrix} \hat{\omega} \\ r_{joint}^p \wedge \hat{\omega} \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} \hat{\omega}_x \\ \hat{\omega}_y \\ \hat{\omega}_z \end{pmatrix} \\ \begin{pmatrix} r_{joint_x}^p \\ r_{joint_y}^p \\ r_{joint_z}^p \end{pmatrix} \wedge \begin{pmatrix} \hat{\omega}_x \\ \hat{\omega}_y \\ \hat{\omega}_z \end{pmatrix} \end{pmatrix} \rightarrow \text{Is Simplified to: } r_{joint}^p = \begin{pmatrix} r_{joint_x}^p \\ r_{joint_y}^p \\ r_{joint_z}^p \end{pmatrix}$$

Finally the only required parameters for the joint and multi-dof joint are shown in table 4.6.

Joint			Multi-dof joint		
Parameter	Size	Shape	Parameter	Size	Shape
$H_c^p(0)$	4x4	$\begin{pmatrix} R_0^p(0) & o_0^p(0) \\ 0 & 1 \end{pmatrix}$	$H_c^p(0)$	4x4	$\begin{pmatrix} R_0^p(0) & o_0^p(0) \\ 0 & 1 \end{pmatrix}$
$S_c^{p,p}$	6x1	$\begin{pmatrix} \hat{\omega} \\ r_{joint}^p \wedge \hat{\omega} \end{pmatrix}$	r_{joint}^p	3x1	$[r_{joint_x}^p, r_{joint_y}^p, r_{joint_z}^p]^T$

(a) Joint subsystem parameters (b) Multi-dof joint subsystem parameters

Table 4.6: Joint and multi-dog joint tuneable parameters.

4.5 Compact Child (joint + child)

A *child* structure will always require a *joint* in order to be attached to a *parent*. Therefor it makes sense to put both subsystems together under one single subsystem. The *child-joint* subsystem, depicted in figure 4.13, will be simply refereed as (compact) **Child** from now on for simplicity. Building the models this way saves space and time, as the ports between *joint* and *child* will be already connected. There are no internal changes in the *joint* nor the *child*, but it is just a mere abstraction of those interconnected subsystems.

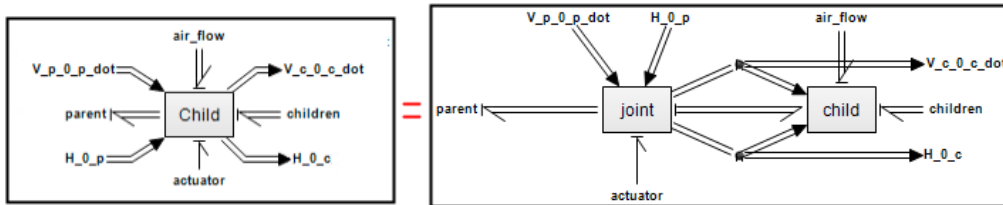


Figure 4.13: Joint-Child compact form subsystem (left) and expanded form (right).

4.6 Conclusion

The presented framework was explained during this chapter. It consists on a port-Hamiltonian framework, so all elements can be defined as Dirac structures that interconnect trough power ports. The dynamic equations ruling each type of body, *parent*, *child* or *joint* are developed. The connection ports are illustrated and the configuration parameters are explained.

The basic parent-joint-child diagram interconnected by means of power ports can be expressed as in figure 4.14.

Finally, it can be concluded that the dynamic equations, while probably difficult to understand, are rather simple to write, since these only consist on one equation for the *parent*, eq.(4.1), and one for the *child*, eq.(4.3) (two each if taking into account the gravity term). It can be considered that the *joint* contains all the mathematical complexity, as it takes care of several coordinate frame transformation operations and the kinematics.

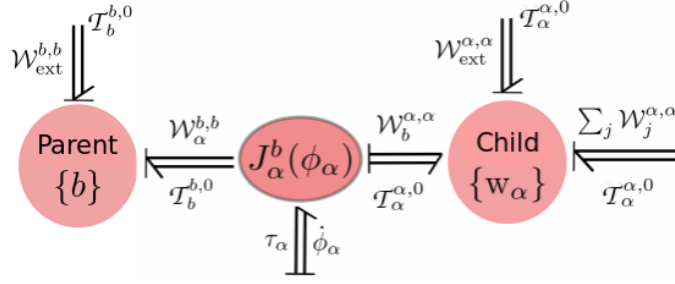


Figure 4.14: Bond-graph schematic of a parent-joint-child system represented as Dirac structures. Modified from [Hong et al. (2021)].

Regarding the tuning parameters, these are very reduced. Ignoring the acceleration vectors that should be in most cases a default value, the *parent* requires the definition of only two parameters, its initial configuration (H_0^p) and its generalized inertia matrix (G^p), while the *child* requires only one, its generalized inertia matrix (G^c). Similarly, the *joint* requires the specification of two parameters, the initial configuration of the child wrt the parent (H_c^p) and the unit twist defining the dof of that joint ($S_c^{p,p}$), unless it is a *multi-dof joint*, in which case it requires the distance from the *parent* to the joint (r_{joint}^p) instead.

Moreover, a joint with multiple degrees of freedom (up to six) is developed in this chapter. It becomes very convenient when modelling complex joints, like an universal or spherical joint. With the simple *joint* implementing an universal type of joint would require three *joints* plus three *child* subsystems symbolically located at the same physical point. Conversely, with a *multi-dof joint* this would be reduced to one *multi-dof joint* and one *child*. Not only there is a reduction in blocks, but also in configuration parameters, as the unit twists can be tedious. With simple *joints* it would be necessary to specify as many unit twists as dofs. However, the *multi-dof joint* only requires the position of the joint wrt the parent, r_{joint}^p , no matter how many dofs the joint has.

The *multi-dof joint* can clearly be advantageous in many situations, but of course, it will most likely be more cost-computing, since it requires multiplying larger matrices than the simple *joint*.

Ultimately, a compact *Child*, which comprehends a *joint* plus a *child* is introduced. This construction make sense as the inclusion of a *child* in a model will always require a *joint* as well, so both can be put under one single subsystem. This is already a demonstration of the port-Hamiltonian framework potential, since thanks to the approach characteristics, the newly created subsystem, as a result of unifying two port-Hamiltonian subsystems, it's also a port-Hamiltonian subsystem.

5 Morphing-Wing Robots Case Studies

During this section a set of several experiments and model constructions is presented along with a discussion of the different observations during the process and simulation. Several literature models are imitated, sometimes in a rough way, adding dofs that the original models didn't have, seeking for comparison and enhancement of the ease of use and broad possibilities the parent-child modelling paradigm may bring us. Divers case studies are selected, with different characteristics each, that should exemplify the functioning of the modelling and simulation framework. In some cases the parameter values are known and used, in order to compare the results of the models more closely, while in others the parameters are irrelevant. What the reader will see in most case studies is the dynamic behaviour of the models in a gravity free environment, since there is no air in the simulations.

But first of all, an introduction into the programming environment 20-Sim is provided with an explanation of some elements that will be extensively used along the different case studies.

It must be mentioned, that the case studies are presented by level of complexity, trying to include new elements progressively, starting from the most basic models and finishing with the most complex/illustrative ones. The first model, a flapping-wing base model, won't be reflecting an imitation of any particular case study, but it is meant to serve as an introductory model, base for all the rest, and to validate the subsystems energy consistency and dynamics.

All models and simulations are built and run on a windows 7 virtual machine with 4 processors 10GB of RAM dedicated for the system on top of Ubuntu 22.

5.1 20-Sim Environment

20-Sim is the software selected for the implementation of the models, as it allows the modelling of port-Hamiltonian multi-domain systems in a schematic/graphical way, by means of bond-graphs, equations, block diagrams or just physics blocks. In our case, we are very much interested in the port-Hamiltonian bond-graph and equation representation, but all the formats are compatible one with each other. The software also includes a simulation environment, with both plots and 3D visualizations.

The capabilities of 20-Sim makes it very suitable for our purposes, as it provides very intuitive tools for implementing the parent, joint and child subsystems, enabling modularity of the models as well as allowing a port-based implementation. Thus, a balanced energy exchange between elements can be ensured.

Also, as it will be demonstrated later on, the modularity of the subsystems make 20-Sim optimal for creating new models rather fast, adding different actuators, new bodies or new motions easily. Besides, the control laws should be easily applicable as well, but that won't be part of this study.

The 20-Sim implementation is tackled in detail along appendix A, including an explanation of the construction of a basic model in A.2 and the implementation of the subsystems and the definition of the parameters in A.4.

5.1.1 Generalities of the 20-Sim models

All models will be built using the **(compact) *Child*** form (joint + child) introduced in section ?? for simplicity. Building the models this way saves time and space, as it even allow us to configure the hidden *joint* and *child* blocks together from the abstracted block *Child*. All flapping-wing UAV systems should be modelable using only the *Parent* and (compact) *Child* subsystems plus the actuators.

Actuators are external elements, independent from the modelling method developed throughout this document, but they are always necessary. The actuators possibilities are very broad, as it will be demonstrated along this chapter.

Additionally, a so called **World Attachment** will be used for almost all models. This is an element that imitates a test-bed platform. It enables/disables the 6 degrees of freedom of the whole model. It permits performing very different kind of tests liberating only the degrees of freedom of interest. The subsystem is very simple, consisting on a resistance-capacitor combination, as shown in figure A.8, which allow the control of the compliance of the test bed. It can be connected to the parent through a *One-junction*, as it will be shown repeatedly in the models.

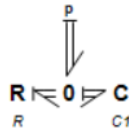


Figure 5.1: World attachment/test-bed modelling.

Finally, it should be mentioned that many **wing angle/position, q , graphs are in relative values**, since the graphs will all start from 0° when the initial position of the bodies might be different. This is, many angle/position graphs are drifted due to the way the initial configuration is implemented, that instead of specifying an initial angle q_0 to the integrator of the joint velocity the initial angle is specified in the initial configuration matrix $H_c^p(0)$.

5.2 Flapping-Wing Base Model. Subsystems validation.

This first model represent the most basic configuration possible. It simply consists on a body with two flapping wings. It doesn't correspond with any particular flapping-wing robot in the literature, but it is the base of all of them, as it will be demonstrated during the rest of the examples. Additionally, this model will be used to validate the subsystems energy consistency and dynamics.

The 20-Sim model is shown in figure 5.2a and in 3D in figure 5.2b. At first sight one can distinguish three main blocks, that are the body and the two wings. The body is represented by a *parent* subsystem, from which all other subsystems hang, connected through a one-junction that represent the sum of the wrenches of the two children. Both *Child* blocks, "Wing R" and "Wing L", as they are leaves, require a 0-wrench source connected to their "children" ports, as to indicate that there are no more subsystems underneath them.

Also the children, as they are in compact form (as explained in section 5.1.1) include the *joint* inside, which have the actuators power port. In this particular example, the actuators is a wave generator used as a velocity profile generator.

Direct velocity profile into joint. It is probably the most intuitive actuator. A direct velocity profile implemented as a wave generator, since it allows the user to specify a velocity or position path by means of an equation, with very little parameters to tune, therefor leading to a rather fast implementation of the model. The modulated sources of flow present in figure 5.2a is a mere transducers from signal to power port, that produce, in this case, rotational velocities.

The dof of the joint is intrinsic in the definition of the unit twist S of each *Child*.

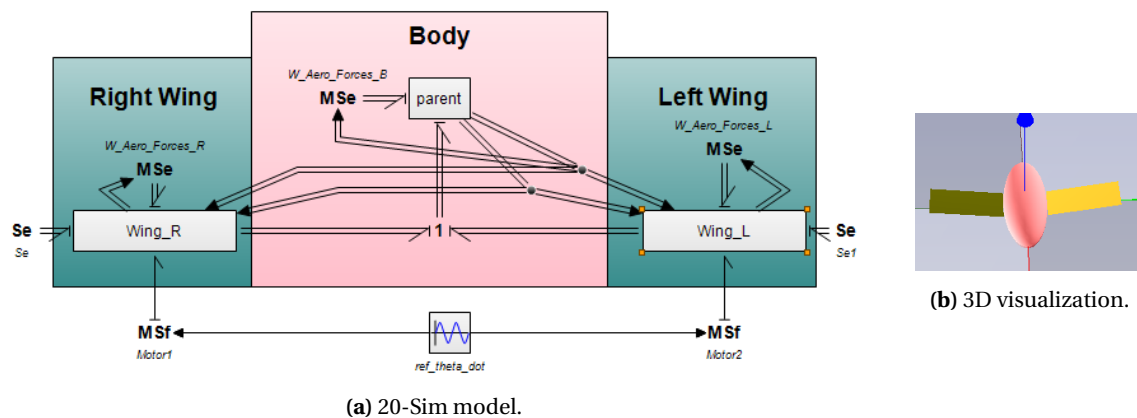


Figure 5.2: Basic flapping-wing robot model.

(File: "\\Parent_Child_Modelling\Other Models (tests)\Bird_Basic_Gcomp_test.emx").

The last elements present in the model are the modulated source of effort (MSe) that represent the air flow or the aerodynamic forces, which in our case are not implemented. Instead, as we would like to test the dynamics in a gravity free, air free environment what we are doing is counteracting the gravity in each body. The signal into the MSe contains the configuration of the body wrt the world, so that the weight of the body can be counteracted. Truth is, there are other easier ways to counteract gravity, but this is just a possibility (refer to appendix A.2.3 for more info on the topic).

Energy consistency validation

Finally, with this first model we can already demonstrate that the method is energy consistent, by looking into the power and energy balance. The first is computed in the *joints*, as the sum of all powers inside the joint. The second is computed in the *child* subsystems, as the difference

between kinetic co-energy and the integral of the body's power. The mathematics behind this were introduced in section 3.2.3.

The graphical results of the power and energy balance during a constant synchronous flapping motion are shown in figure 5.3. The top graph shows the sinusoidal motion of the wings. From the graphs we can see that both power and energy tend to zero, even though there is always a slight numerical error. It is clear it is a numerical error because the curve is random along the hundred seconds the simulation lasts. The power error, as it is a result of summing the twist and wrench products it will have the range of the floating point data precision. Conversely, the energy is the result of integrating the power, thus the error is in the range of the maximum integration error, that is normally set around $1e^{-5}$ or $1e^{-6}$. We can therefore verify that all three subsystems *parent*, *child* and *joint* are properly built and that simulations with this method seem to be energy consistent.

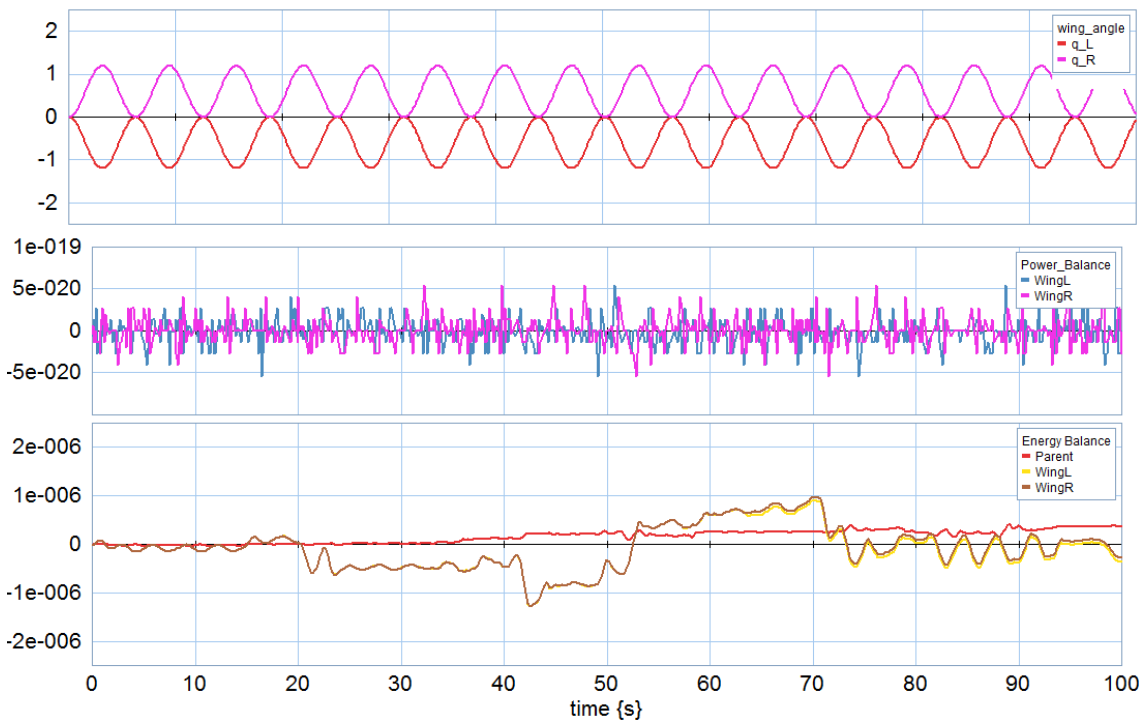


Figure 5.3: Power and energy balance of the bodies during a flapping motion in a basic flapping-wing robot model.

(File: "\Parent_Child_Modelling\Other Models (tests)\Bird_Basic_Gcomp_test.emx").

Dynamics validation

With this same model we can already validate the correct functioning of the subsystems. As there is no air in the simulation (nor gravity), the body dynamics can be validated by checking the inertial forces generated during the flapping movement.

Figure 5.4 show the graphs of the main body's momentum (*parent*) against the motion of the wings and the position of the body. In the momentum graph we can see that the only generated momentum is in the linear Z direction, since both wings are flapping synchronously. At the beginning of the movement, as the wings are moving downwards (in this case absolute increasing angle means downstroke), the momentum in Z becomes positive (action-reaction). The downwards movement of the wings is exerting an upwards force onto the main body. Therefore, the body climbs while the momentum is positive, this is, until the orange line. During the upstroke,

the momentum becomes negative and the robotic bird goes back down to its original position (green line).

The momentum that appears in the model are due to the inertial forces. With this we can validate the consistency of the subsystem's dynamics.

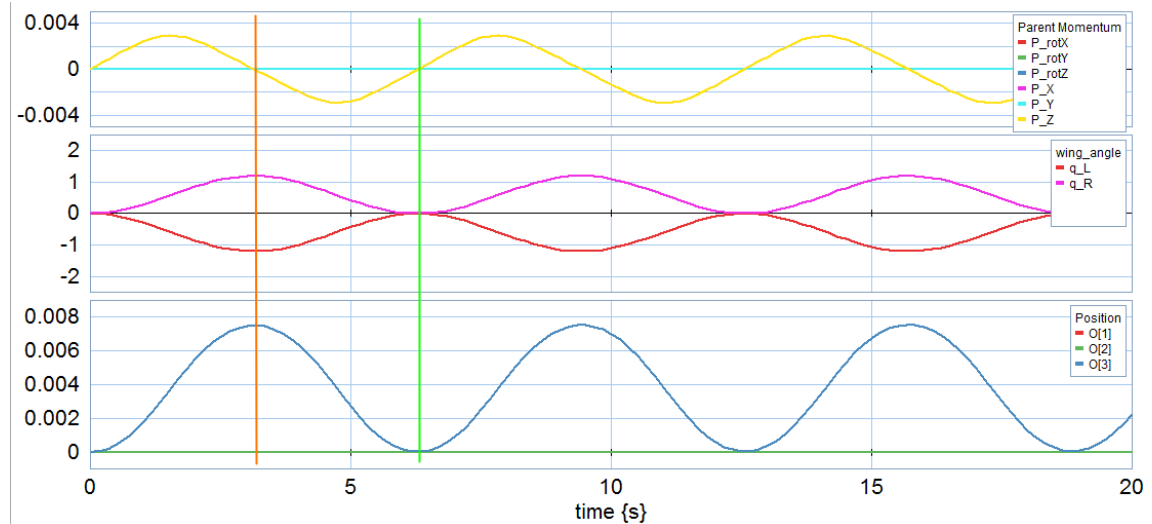


Figure 5.4: Up-down movement of the main body (bottom graph) due to the inertial forces exerted by the flapping movement (middle graph), reflected in the main body momentums (top graph). (File: "\Parent_Child_Modelling\Other Models (tests)\Bird_Basic_Gcomp_test.emx").

5.3 Gull wing

The Gull wing is a very extended kind of flapping-wing UAV, where the possibility of changing the shape of the wings asymmetrically can be exploited to control the attitude of the UAV both while gliding and flapping. In this particular case the turning of the bird is demonstrated while gliding as studied by [Guo et al. (2016)], but the model can flap the wings as well. Figure 5.5 show the comparison between a real gull, the model presented by [Guo et al. (2016)] and the 20-sim 3D version.

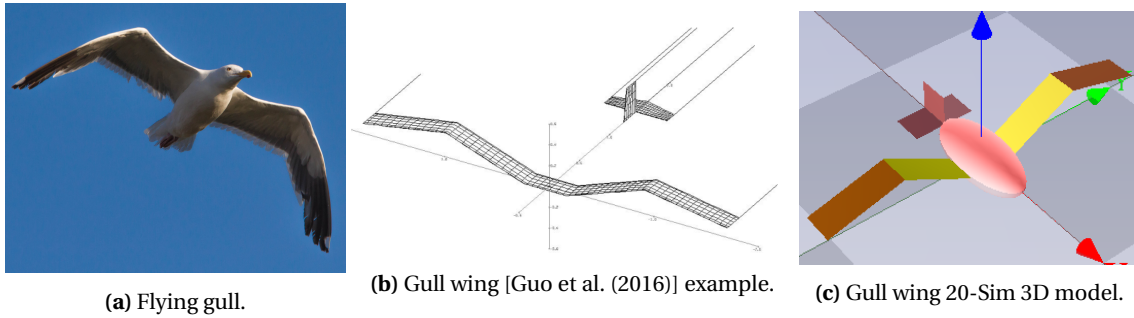


Figure 5.5: Gull wing models comparison, [Guo et al. (2016)] vs 20-Sim parent-children model.

Following with the modelling structure previously shown for the base model in 5.2, the new model shown in figure 5.6a has the same base structure, but several new elements are added.

For starters the wings have now two sections, referred as wing and subwing. The second are represented using leaf-*Child* subsystems, which does not have a "children" port, so doesn't require the zero-effort source connected at the end of them like the previous model did.

Now an horizontal tail and a ruther, which are both fixed to the main body, are included under the block "Tail", to simplify the outer view, and connected to the main body (internal view shown in figure 5.6b). The "Tail" block has two power ports since the tail and ruther are independent from each other and are both attached directly to the main body via **rigid joints**. As a *Child* is always expecting an actuators input, if we want a body to be attached rigidly to the precedent body it would be as simple as connecting a zero source of flow ($S_f=0$) to the "actuator" power port. The unit twist of this joint won't matter so it can be set to zero zero unit twist ($S_{tail}^{b,b} = [0; 0; 0; 0; 0; 0]$), in which case the actuator would have no effect anyway. Anyway, both tail elements actuators are zero-flow sources.

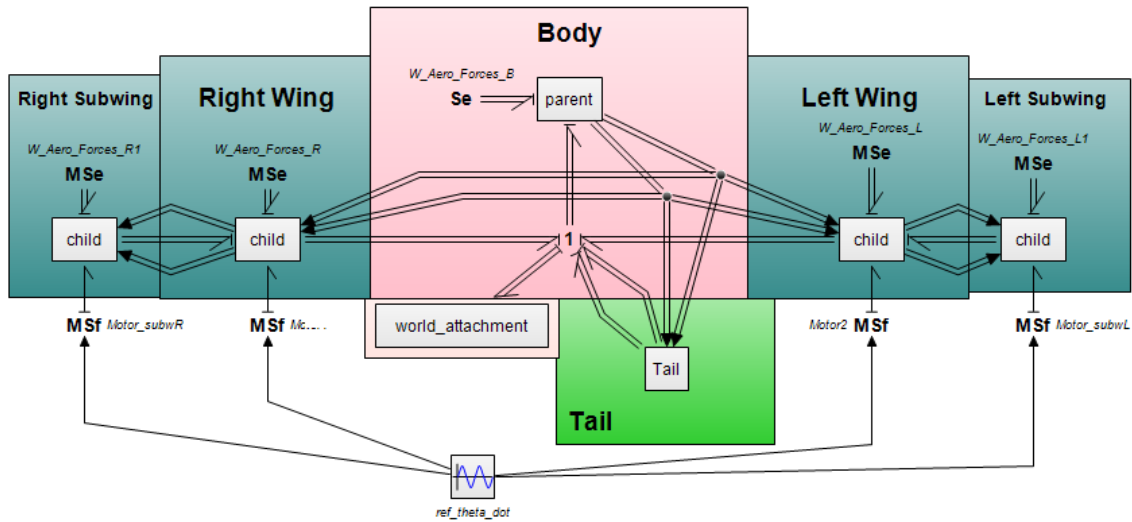
Another change is the employment of one single wave generator block to generate the four signals required for the four actuators that move the shoulder and elbow of the wings, this is, the joints body-wing and wing-subwing.

At last, a block called "world attachment" is connected to the main body through the central one-connection. This block, already introduced in section 5.1.1, help us simulate a test-bed, as it permits constraining any dof of the robot.

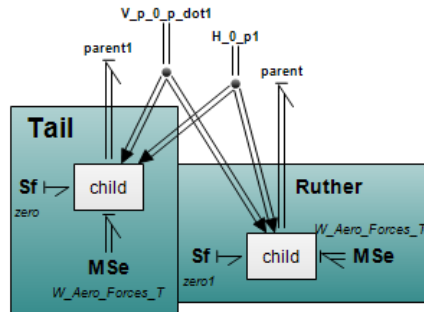
The physical parameters of the constructed model are shown in table 5.1.

Authors [Obradovic and Subbarao (2011)] demonstrate in their paper the use of gull wings for controlling the turning of the UAV. They explain the forces generated on the wings due to the aerodynamic effects that lead to the roll of the body (see figure 5.7). In plain words, the sum of vertical components of the force (in +Z of the body direction) when both elements of the wing are in parallel configuration (straight) is larger than the sum of vertical forces (in +Z of the body direction) on angled wings.

In order to imitate such behaviour an aerodynamic thrust is applied to each wing, slightly superior to the value of their weight, on the positive +Z direction of the wing's plane, this is, perpendicular to the wing planes and applied at the center of them. This thrust (wrench) is a fixed



(a) Gull Wing model outer view.



(b) Tail of the gull wing model inside.

Figure 5.6: Gull Wing model with a parent-child approach in 20-Sim.
 (File: "\Parent_Child_Modelling\Literature_Models\Gull_Wing\Gull_Wing_Turns.emx")

Parameters	Model
Body Mass	730 g
Body Length	70 cm
Body Radius	15 cm
Wings Mass	10 g
Wings Length	50 cm
Wings Chord	20 cm
Subwings Mass	10 g
Subwings Length	50 cm
Subwings Chord	20 cm
Tail Mass	8 g
Tail Length	40 cm
Tail Chord	20 cm
Ruther Mass	6 g
Ruther Length	20 cm
Ruther Width	20 cm

Table 5.1: Gull wing model physical parameters.
 (File: "\Parent_Child_Modelling\Literature_Models\Gull_Wing\Gull_Wing_Turns.emx")

value specified inside the sources of effort of each wing section (see figure 5.6), since there is no air implemented yet in the models, therefore no complex aerodynamic effects are used.

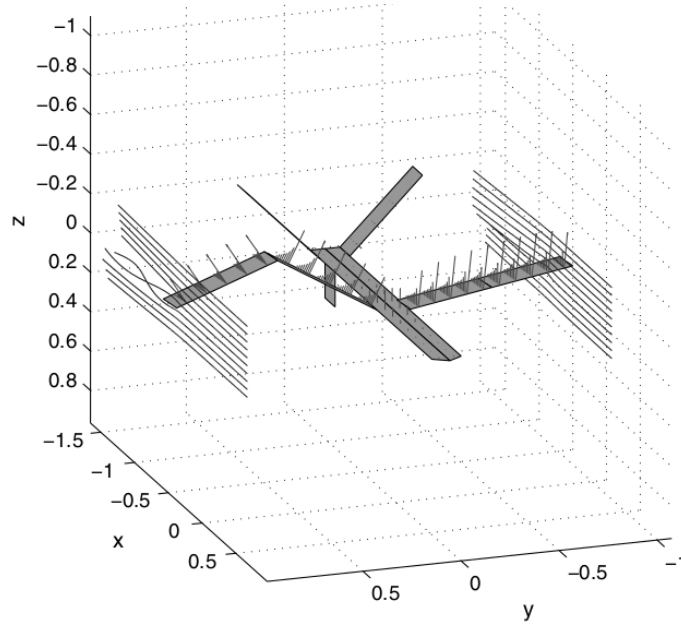
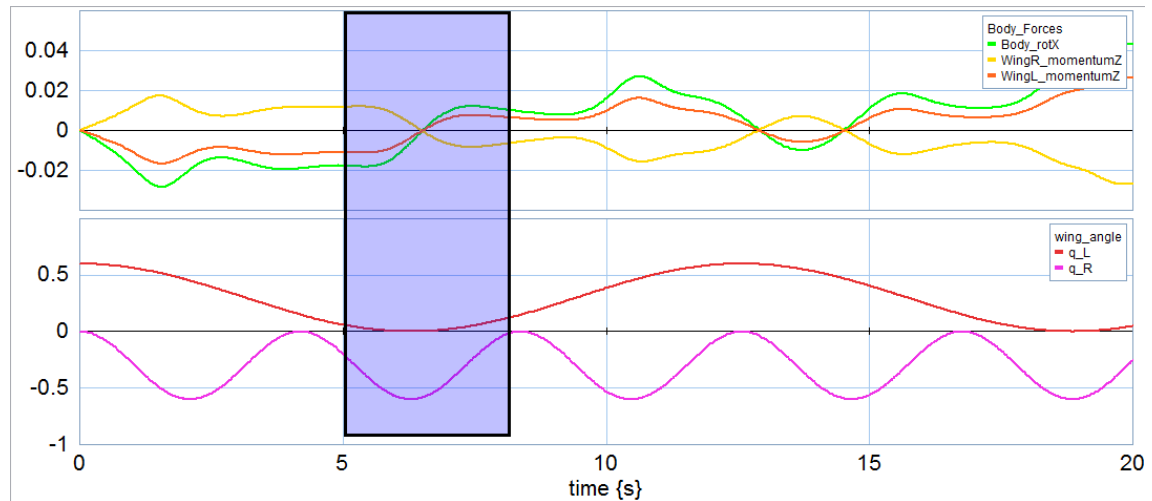


Figure 5.7: Forces on gull-wing surfaces with asymmetric configuration. [Obradovic and Subbarao (2011)].

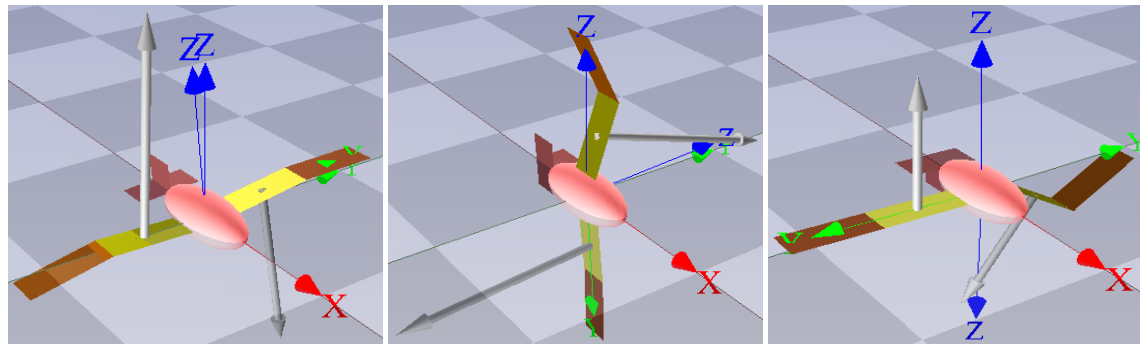
For testing the turning effect the UAV is coupled to the "world attachment" as already mentioned, that is set as a test bed that allows the bird to rotate around its X axis only (X axis is depicted in 3D simulation figures in 5.8). Furthermore, the right and left wings change their shape from "straight" to "gull" asymmetrically, in order to make the aerodynamic effects noticeable. The figures in 5.8 explain how the momentum changes while the wing is morphing. The vectors (gray arrows) represent the momentum in Z of the wing, not the force. The momentum as

$$P^i = G^i V_i^{i,0}$$

That is why both wing's momentum always point towards the same direction, on the direction of the rotation. The graph in figure 5.8a shows the position or angle of the wings, together with the rotation of the body (top graph, color green) and the momentum of each wing. The following images correspond with the lapse of time highlighted in purple in the graph. Originally, the bird is turning to the right (from the viewer's perspective), but during instants 1-4 the left wing is bending while the right one is straight, meaning that there is a larger thrust on the right wing. Therefore, the momentum decreases until instants 4-5 (figures 5.8e, 5.8f, when the momentum is finally inverted and the bird changes its rotation, proving the effectiveness of the gull morphing wing.



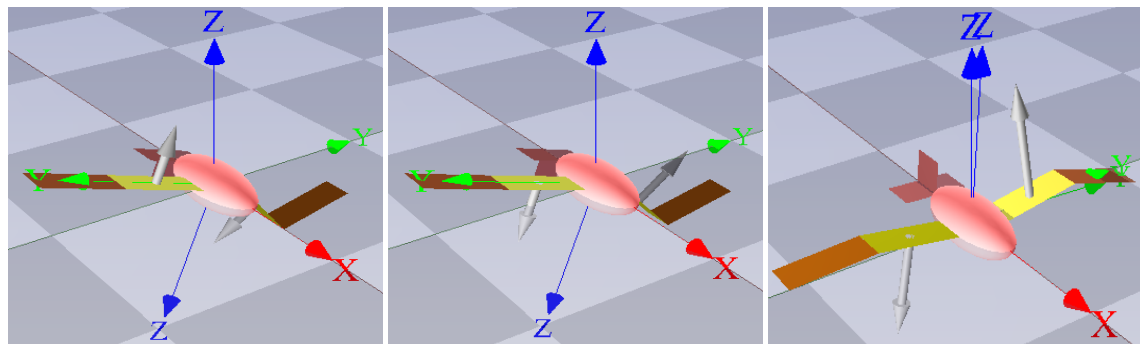
(a) Gull wing momentums and wing angles graph.



(b) Instant 1.

(c) Instant 2.

(d) Instant 3.



(e) Instant 4.

(f) Instant 5.

(g) Instant 6.

Figure 5.8: Gull wing model rotating using wing's change of shape. Frames a), b), c), d), e), and f) correspond with the lapse of time highlighted in the graph a). Originally the bird is turning right, until instant 5, when the rotation orientation is already changed.

(File: "\Parent_Child_Modelling\Literature_Models\Gull_Wing\Gull_Wing_Turns.emx").

5.4 Transformer aircraft (Flapping + prismatic wing)

[Ajaj and Jankee (2018)] present a model of an aircraft with prismatic wings, as shown in figure 5.9. Originally their model didn't modify its shape at high frequencies, but for the interest of this research an (not so similar) UAV will be modelled adding a flapping motion and incrementing the prismatic morphing frequency. This model is used for presenting the prismatic joint.

As presented during the framework chapter 4.4, joints can be prismatic as well. In this case, a part of the wing (subwing) is initially hidden inside the parent wing. When morphing happens the overall wing surface "expands". The model includes one rotational joint for the flapping

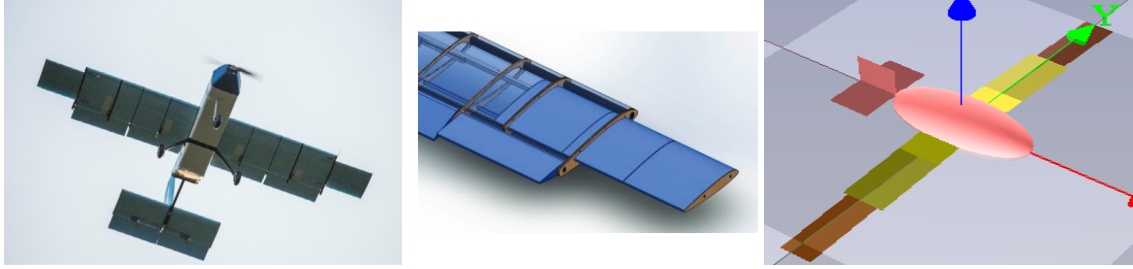


Figure 5.9: Prismatic wing aircraft, literature example compared to 20-Sim 3D model. [Ajaj and Jankee (2018)].

and one prismatic joint for the expansion/contraction of each wing. The prismatic joint child looks exactly the same as the rotational joint child, so the model look exactly the same as the gull-wing model in figure 5.6. An important difference though is that the "world attachment" is now set to constrain all dofs but the vertical movement along the Z axis.

The physical parameters of the model are shown in 5.2.

Parameters	Model
Body Mass	730 g
Body Length	70 cm
Body Radius	15 cm
Wings Mass	10 g
Wings Length	50 cm
Wings Chord	20 cm
Subwings Mass	8 g
Subwings Length	40 cm
Subwings Chord	15 cm
Tail Mass	8 g
Tail Length	40 cm
Tail Chord	20 cm
Ruther Mass	6 g
Ruther Length	20 cm
Ruther Width	20 cm

Table 5.2: Prismatic wing model physical parameters.

(File: "\Parent_Child_Modelling\Literature_Models\Bird_prismatic\Bird_prismatic.emx").

The motion of the model is represented in figure 5.10. The model flaps and expands/retracts its wings simultaneously. In the wing angle graph of figure 5.11 (middle graph) it is shown the motion of the wings. The prismatic movement frequency is double the frequency of the flapping.

When looking to the position graph in 5.11 one thing caught our attention over the rest. The Z position is slowly drifting downwards during the simulation. This are considered 20-Sim specific numerical errors. To check exactly the amount of error accumulated along time it is possible to see the exact values of the local minimums in the graphs. The error at the local minimum around 100s (exactly located at 100.55s) is of 0.0007m, this is, the relative error, taking that the amplitude of the bodies movement in Z is 0.0188m (extracted from graph), after a 100s of simulation would be of $0.0007m/0.0188m \times 100 \approx 3.72\%$. But if we consider the minimum Z position visible in the graph which is an accumulated 0.0035m at time 157.126s, it results in a relative error of $0.0035m/0.0188m \times 100 \approx 18\%$, which make us think the error is growing exponentially.

What's very likely is the position error to be very closely related to the energy balance progressive error appreciated for the model in graph 5.12. The maximum integration error is set in the simulation as $10^{-5} m/s$, but it is a possibility that the spring-damper element of the world attachment may be making a difference as well (further tested in appendix B.2). Nevertheless, there is no pattern found in the error drift, so it will be considered numerical error.

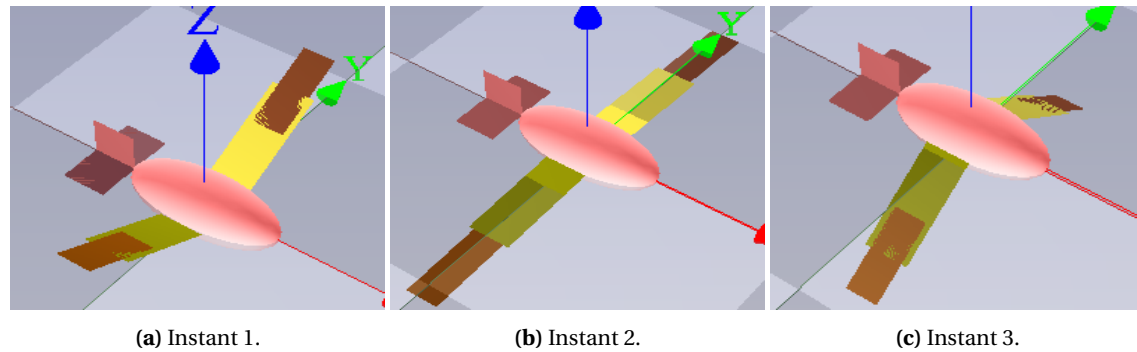


Figure 5.10: Flapping wing with prismatic subwing.

(File: "\Parent_Child_Modelling\Literature_Models\Bird_prismatic\Bird_prismatic.emx").

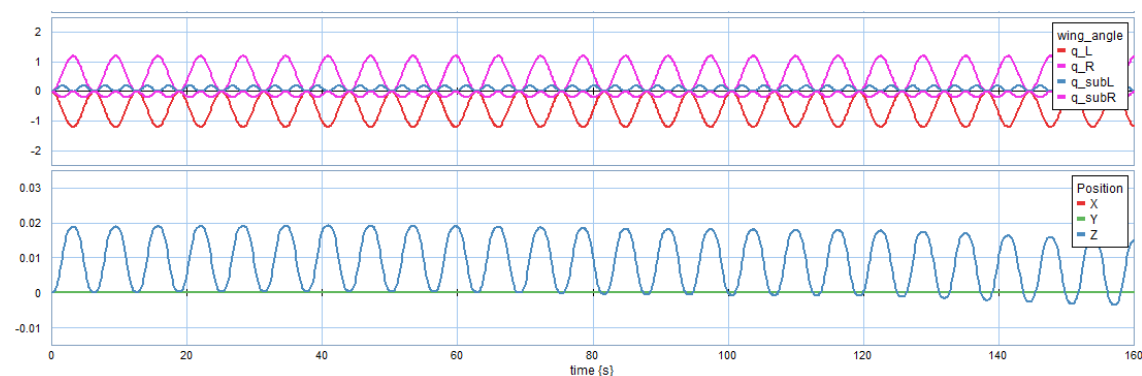


Figure 5.11: Flapping wing with prismatic subwing motion (top graph) and position (bottom graph) slowly decreasing height due to numerical error.

(File: "\Parent_Child_Modelling\Literature_Models\Bird_prismatic\Bird_prismatic.emx").

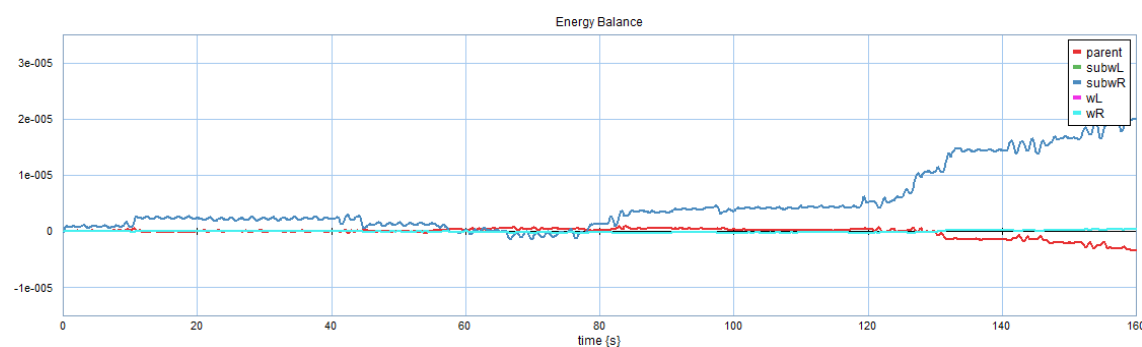


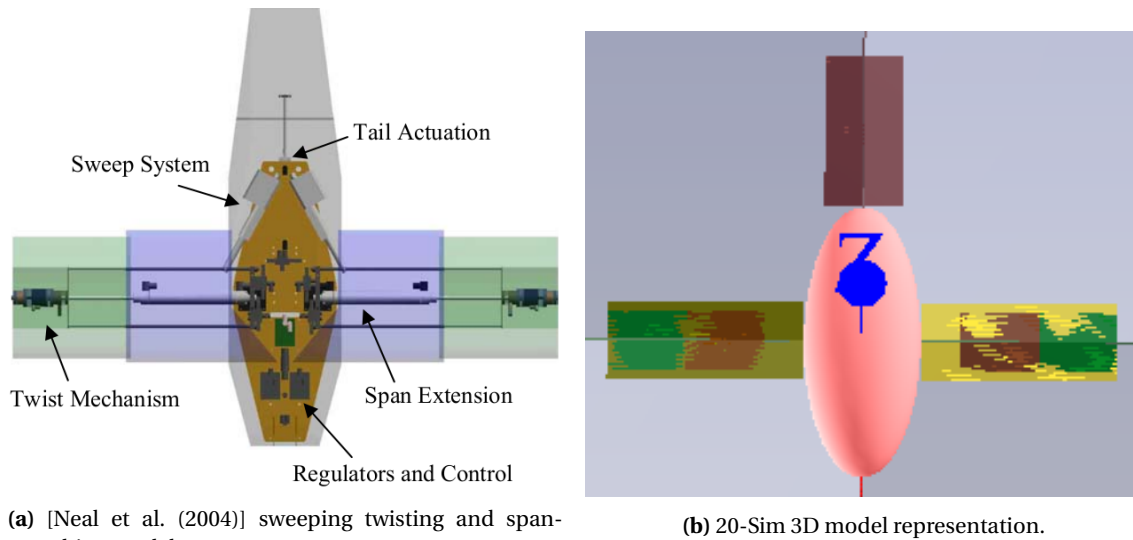
Figure 5.12: Flapping wing with prismatic subwing energy balance graph.

(File: "\Parent_Child_Modelling\Literature_Models\Bird_prismatic\Bird_prismatic.emx").

5.5 Fully Adaptive UAV (Sweeping + Twisting + Span-morphing wing)

[Neal et al. (2004)] present a model of an aircraft with sweeping and prismatic wings that depending on the phase of the flight it acquires a configuration or another. The model from [Neal et al. (2004)] will be imitated rather accurately, even though they don't modify its shape at high frequencies, but for our interest we will increase them.

The fully adaptive UAV present three wing motions altogether in their design, the wing sweeping, the span extension and the wing twisting. In this case the UAV won't flap its wings. The shape of the UAV is presented in figure 5.13, where the three different parts of the wing are well differentiated. If we look to the 20-Sim 3D model in figure 5.13b, in yellow is the main wing that sweeps with respect to the main body (red). Second, in orange (+green), is the span extendable sub-wing. And finally, in green is the twisting section of the sub-wing. These morphing elements of each wing of the UAV are modeled as three bodies with simple joints (one dof each), two rotational, the sweep and the twist, and one prismatic, corresponding to the span-morphing.



(a) [Neal et al. (2004)] sweeping twisting and span-morphing model.

(b) 20-Sim 3D model representation.

Figure 5.13: Fully Adaptive UAV (Sweeping, twisting and span-morphing) literature example (left) vs 3D model in 20-Sim (right). [Neal et al. (2004)].

The constructed 20-Sim parent-child model is presented in figure 5.14, where the three elements of the wings are distinguished as 'wing' (sweep), 'subwing' (span) and 'subsubwing' (twist). The tail is fixed to the body by forcing a zero-flow ($Sf=0$) through its actuator input. The "world attachment" is set to constrain all rotational dofs of the robot but the displacements along the X, Y, Z axis are free. The only displacement there will be though is in the X axis, forward and backward due to the inertial forces generated by the synchronous wing sweeping mainly.

The physical parameters used for this model are displayed in table 5.3. At last, the motion is demonstrated in figure 5.15 and 5.16, where again a progressive drift in the position (lower graph) is visible. The X displacement curve is the sum of the sweeping and the prismatic motions, which are presented in the top graph of 5.16 as $q_{R/L}$ and $q_{subR/L}$ respectively, that generate the inertial forces in the X axis.

Originally, the authors in [Neal et al. (2004)] need to explicitly take care of both, the center of gravity displacement and the aerodynamics center displacement and account for this displacement at each time-step, which is very inefficient. These centers position vary with the morphing of the wings, as shown in figure 5.17. Nevertheless, when the aircraft is modelled with

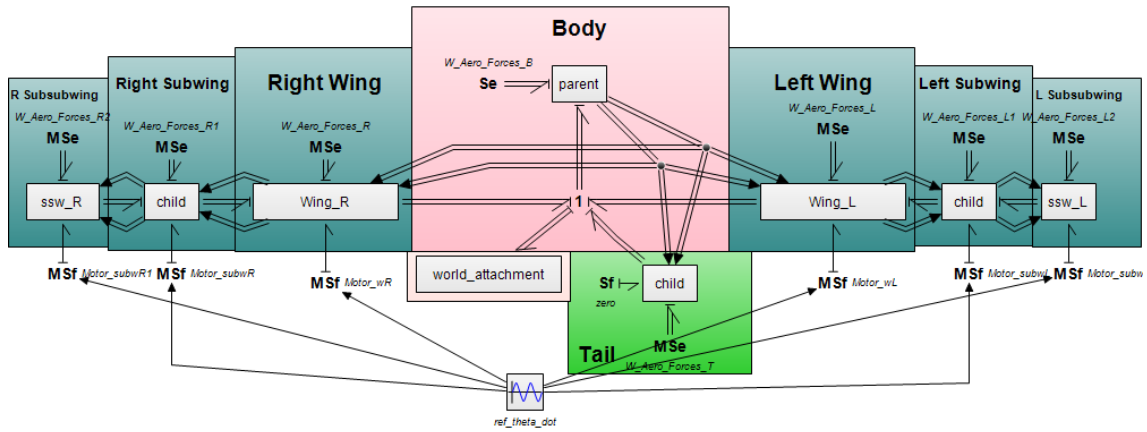


Figure 5.14: Fully Adaptive UAV model with a parent-child approach in 20-Sim. (File: "\Parent_Child_Modelling\Literature_Models\Fully Adaptive UAV\UAV_fully_adaptive.emx").

Parameters	Model
Body Mass	730 g
Body Length	70 cm
Body Radius	15 cm
Wings Mass	10 g
Wings Length	50 cm
Wings Chord	20 cm
Subwings Mass	4 g
Subwings Length	20 cm
Subwings Chord	15 cm
Subwings Mass	4 g
Subwings Length	30 cm
Subwings Chord	15 cm
Tail Mass	8 g
Tail Length	20 cm
Tail Chord	40 cm

Table 5.3: Fully Adaptive UAV model physical parameters. (File: "\Parent_Child_Modelling\Literature_Models\Fully Adaptive UAV\UAV_fully_adaptive.emx").

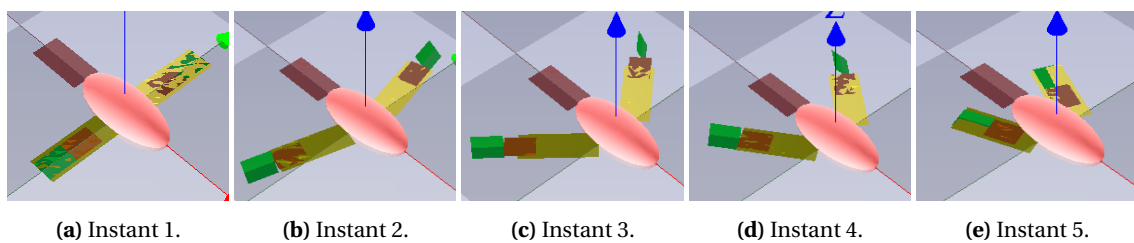


Figure 5.15: Fully Adaptive UAV 20-Sim 3D model motion. (File: "\Parent_Child_Modelling\Literature_Models\Fully Adaptive UAV\UAV_fully_adaptive.emx").

the parent-child construction such displacements are automatically accounted by the bodies, without the need of introducing any extra equations into them, greatly reducing the computational effort and facilitating the design at the same time.

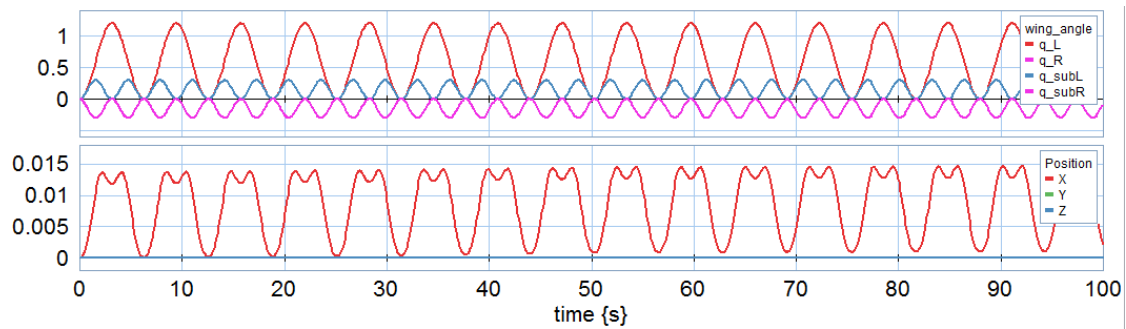


Figure 5.16: Position of the Fully Adaptive UAV along time (bottom graph) with a sweeping twisting and span synchronous motion (top graph).

(File: "\Parent_Child_Modelling\Literature_Models\Fully Adaptive UAV\UAV_fully_adaptive.emx").

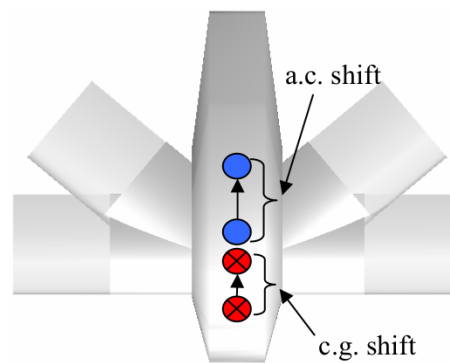


Figure 5.17: Center of gravity and aerodynamic center displacement with wing morphing. [Neal et al. (2004)].

5.6 Sweeping-Wing UAV (Flapping + Sweeping)

The gull wing in section 5.3 has proven to be effective so far for turning. Nonetheless, other researchers, like the authors of [Grant et al. (2006)], have studied the possibility of controlling the flapping-UAV by means of sweeping several wing sections forth and back, both symmetrically and asymmetrically, as shown in figure 5.19. In this case, differently from the gull wing, the aerodynamics become a lot more complex, since they are dependent on the orientation of the wing with respect to the air flow, so that won't be simulated, but we can show the capabilities of the 20-Sim parent-children model, shown in figure 5.20, to change its sweep while flapping. The original UAV is blade propelled, but it will be modified to make it a flapping-wing UAV in the seek of our interest, as it would be the natural step forward from the researches to take.

But before diving into modelling the flapping wing with multiple sweeping sections UAV, a slightly more simple model with only one wing section that can sweep and flap simultaneously should be built, since this requires the introduction of an universal joint (multi-dof joint).

Base model with a multi-dof joint

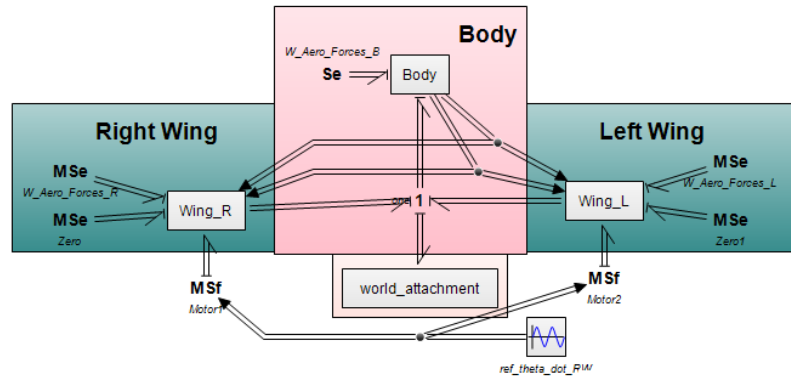
The most basic flapping model with an universal joint would be a bird which wings have two dofs, for instance flapping plus sweeping wings, which can reproduce very varied trajectories. For example, the trajectory could be an infinite-shape or similar to a circle. This basic 20-sim parent-child model is presented in figure 5.18a. It includes the main body and two wings, but both wings have multi-dof joints, so they can be actuated in all directions simultaneously. As visible in the parent-child model, the main difference with previous models is in the actuator (also explained during section 4.4 and extended in appendix A.2.2), which becomes an array

input that can actuate all six dofs instead of only one. For generating an infinite shape path with the tip of the wings, these need to rotate around both the X and Z axis, so those would be the only actuated inputs and the rest will be set to 0, symbolically constraining those dofs. The frequency of rotation around Z, ω_z , will be double the frequency of rotation around X, ω_x , $\omega_z = 2 * \omega_x$. If the desired path was a circle the frequencies ω_z and ω_x should be equal, $\omega_z = \omega_x$. The infinite path flapping is simulated in figures 5.18a-5.18g.

The physical parameters of such basic model are enumerated in table 5.4.

Parameters	Model
Body Mass	730 g
Body Radius	20 cm
Wings Mass	100 g
Wings Length	100 cm

Table 5.4: Basic multi-dof joint model physical parameters.



(a) Basic model for the experimentation of sweeping+flapping. Use of universal joints.

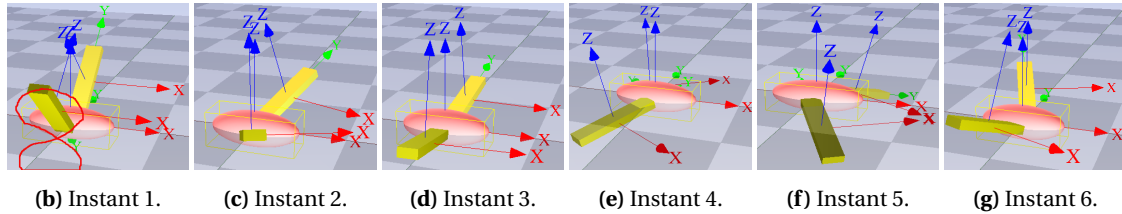


Figure 5.18: Flapping + Sweeping for infinite-shape path example using a universal joint.

(File: "\\Parent_Child_Modelling\Literature_Models\Bird_sweep&perch\Sweep_infinite.emx").

Once the multi-dof joint has been introduced it becomes quite easy to move forward and model the sweeping multi-wing UAV presented by [Grant et al. (2006)] (see figure 5.19) using as starting point the previously modeled gull wing in section 5.3. The construction is the same with the exception of the joints between body and wings, that this time they are universal type of joints, this is, the *child-parent* elements representing the main wings sections need to have several dof instead of one. In terms of the wing parameters the only difference is that, while with the simple joint the specification of the rotation axis is done with the unit twist, so we would need two for the two dofs, with the multi-dof joint these two unit twist parameters are replaced with one single array representing the relative position of the joint wrt the parent instead, as explained in section 4.4.

$$\text{From: } S_c^{p,p} = \begin{pmatrix} \hat{\omega} \\ \hat{\omega} \wedge \hat{v} \end{pmatrix} \rightarrow \text{To: } r_{joint}^p = \begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix}$$

This is also illustrated along section A.4.3.1. Besides the aforementioned changes, the "actuator" inputs are now 6x1 arrays, even though we use only rotation around X, for flapping, and around Z, for sweeping. Thus the wave generator will generate those two signals.

On the other hand, the children representing the subsections of each wing can only sweep, so they can use simple joints configured via the unit twist to rotate around Z exclusively.

The resulting model looks as in figure 5.20. In this occasion the "world attachment" is constraining all six dofs of the UAV, since the eye is put on the morphing and the use of the multi-dof joint only. Figure 5.21 shows a 3D vision of the sweeping wings morphing while flapping. Model's physical parameters are shown in table 5.5.

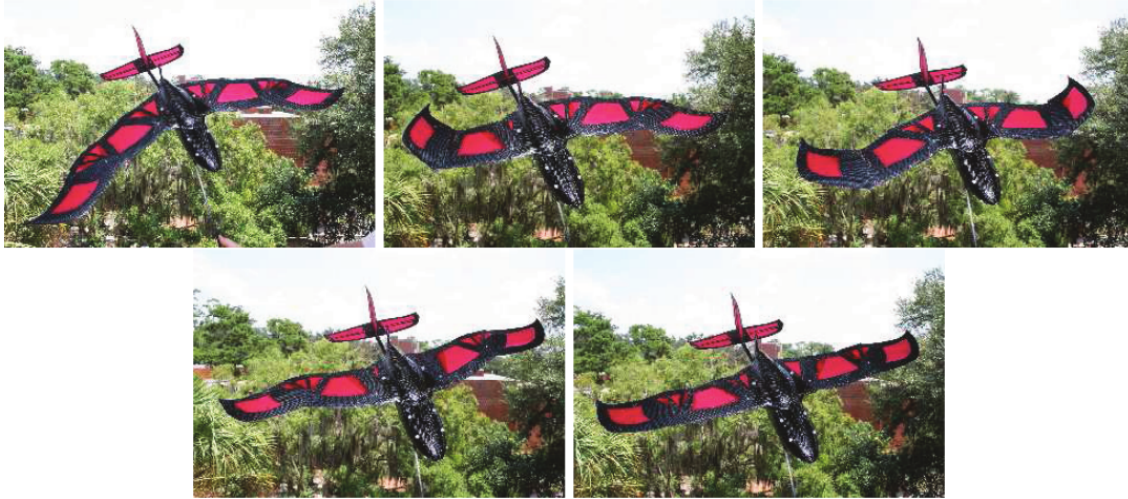


Figure 5.19: Literature example of morphing-wing UAV that can sweep its wings into many different configurations. [Grant et al. (2006)]

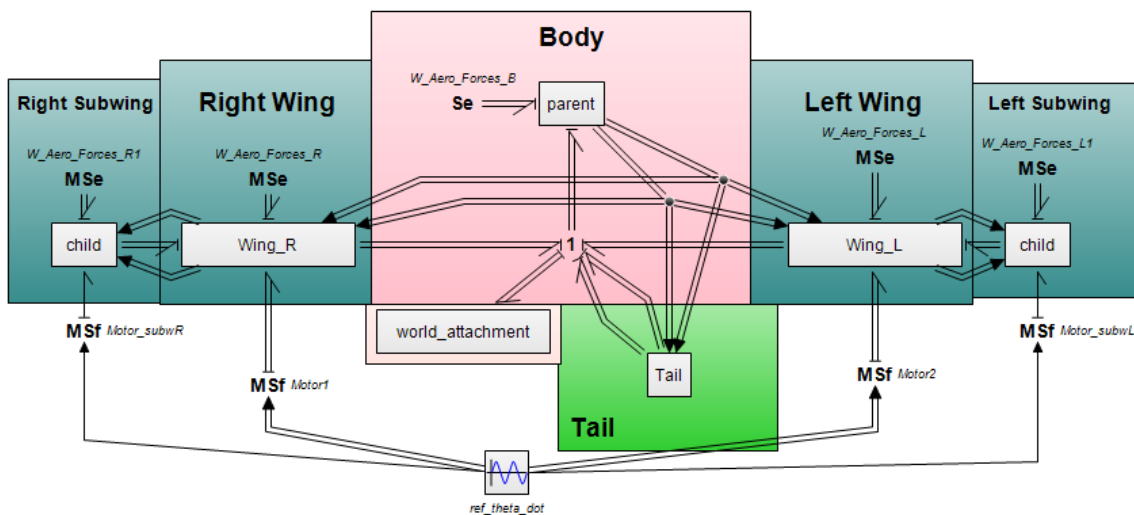


Figure 5.20: Flapping + Sweeping and/or twisting parent-children model in 20-Sim.
 (File: "\\Parent_Child_Modelling\Literature_Models\Bird_sweep&perch\Bird_Sweep.emx").
 (File: "\\Parent_Child_Modelling\Literature_Models\Bird_sweep&perch\Bird_Perch.emx").

Even though we cannot test the effect of sweeping the wings for rolling (rotate around X), we can test the inertial effects that such morphing provokes for rotating around Z (yaw). For this, we can set the "world attachment" to free the rotations around Z together with the linear displacements.

Parameters	Model
Body Mass	730 g
Body Length	70 cm
Body Radius	15 cm
Wings Mass	10 g
Wings Length	50 cm
Wings Chord	20 cm
Subwings Mass	10 g
Subwings Length	50 cm
Subwings Chord	20 cm
Tail Mass	8 g
Tail Length	40 cm
Tail Chord	20 cm
Ruther Mass	6 g
Ruther Length	20 cm
Ruther Width	20 cm

Table 5.5: Sweeping wing multi-configurations and twisting wing model parameters.

(File: "\Parent_Child_Modelling\Literature_Models\Bird_sweep&perch\Bird_Sweep.emx").

(File: "\Parent_Child_Modelling\Literature_Models\Bird_sweep&perch\Bird_Perch.emx").

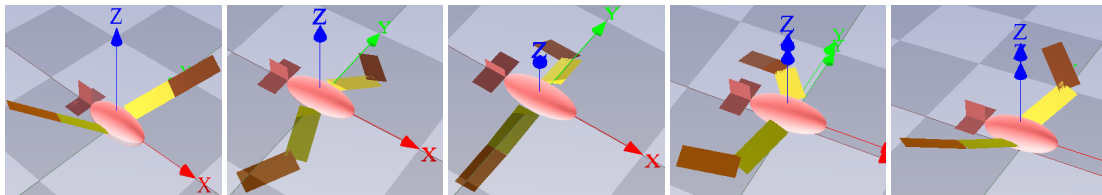


Figure 5.21: Flapping + sweeping wing and sweeping subwing motion.

(File: "\Parent_Child_Modelling\Literature_Models\Bird_sweep&perch\Bird_Sweep.emx").

The inertia momentums generated by the sweeping motion effectively make the UAV yaw as demonstrated in figure 5.22, where in the lower graph it is shown the orientation of the body, that is constantly rotating in Z, as indicated by the blue line, thus increasing (in absolute terms) its angle. The very asynchronous wing motion is shown in the top graph of the same figure. In the middle graph appears the position of the body along time.

5.7 Perching UAV (Flapping + Twisting)

Continuing with the multi-dof joint, it might be used as well for modelling a bird that executes a perching or landing maneuver, where instead of sweeping the wing the required movement is a twist of the wings on top of the flapping. The 20-Sim model and parameters for this case are exactly the same as for the previous sweeping UAV, only instead of sweeping the wings the motion requires twisting them, this is, instead of rotating around Z, the rotation of the wings is around Y. In both cases there is still a movement around X, the flapping. The only change required between models is a swap of inputs into to the "actuator" power port, thanks to the use of the multi-dof joint, otherwise we would need to change the unit twist definition, which would be more tedious. The rest is exactly the same for the sweeping-wing example and the perching example. This movement is studied by [ji et al. (2020)]. Note that in figure 5.23 the Z and Y axis are configured differently (rotated) compared to the the 20-Sim model frames in figure 5.24. The twisting rotation axis is parallel to the wingspan axis.

In the case of the perching UAV the "world attachment" is set to constrain the rotational dofs only, leaving the linear displacements free. Due or thanks to the inertia momentums generated

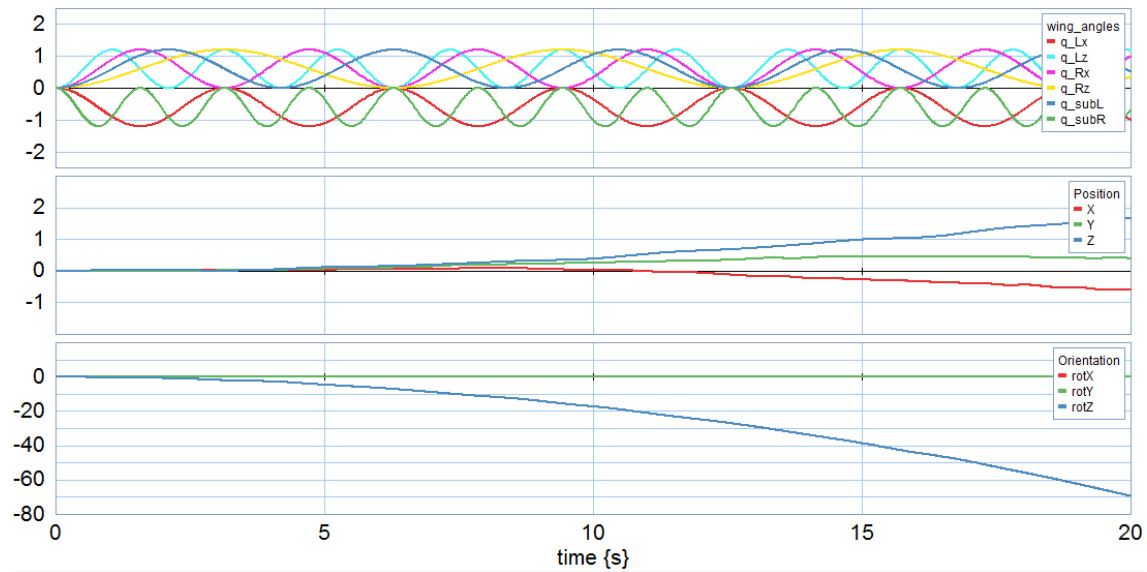


Figure 5.22: Sweeping wings (Flapping + sweeping) UAV motion (top) and body position (mid) and orientation (bottom) simulated in 20-Sim.

(File: "\Parent_Child_Modelling\Literature_Models\Bird_sweep&perch\Bird_Sweep.emx").

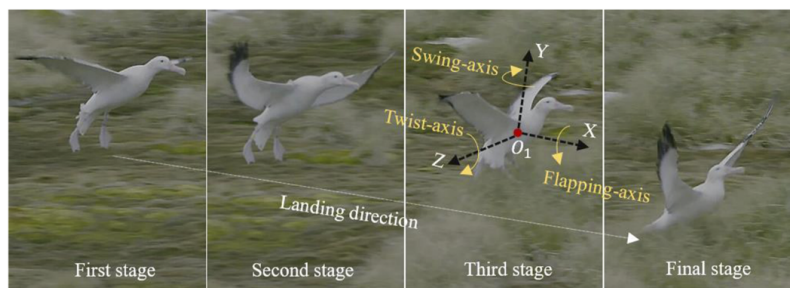


Figure 5.23: Example of seagull executing a perching maneuver. [ji et al. (2020)].

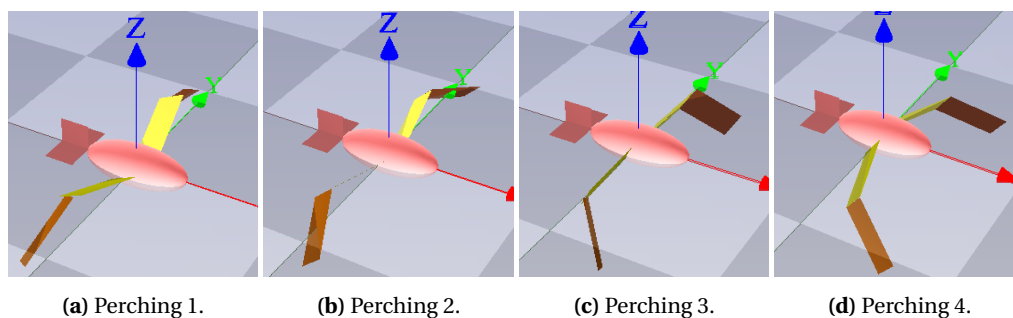


Figure 5.24: Flapping + twisting for a perching (landing) maneuver.

(File: "\Parent_Child_Modelling\Literature_Models\Bird_sweep&perch\Bird_Perch.emx").

by the motion of the wings, that move up and down while forward and backwards simultaneously (thanks to the multi-dof joint) and the sub-wings that sweep back and forth, the UAV suffers a displacement in both X and Z axis, as shown in figure 5.25.

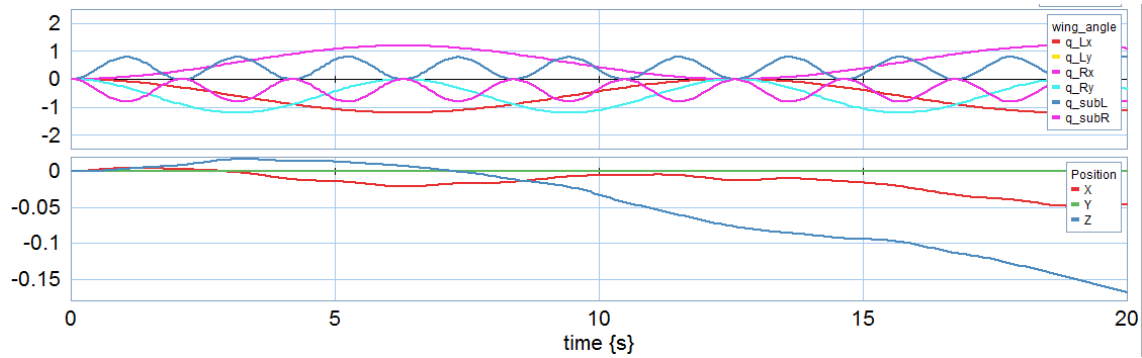


Figure 5.25: Perching (Flapping + twisting) UAV wings motion (top) and body position (bottom) simulated in 20-Sim.

(File: "\Parent_Child_Modelling\Literature_Models\Bird_sweep&perch\Bird_Perch.emx").

5.8 LisHawk (Flapping + Sweeping wing + Tail Morphing)

The next model from [Ajanic et al. (2020)], figure 5.26, is selected for its explicit universal joint, to emphasize the necessity of the multi-dof joint subsystem. The UAV presented in [Ajanic et al. (2020)] is driven by a blade propeller, nonetheless in the 20-Sim parent-children model the propeller is removed and the UAV is propelled by flapping wings instead.

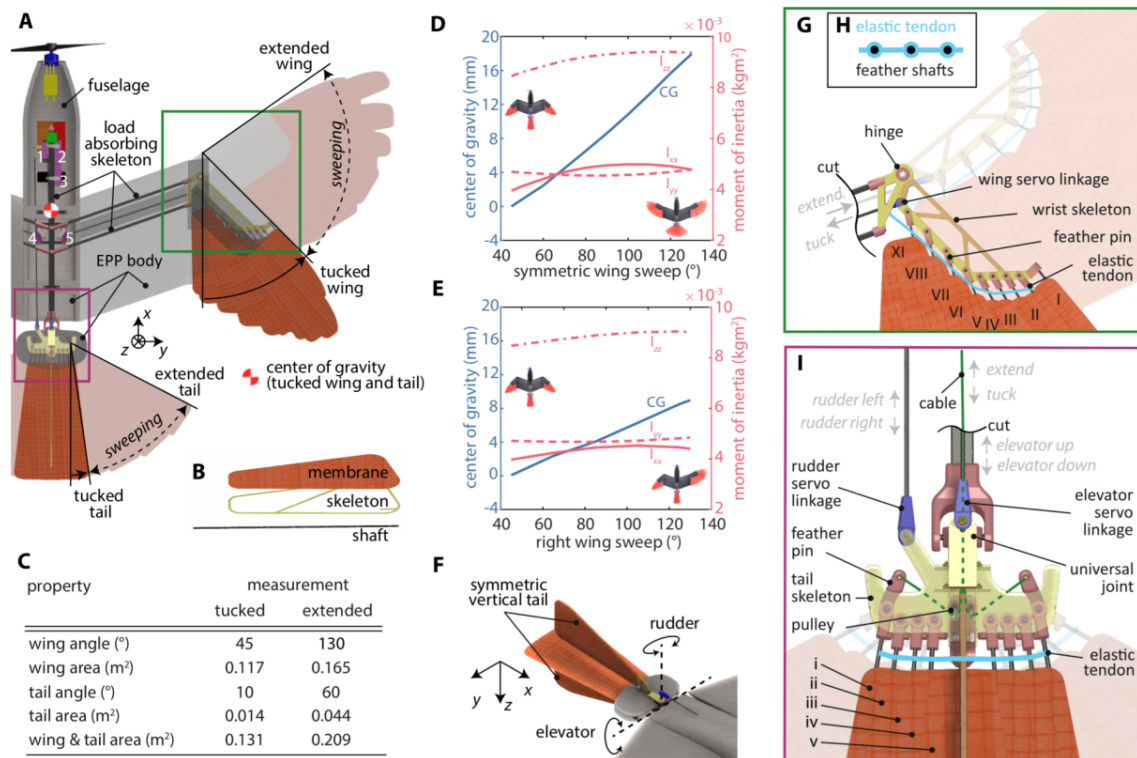


Figure 5.26: Literature example of a (non-flapping) bird with an actuated elbow on the wings and universal tail joint. [Ajanic et al. (2020)].

The tail is the most interesting part of this UAV, since it is explicitly actuated by an universal joint, as depicted in figure 5.26. The tail can be modeled in different ways, but it will be treated as if it had two horizontal plates/elements that can rotate up-down and to the sides, imitating an universal joint (a *multi-dof joint* is used) plus a sweeping mechanism. In the paper, the tail appears to extend and retract, this is, the tail surface expands, so the model will be built in such a way that when tucked the two tail elements are located one on top of each other,

so the "opening" of the tail wing is simulated as a separation of these two elements, that are always the same width individually. At the same time, the wings are divided in two sections with two simple joints in shoulder and elbow. Figure 5.27a shows the whole 20-Sim model, where the instances "Wing L", "Wing R" and "Tail" are subsystems or abstractions composed by, in the case of the wings, two wing sections (see figure 5.27b), and, in the case of the tail, also two elements as previously mentioned, indicated as right ("Tail R") and left ("Tail L"), both actuated by *multi-dof joints* (see figure 5.27c). The distance vector $r_{tail_joint}^{parent}$ from the main body to the tail is depicted in figure 5.28. This vector is required by both *multi-dof joints*.

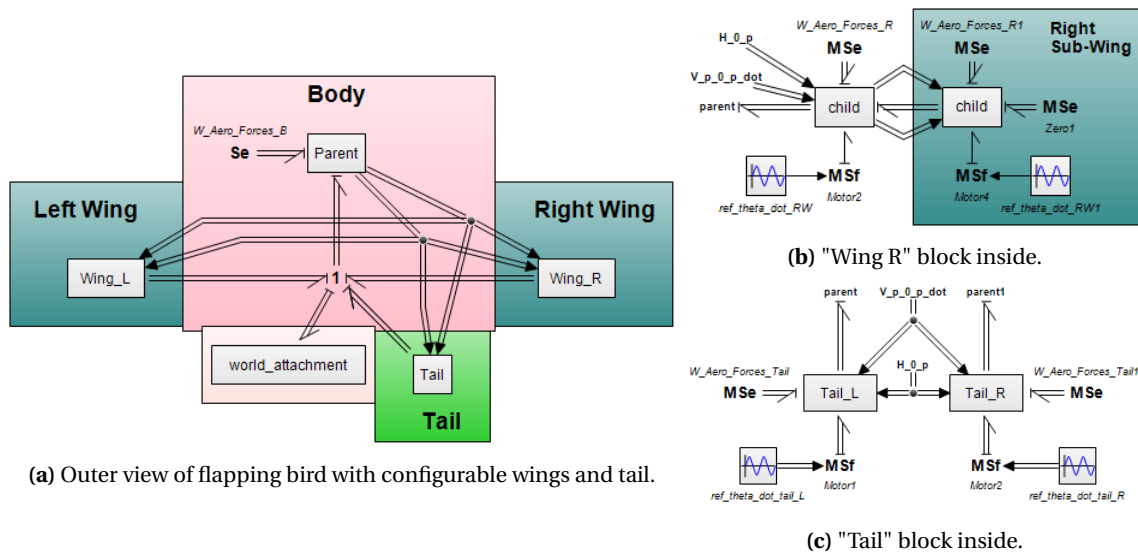


Figure 5.27: Literature example of a (originally non-flapping) bird with an actuated elbow on the wings and universal tail joint modeled on 20-Sim with the parent-children philosophy. (File: "\Parent_Child_Modelling\Literature_Models\LisHawk\LisHawk.emx").

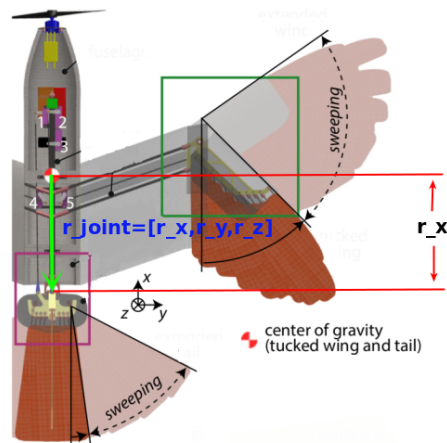


Figure 5.28: Illustration of r_{joint}^p vector, in green, of universal tail joint (*multi-dof joint*). Modified from [Ajanic et al. (2020)].

The physical parameters of the model are presented in table 5.6. Additionally, the motion of the bird is depicted in figure 5.29, where frames of the 3D simulation are displayed.

The *multi-dof joint* demonstrate its usefulness. It is in fact very simple to implement, easy and effective to use. All the complexity one can find would be in the generation of the movement path, but that would be also the case with simple *joints*. It should cost more computational effort, but it didn't mean a noticeable difference so far. A simulation of 20 seconds takes way less than a second up to this point of simulating models with the parent-child methodology. Whats

Parameters	Model
Body Mass	156.2 g
Body Length	80 cm
Body Radius	20 cm
Wings Mass	31.12 g
Wings Length	52.5 cm
Wings Chord	32 cm
Subwings Mass	20 g
Subwings Length	52.5 cm
Subwings Chord	32 cm
Tails Mass	12.78 g
Tails Length	24 cm
Tails Chord	16.1 cm

Table 5.6: Morphing wing and tail model parameters.
(File: "\Parent_Child_Modelling\Literature_Models\LisHawk\LisHawk.emx").

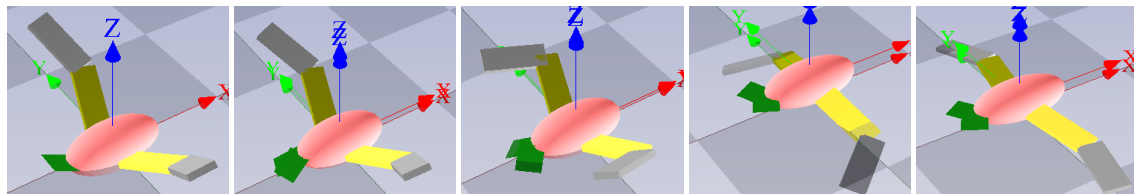


Figure 5.29: Flapping while wing sweeping and tail with multi-DOF joints morphing.
(File: "\Parent_Child_Modelling\Literature_Models\LisHawk\LisHawk.emx").

more, the method, as already mentioned before, is energy consistent for long simulations, as demonstrated again for this model in figure 5.30, where the energy balance is always around 0, with some numerical error, but not drifting.

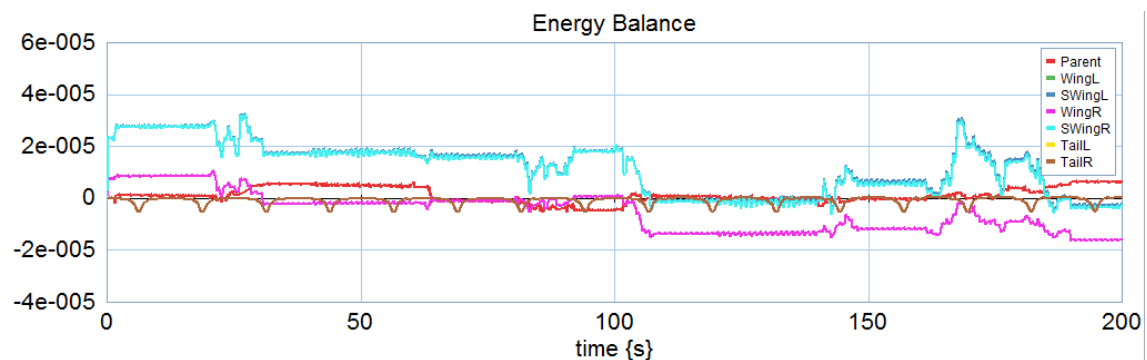


Figure 5.30: Energy consistency of the LisHawk model with two multi-dof joints for the tail components.
(File: "\Parent_Child_Modelling\Literature_Models\LisHawk\LisHawk.emx").

Additionally, in figure 5.31 is demonstrated the momentum generated by each section of the wing during the flapping motion while the sub-wing is also sweeping. The wing momentum in Z (top graph) is a constant sine, while sub-wing Z momentum (bottom graph) varies accordingly to the distance of its CG to the main body, as when it is tucked the CG is closer to the main body, thus generates less momentum.

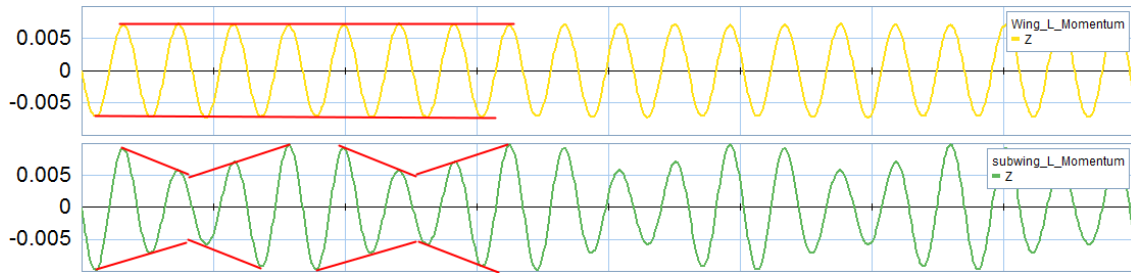
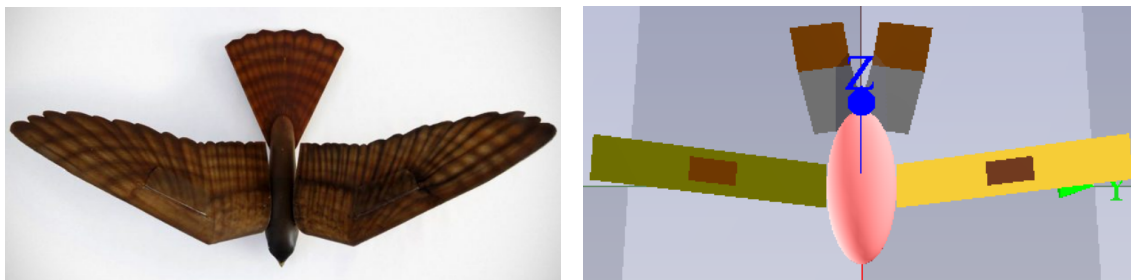


Figure 5.31: Wing and sub-wing momentums comparison with sub-wing morphing from tucked to extended. Wing momentum in Z (top) remains the same, while sub-wing Z momentum (bottom) varies with its CG distance to the main body.

(File: "\Parent_Child_Modelling\Literature_Models\LisHawk\LisHawk.emx").

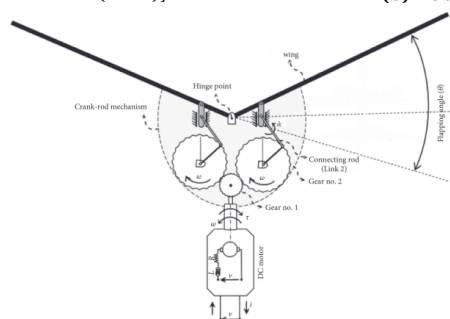
5.9 Robird

The Robird ([Folkertsma et al. (2017)]) is, mechanically, a rather simple robotic bird, since both wings can only flap simultaneously, being the turning generated by flaps located on both wings and more importantly on the tale, in a similar manner to a plane. The base model, exposed in figure 5.33, is a simplification of the Robird shown in figure 5.32a. Both wings move synchronously, actuated by a double **crank-rod mechanism** similar to the one shown in figure 5.32c, moved by an electric **DC motor** fed by a **regulated voltage source**. 20-Sim provide out-of-the-shelf elements that can be attached to our models thanks to the power port characteristics, that allow the interconnection of multi-domain subsystems, together with the ease of use of our modelling method. Building a model with these elements is of course more challenging, but not due to the Child-Parent structure, but because of the complexity of the mechanical, hydraulic or electrical sub-models themselves. Therefore an out-of-the-shelf crank-rod mechanism a DC motor and a regulated voltage source are used for constructing the Robird model (see figure 5.33).



(a) Robird top view. [Folkertsma et al. (2017)]

(b) Robird 20-Sim 3D model.



(c) Diagram of flapping bird with crank-rod mechanism actuator.

Figure 5.32: Robird literature model, 20-Sim 3D model and crank-rod actuator of the synchronous flapping.

The parent-child model 3D view is shown in figure 5.32b. The design, described in blocks in figure 5.33, includes flaps on each wing and two horizontal (inclined 15° around x) tails with one active ruther or flap at the end of each tail. The tail bases are static components directly attached to the main body. Considering those parts different bodies from the main body would ease the expression of the aerodynamic forces over this area later on, but they could also be considered as part of the body. Similarly, due to the change of shape along the wing in the real model, the wing could be divided in as many parts as desired, fixed one to the other to provide it a more accurate shape, even though it is probably unnecessary. This could even depend on the aerodynamic approximation used, because if for instance a strips theory was selected, like in [Abdelbadie (2021)], the wing could be conveniently modeled as a large amount of consecutive strips all fixed one after the other. The methodology is agile enough to consider using many elements to define the bodies.

All four flaps and ruthers are actuated by wave generators and the tails are kept static attached to the main body thanks to the "action" of a zero-flow source ($S_f=0$) and a zero unit twist ($S_{tail}^{b,b} = [0; 0; 0; 0; 0; 0]$).

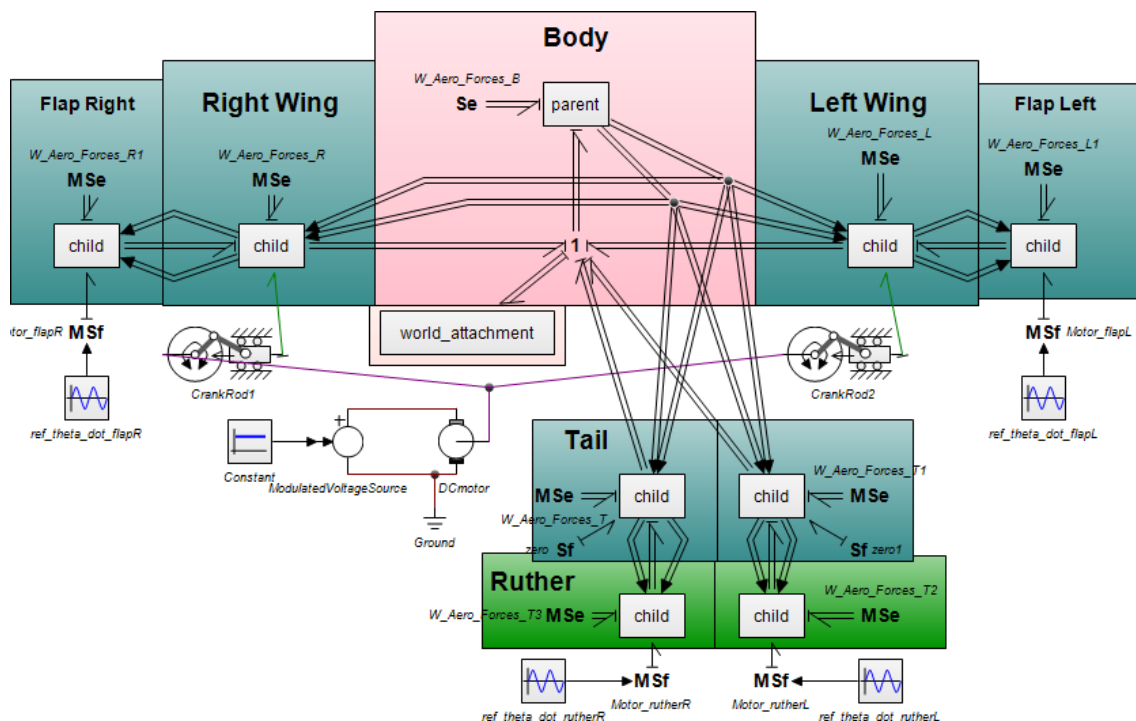


Figure 5.33: Robird model with a parent-child approximation in 20-Sim. (File: "\\Parent_Child_Modelling\Literature_Models\Robird\Robird.emx").

In figure 5.34 it is demonstrated the movement of the wings and flaps simultaneously. The physical parameters of the model are presented in table 5.7. Additionally, figure 5.35 intend to show the inclination of the tail base, that are internally rotated 15° , as already mentioned.

An experiment was carried where the wings and flaps of the Robird are kept still while the tail ruthers (in orange in the 3D models) are rotated alternatively. Additionally, the "world attachment" was set to merely constrain the rotations around Y (pitch), while all other five dofs remain free. The experiment is shown graphically in figure 5.36. The results of such movement is, first of all, a rotation around the X axis (roll) due to the inertial momentum generated by the motion of the ruthers (see red line in middle graph), which comes hand by hand with a rotation around the Z axis due to the inertial momentum generated by the ruthers in the X direction. At last, this combination of rotations generate as well a continuous lateral displacement along the Y axis, as it is clear from the green line in the bottom graph.

Parameters	Model
Body Mass	730 g
Body Length	70 cm
Body Radius	15 cm
Wings Mass	18g
Wings Length	100 cm
Wings Chord	20 cm
Flaps Mass	4 g
Flaps Length	20 cm
Flaps Width	10 cm
Tails Mass	6 g
Tails Length	30 cm
Tails Chord	25 cm
Ruthers Mass	4 g
Ruthers Length	20 cm
Ruthers Width	25 cm

Table 5.7: Robird model parameters.

(File: "\Parent_Child_Modelling\Literature_Models\Robird\Robird.emx").

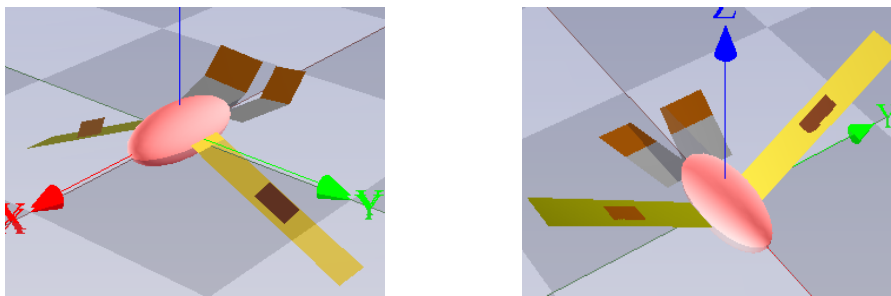


Figure 5.34: Robird parent-child model flapping 3D simulation.

(File: "\Parent_Child_Modelling\Literature_Models\Robird\Robird.emx").

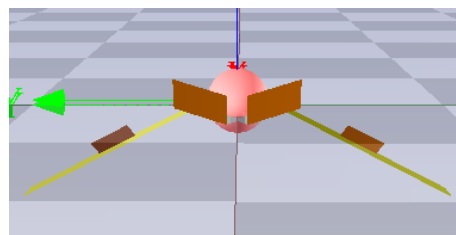


Figure 5.35: Robird parent-child model 3D from the back.

(File: "\Parent_Child_Modelling\Literature_Models\Robird\Robird.emx").

Finally, a reminder that what the top graph is showing in 5.36 "wing angles" graph is not the absolute wing or ruther angle, but the variation of the angle from its original configuration (relative angle). This is, the ruther angles are in this case shifted ± 0.6 rad since the left ruther is starting at 0.6 rad and the right ruther is starting at -0.6 rad.

In general, adding new elements to the models is very straight-forward, as demonstrated with the cranck-rod, DC motor and voltage source subsystems. Furthermore, most models in the literature treat all non-moving surfaces as a whole, but with the parent-child paradigm it is possible to define fixed sub-structures but with particular aerodynamic and inertial effects individually. This is, instead of defining the aerodynamic effects of a big body it is possible to divide the body in several parts with each their own aerodynamic effects and external forces.

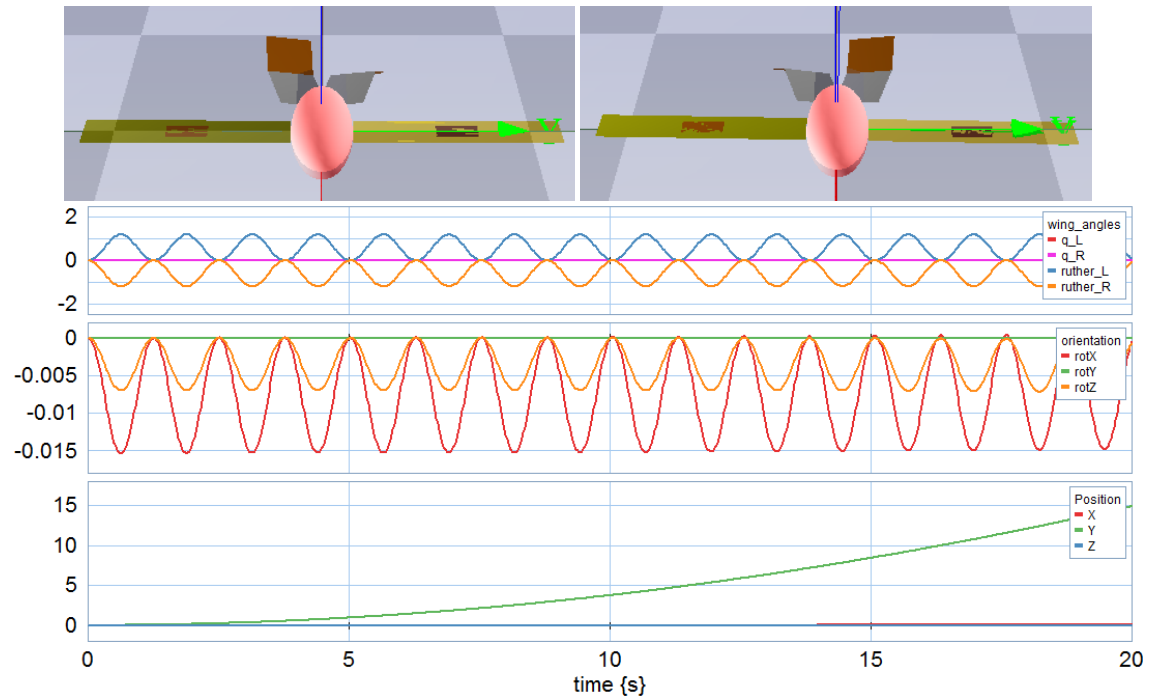


Figure 5.36: Robird parent-child model 3D simulation moving ruther alternatively (top graph) to generate a roll (rotation around X) and yaw (rotation around Z) (middle graph) and a lateral displacement (bottom graph).

(File: "\Parent_Child_Modelling\Literature_Models\Robird\Robird_ruther.emx").

5.10 Robird with Flexible Wing: Comparison to a Bond-graph Model with Flexible Wing

In the previous section 5.9 it was developed a model imitating the Robird robot [Folkertsma et al. (2017)]. Nevertheless, the model did not take into account the flexible structure of the wings. In this section such flexibility of the wing will be included, but the tail and flaps will be removed in order to focus only on the wing flexibility. Furthermore, in the literature there are some recent studies that implement flexible flapping-wing UAV models with bond graphs. In this section it will be also demonstrated how easy it is modelling a flexible flapping-wing UAV with the parent-child paradigm against the bond graphs structure presented by [Abbasi et al. (2021)].

But before diving into the actual model let's first introduce the concept of **passive actuator**, since they can be used to represent, among other things, a flexible wing. A passive joint would merely consist on a spring-damper joint, as exemplified in figure 5.37, where the definition of the values of the damper and the spring determine the flexibility of the union. An I-element (mass) is necessary to be included in order avoid causality issues and should be very small (i.e. $i \leq 0.01$). The inertia of the body section is already accounted by the child subsystem itself. Also, in the case of a flexible wing point the resistance can be set very small as we can consider the two wing sections are completely attached and there is no friction between them. Including the three elements R, L and C is for a matter of either causality issues (L) or for standardization of the passive joint (R). But in the case of a flexibility point the most important element is the spring (C).

A wing can be divided in multiple sections connected by "passive joints" for better accuracy in the implementation of the flexibility of the wing. Whats more, the values of the spring and inertia (defined inside the *child* subsystem) can be obtained from the density, cross sectional area and the flexural rigidity as explained in the book [Merzouki et al. (2013a)], even though

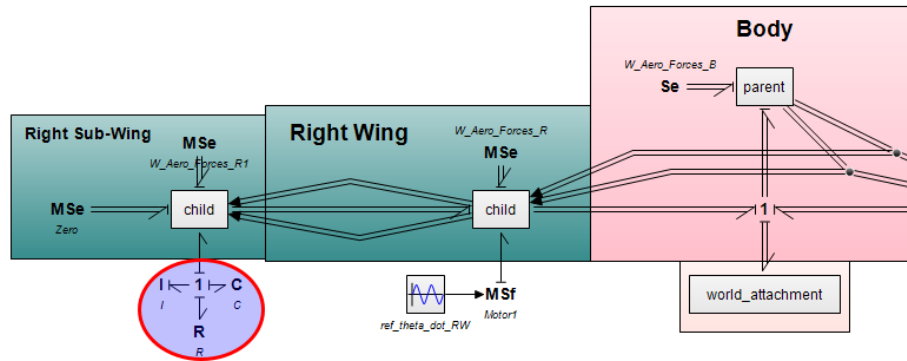


Figure 5.37: 20-sim model with the joint between sub-wing and wing passive, represented as a spring (C element) and a damper (R element).

(File: "\Parent_Child_Modelling\Literature_Models\Robird_flexible\Robird_passive_joint.emx").

that won't be tackled in this thesis. The flexibility values that will be used for the flexible wing model won't be realistic, as there are no aerodynamic forces (no air modelled) applied on the wings. Otherwise the flexible effects wouldn't be easily appreciable. The only forces that are present in the model affecting the wings are the inertial forces.

In figure 5.37 it is demonstrated the simplicity of this implementation, where the sub-wing has no actuation on its joint, but a spring and damper element instead. Also in figure 5.38 the effect of this passive joint is shown. When wing is moving downwards the sub-wing is moving upwards and vice versa, due to the inertia and thanks to the spring-damper configuration, resulting in a flexible joint/wing. This represents a wing with one single flexibility point, but it can be easily extended in case a more complex flexible wing was to be modeled, adding more flexibility points along the wing.

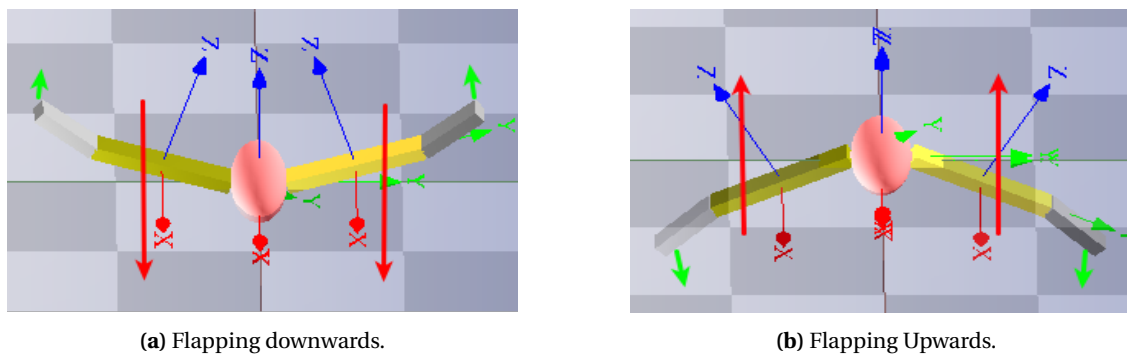


Figure 5.38: Passive joint Subwing-Wing 3D simulation. (Flexible wing).

(File: "\Parent_Child_Modelling\Literature_Models\Robird_flexible\Robird_passive_joint.emx").

Once the passive actuator is introduced we can move onto the **bond-graph flexible flapping-wing model** of [Abbasi et al. (2021)]. First of all, by looking into figure 5.39, it stands out the large amount of elements present in the bond-graph model. Only the main rigid body is composed of 20 elements that need to be configured. With the parent-child philosophy this bunch of elements would be reduced to one *parent* subsystem only. It is very noticeable as well the large amount of "TF" elements or reference frame transformations present in the model. If we focus only in the "Rigid Body BG" and the "Elastic Beam Wings BG" these elements are completely unnecessary in the case of a parent-children model, since the changes of frame are automatically taken into account by the children and parent without the need of any explicit element. More specifically, it is the *joint* subsystem inside the *children* the component which deals with the frame transformations. It is true that the DC motor, together with the crank-rod

mechanism cannot be implemented with the parent-children structure, but that part, in bond graph form can be directly interconnected with the rest of the model in parent-child form or even more simple blocks can be used, as done just before in the first Robird model from section 5.9, since 20-sim already provide with electrical and mechanical iconic diagrams, including a crank-rod code block, that make the implementation of the electric motor and mechanics much easier or at least, visually much clearer, more meaningful, as shown in figure 5.40a.

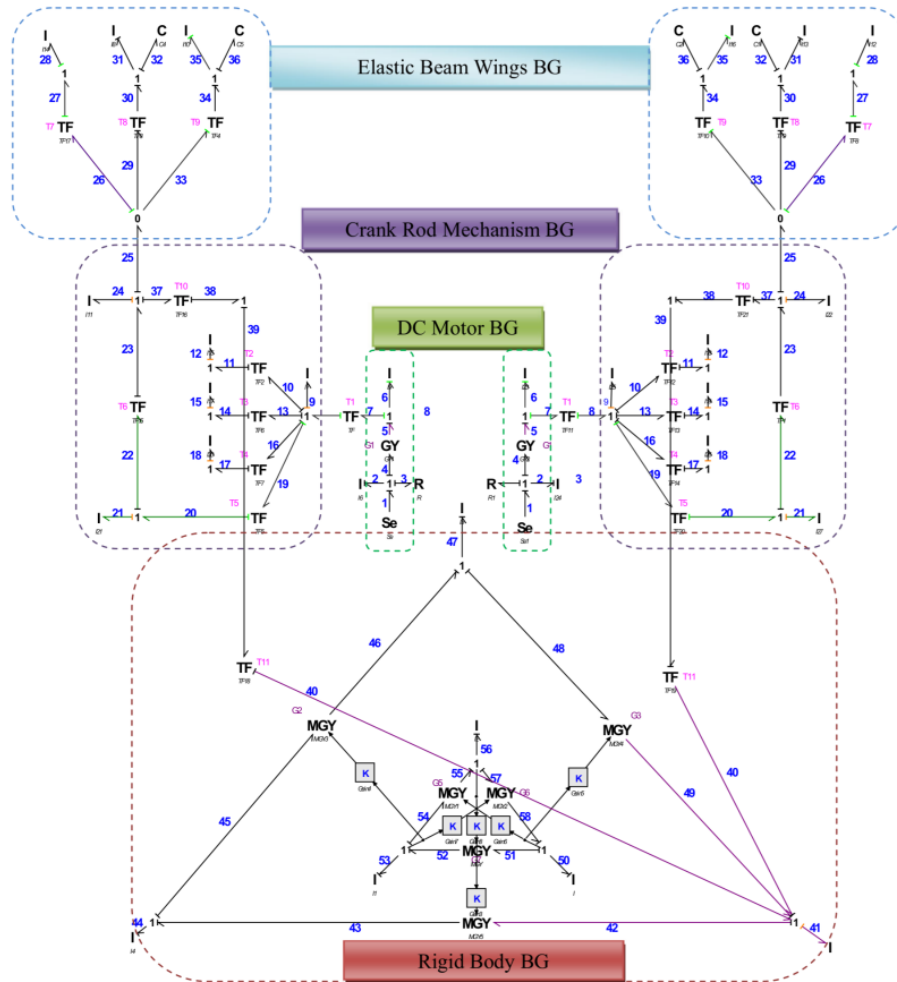
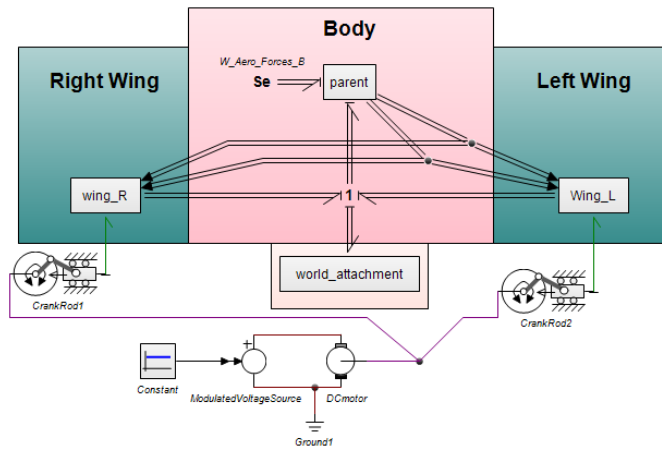


Figure 5.39: Bond graph flexible wing literature example. [Abbasi et al. (2021)]

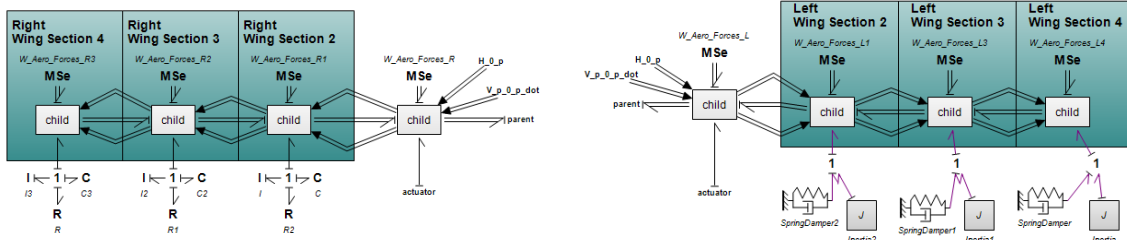
The "Elastic Beam Wings BG" part of the bond-graph model represents a flexible beam-like wing, expressed as a beam with a flexibility point in the middle. In other words, two rigid beams joined by a flexibility point/joint. In the parent-children model though, instead of one flexibility point there will be three, to take the experiment one step further (an example with one single flexibility point is already shown in figure 5.38), which means that the wing will be divided in 4 parts or sections. Also it is considered that including several flexibility points is more faithful to a real wing, since its elasticity normally vary along the wing together with its cross-section.

As it was already done for previous models, the 20-Sim implementation of the flexible flapping-wing model, shown in figure 5.40a, is represented in an abstracted form, as subsystems. This is, the wings, that are actually divided in four sections each are contained under the "wing R" and "Wing L" subsystems. This subsystems are expanded in figures 5.40b and 5.40c where two different options for implementing the flexibility of the wings are shown, the first with bond-graph standard diagrams and the second with mechanical diagrams. Both represent equivalent

spring-damper systems. The inertia (mass) value should be selected very small as mentioned before, as it would represent the mass of the joint.



(a) Outer view of flapping bird with multiple flexibility points distributed over the wing.



(b) "Wing R" block inside. Flexibility option 1, using standard bond-graph diagrams.

(c) "Wing L" block inside. Flexibility option 2, using mechanical diagrams.

Figure 5.40: Robird model with flexible flapping-wings equivalent to the bond-graph flexible flapping-wing model presented by [Abbasi et al. (2021)], implemented in 20-Sim with the parent-child methodology.

(File: "\Parent_Child_Modelling\Literature_Models\Robird_flexible\Bird_Multiflex.emx").

The model includes the same actuation of the wing introduced for the Robird in section 5.9. A crank-rod mechanism actuated by a DC motor controlled by a voltage source. The "world attachment" this time is set to constrain all dofs but the linear movements on the Z axis and the rotation around X (roll) for these simulations. Leaving the rotations around X free allow us to prove that both flexible wings implementation, even though the different shape, behave exactly the same. Otherwise, the bird's flapping wouldn't be symmetric and it would rotate around X.

The physical parameters of the 20-Sim model are presented in table 5.8.

Parameters	Model
Body Mass	730 g
Body Radius	20 cm
Wing Mass	18 g
Wing Length	100 cm
Subwings Mass	8 g
Subwings Length	50 cm

Table 5.8: Flexible wing model parameters.

(File: "\Parent_Child_Modelling\Literature_Models\Robird_flexible\Bird_Multiflex.emx").

The figures in 5.41 a-f demonstrate the flexible behaviour of the wings. The only actively actuated joints are the shoulders, while the other six (three per wing) are "free" (passive). When moving downwards these flexible section point upwards due to the wing beat velocity and the inertia of the sections. When the wing reaches the bottom part the acceleration of the wing is negative, leading to the flexible sections to keep moving downwards due to the inertia again. The opposite situation happens when the wing reaches the top part of the wing beat, that again the acceleration changes from negative to positive, so the wing will decrease its velocity until it reverses, but the flexible sections will remain climbing propelled by the inertia a bit longer. The flexible elements are so to say delayed wrt the main wing section. Such effect is more clear in image 5.42, where the momentum in Z of the four sections of the left wing are overlapped. Each section's momentum is slightly delayed from the precedent and the absolute value grows bigger progressively as well, as the section is located further from the main body, so their momentum is larger.

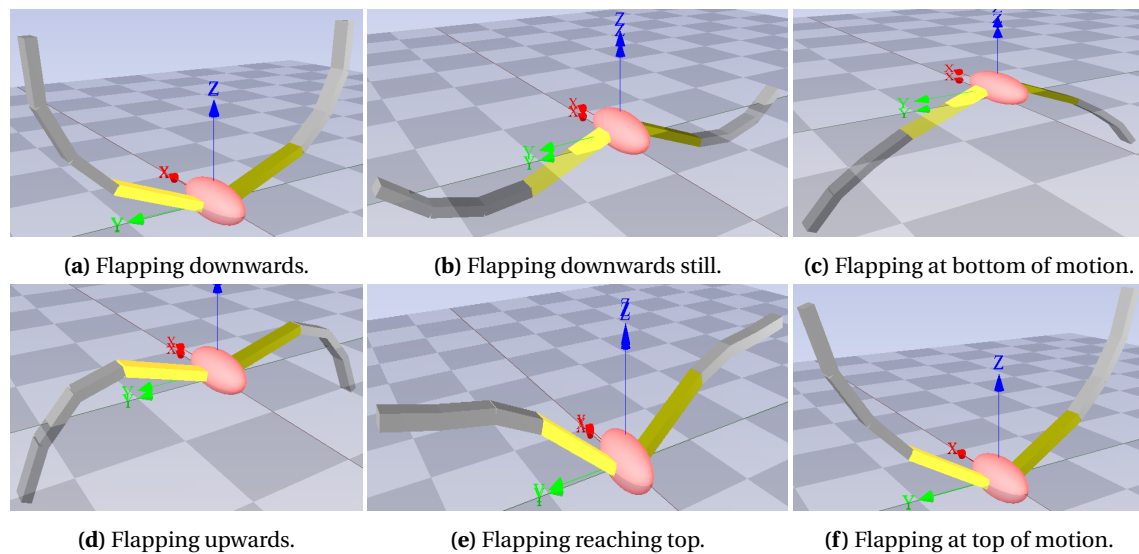


Figure 5.41: Flapping with several flexible wing sections.

(File: "\Parent_Child_Modelling\Literature_Models\Robird_flexible\Bird_Multiflex.emx").

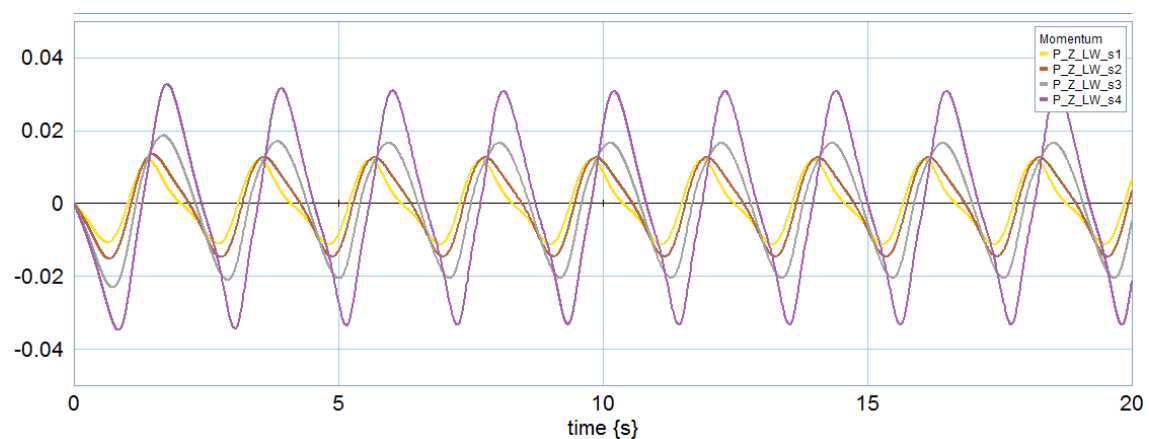


Figure 5.42: Momentum in Z of the four left wing section during wing-beat. Section's momentum are delayed one from each other due to modeled flexibility in the joint. P_Z_LW_s1 stands for momentum in Z of left wing section 1 and so on.

(File: "\Parent_Child_Modelling\Literature_Models\Robird_flexible\Bird_Multiflex.emx").

5.11 Robotic Bat.

At last, [Colorado et al. (2012)] present a very complete paper of their design of a robotic bat, including several experiments they perform, where, mainly, inertial effects of the bat's wings flapping and morphing are measured. The wings in the original paper have six dof each, distributed in: two dof in the shoulder, one dof in the elbow and three dof in the wrist (see figure 5.43). Nevertheless, in the 20-Sim implemented model, shown in figure 5.44, the wrists are simplified to have one single dof each, this is, a total of four dof instead of six per wing.

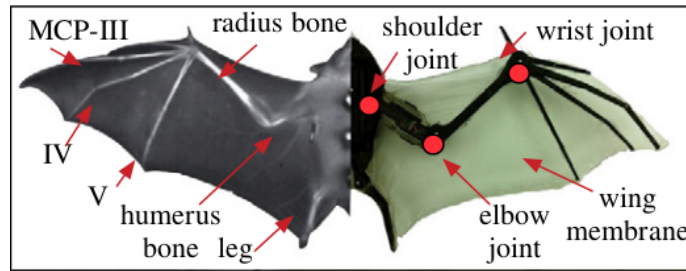


Figure 5.43: Literature example of bat with six dofs per wing and high inertial effects. [Colorado et al. (2012)].

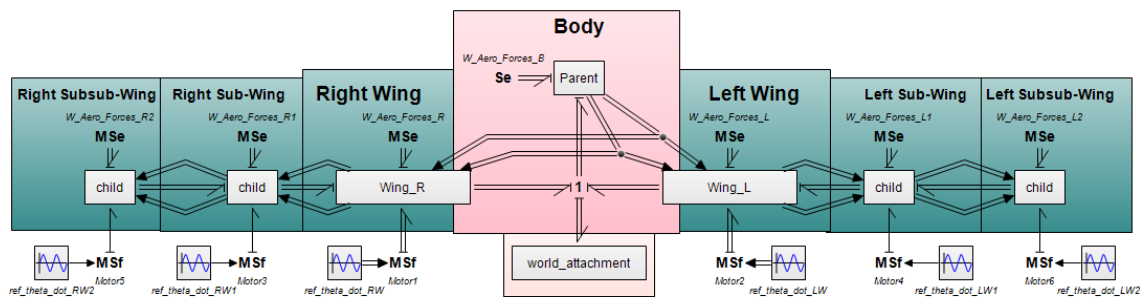


Figure 5.44: 20-Sim model of a flapping bat with four degrees of freedom per wing. (File: "\\Parent_Child_Modelling\Literature_Models\Bat\Bat_up_forward.emx").

The model built with the parent-child approach consists on a parent, the main body, with three children per wing, representing three wing sections. The elbow and wrist joints are represented by simple joints, but the shoulder joint is modelled using a multi-dof joint so it can actuate the two dof for the flapping (X) and the sweeping (Z) rotations. Furthermore, the model has no air and the gravity force is counteracted by applying a force onto the main body (parent) equal to the sum of weights of all the elements (parent plus children), so the the inertia effects during the bat's flight are isolated. The orientation of the body used in the experiments is taken into account when counteracting the gravity force, so it is always applied on the world's Z axis. Besides, the "world attachment" acts as a test bed that constrains the dofs of the bat depending on the experiment. The authors in [Colorado et al. (2012)] attach the bat completely to a bar and they measure the forces the bar suffer due to the bat's motion, but we will free the robot for our simulations, since we can equally see the effects of inertia this way.

Experiment 1:

One of the experiments carried out by [Colorado et al. (2012)] consist on testing the configuration and orientation of the bat's body and the wings for it to move forward and upwards only using the inertia of the wings movement, without taking advantage of any aerodynamic effect. In the experiment, they determine that the bat starts exerting a positive force in X and Z (for-

ward and upward) when the pitch angle (α) reaches 5.5° (nose looking up, see figure 5.45). They also experiment with larger angles, up to 20° .

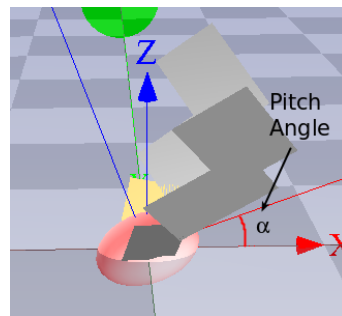


Figure 5.45: Pitch angle α of the bat.

For the up-forward movement there is only need for the shoulder joint to actuate its two dof, the flapping (X) and the sweeping (Z) rotations, while the rest of joints should remain static. In this case, the "world attachment" will constrain only the movement around Y, while leaving the other five dof free.

Note: In fact, even though the bat shouldn't rotate around X nor Z since the wing motion is symmetrical wrt those axis, those dofs can also be constrained to avoid unexpected behaviours due to numerical errors.

Either way, both, the flapping and sweeping motions occur at 2,5Hz frequency ($\approx 16\text{rad/s}$), so the tip of the wings draw a circle. The complete wings movement is very similar to a human swimming butterfly style. Elbow and wrist joints are kept static during the whole motion, configured with the same angles (or at least close enough) as used by [Colorado et al. (2012)]. These angles, among other physical parameters, are listed in table 5.9.

Parameters	Model
Body Mass	125 g
Body Length	60 cm
Body Radius	20 cm
Wings Mass	10 g
Wings Length	3 cm
Wings Chord	3 cm
Subwings Mass	18 g
Subwings Length	6 cm
Subwings Chord	4 cm
Subsubwings Mass	20 g
Subsubwings Length	6 cm
Subsubwings Chord	4 cm
Body Pitch angle	5.5 up to 20°
Wing sweep angle (wrt body)	$(-)$ 30°
Subwing sweep angle (wrt wing)	$(-)$ 45°
Subsubwing sweep angle (wrt subwing)	$(+)$ 90°
Flapping rate	16 rad/s
Other Morphings rate	0 rad/s

Table 5.9: Bat moving up and forward model parameters.

(File: "\Parent_Child_Modelling\Literature_Models\Bat\Bat_up_forward.emx").

The model constructed on 20-Sim employing the parent-child structure was used to test the minimum pitch angle for the bat to climb and move forward. The tested angles were the same

as those used by [Colorado et al. (2012)], from $\alpha=5.5^\circ$ up until $\alpha=20^\circ$. Figure 5.46 shows the inflection point when the bat transitioned from diving to climbing. While the bat would move forward with all tested pitch angles ($\alpha=5.5^\circ$ to $\alpha=20^\circ$), the climbing is different. Conversely from the mentioned paper the bat won't climb with a pitch angle of $\alpha=5.5^\circ$, but it would need at least a pitch angle of $\alpha \approx 8.35^\circ$ instead. From there on the bat will climb with any pitch angle up to $\alpha=20^\circ$. This result is, as already mentioned, proven in figure 5.46, where the Z position graph of the body is shown for $\alpha=8^\circ$, $\alpha=8.35^\circ$ and $\alpha=8.5^\circ$ (from bottom to top).

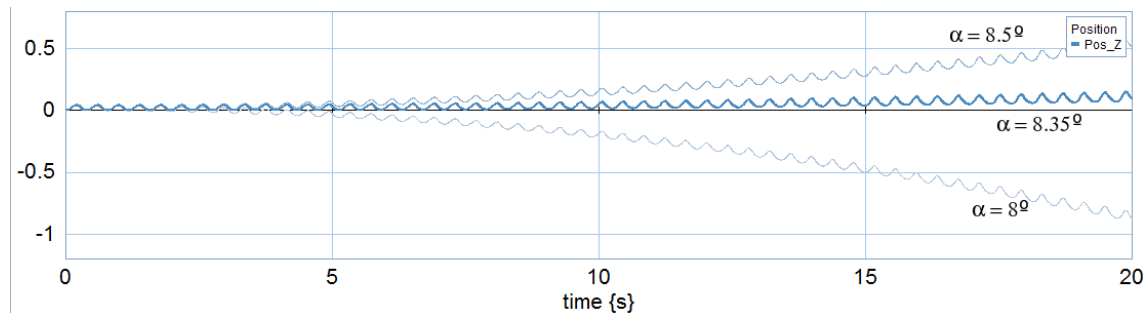


Figure 5.46: Z position graph of the bat model while performing a circular wing motion with $\alpha=8^\circ$, $\alpha=8.35^\circ$ and $\alpha=8.5^\circ$ (bottom to top line).

(File: "\Parent_Child_Modelling\Literature_Models\Bat\Bat_up_forward_8.emx").

(File: "\Parent_Child_Modelling\Literature_Models\Bat\Bat_up_forward_8_35.emx").

(File: "\Parent_Child_Modelling\Literature_Models\Bat\Bat_up_forward_8_5.emx").

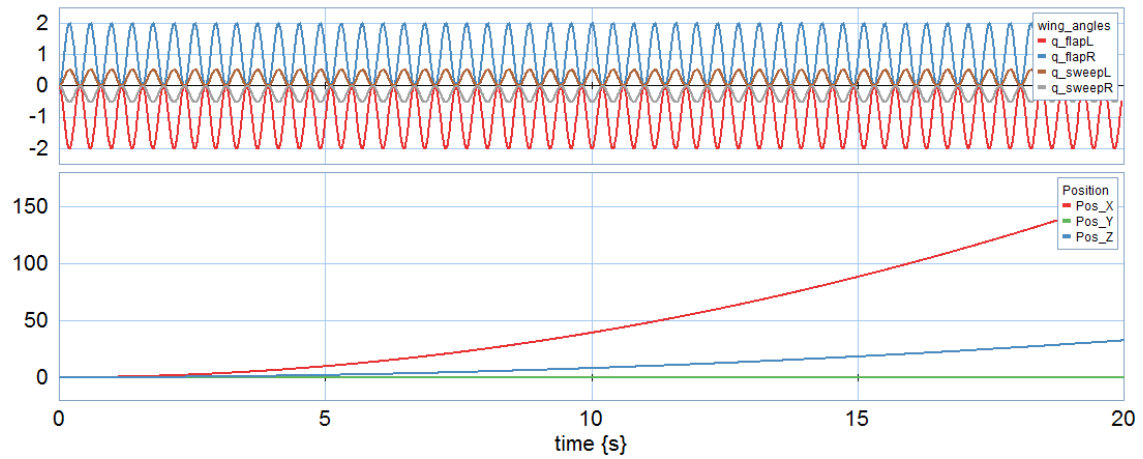
Furthermore, figure 5.47 intends to exemplify the motion of the bat robot with a pitch angle of $\alpha=20^\circ$. The bat clearly moves front and upwards thanks to merely the inertia momentum, as demonstrated in the graph from figure 5.47a, where also the motion of the wing (shoulder joint) is exhibited.

Experiment 2:

Another experiment carried out by [Colorado et al. (2012)] consisted on dynamically morphing the wings during the flapping in such a way that a rotation of the body around X and Z (roll and yaw) is generated thanks to the use of the inertias only. The experiment again test the isolated effect of the inertial forces, leaving out the aerodynamics. The wings during the rotation motion morph asynchronously, generating very different trajectories one wing from the other. Actually, one wing remains mainly tucked, while the other is mostly extended during the complete stroke. [Colorado et al. (2012)] explain their experiments quite extensively, so we can try to imitate it.

But before simulating anything the bat 20-Sim model requires few slight changes in order to be capable of generating a similar motion to that from [Colorado et al. (2012)]. The main change is related to the complex generation of the velocity path, that instead of using simple "wave generator" now the model uses "motion profile wizard" for actuating the elbows and wrists joints. The second change is related to the gravity compensation, that is applied on each body element individually as explained in appendix A.2.3, since this is a more accurate way to compensate it and takes into account the orientation of the bodies at all time (in case more dofs than yaw rotation were freed). Before the gravity of the whole body was compensated on the parent in a very simplistic way and with a fixed orientation. Lastly, the "world attachment" is set to only constrain the Y and X rotations (pitch and roll), so only the translations and Z rotation (yaw) will be appreciated.

The physical parameters for this model and simulations are presented in figure 5.10.



(a) Flapping angles and position of a bat-like model.

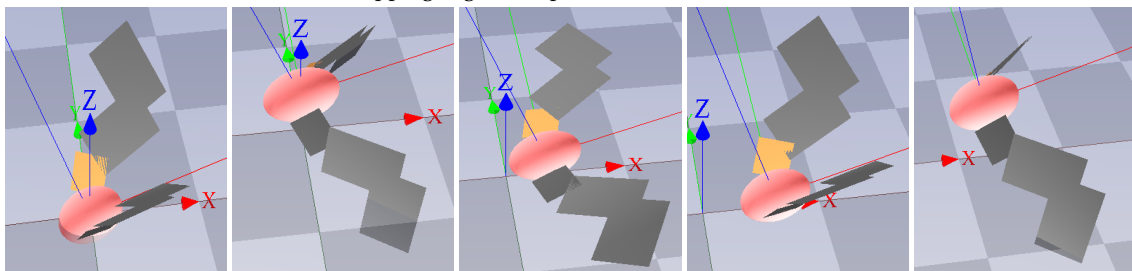


Figure 5.47: Flapping Bat moving upwards and forwards thanks only to inertia with pitch angle of 20° (nose up).

(File: "\Parent_Child_Modelling\Literature_Models\Bat\Bat_up_forward.emx").

Parameters	Model
Body Mass	125 g
Body Length	60 cm
Body Radius	20 cm
Wings Mass	10 g
Wings Length	3 cm
Wings Chord	3 cm
Subwings Mass	18 g
Subwings Length	6 cm
Subwings Chord	4 cm
Subsubwings Mass	20 g
Subsubwings Length	6 cm
Subsubwings Chord	4 cm
Body Pitch angle	-20°
Wing sweep angle (wrt body)	(+) -20° to -34.9°
Subwing sweep angle (wrt wing)	(-) 30° to 58.65°
Subsubwing sweep angle (wrt subwing)	(+) -90° to -107.2°
Flapping rate	16 rad/s
Wing Sweep rate	16 rad/s
Subwing Sweep rate	32 rad/s
Subsubwing Sweep rate	32 rad/s

Table 5.10: Bat turning flight model parameters.

(File: "\Parent_Child_Modelling\Literature_Models\Bat\Bat_Rotate.emx").

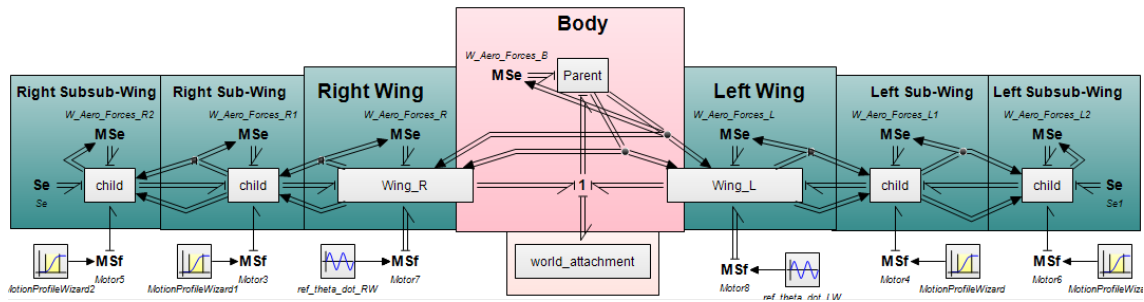


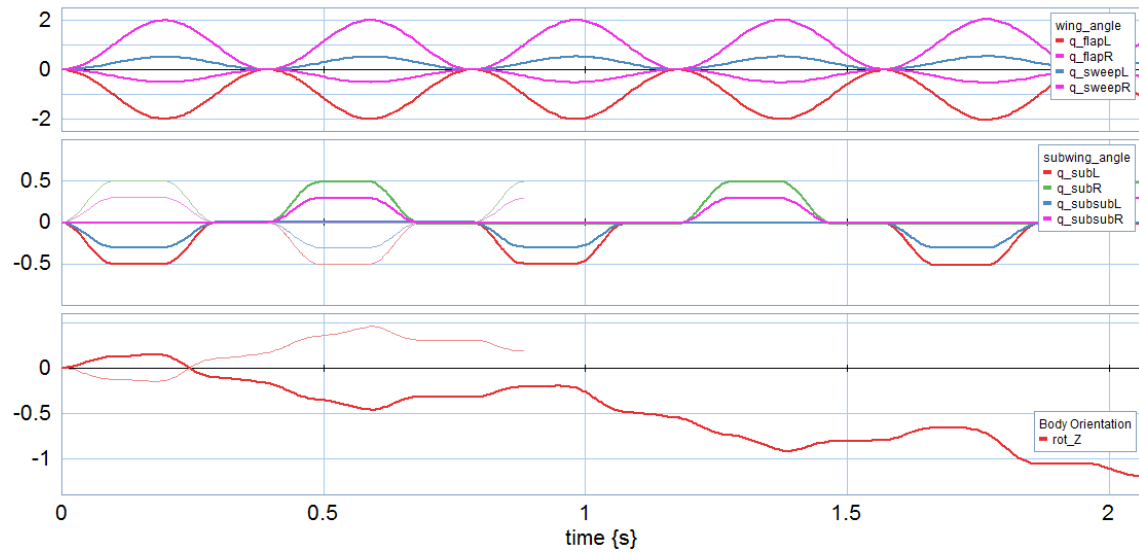
Figure 5.48: 20-Sim model of a flapping bat with four degrees of freedom per wing including a gravity compensation and motion wizards for complex motion profile specification. (File: "\Parent_Child_Modelling\Literature_Models\Bat\Bat_Rotate.emx").

Generating the motion in 20-Sim is really challenging. For simulating very few seconds it required a very big effort and reducing the accuracy of the algebraic solver and the maximum integration error considerably. Also a new motion generator is introduced, that will help us formulate sinusoidal waves that are only generated once every two periods, as represented in the middle graph from figure 5.49a. It should be mentioned that the frequency of the morphings is quite high, at least compared to previous study cases. For this simulation the frequency of flapping and sweeping (shoulder) is 2.5Hz ($\approx 16\text{rad/s}$), while the frequency of the elbow and wrist morphings is the double, 5Hz ($\approx 32\text{rad/s}$) (see table 5.10).

The simulation consists on slowly generating a rotation by asynchronously tucking the elbow and wrist joints of one wing only during one stroke, leaving the other elbow and wrist static and vice-versa during the next stroke, while flapping simultaneously. The shoulder motion, this is, the flapping and sweeping motions, are represented in the top graph of figure 5.49a. The tucking and extending motion is shown in the middle graph of figure 5.49a. Looking at both top graphs together it becomes clear the whole motion of the wing, as first one wing is morphing (blue and red lines) during the first stroke, but during the second stroke it is the other wing that morphs instead (green and pink lines).

The result of such motion depends on which wing starts morphing, that the bat will start rotating in one direction or the other (see in figure 5.49a the shadowed lines of a previous run swapping the starting morphing wing). The rotation of the bat around Z is shown in the bottom graph of figure 5.49a and can be further verified in the 3D simulation shown in figure 5.49. The bat will initially rotate slightly to the left in this case, to next start rotating to the right and gain momentum, therefore continuing rotating to the right from that moment on.

The model demonstrates its effectiveness for generating a rotation using the inertial forces by dynamically modifying the configuration of its wings asynchronously. Nevertheless, it is found to be very difficult to generate these complex motions as to imitate the experiments of [Colorado et al. (2012)] more accurately. The simulation is just a bare approximation to their work.



(a) Flapping angles and position of a bat-like model.

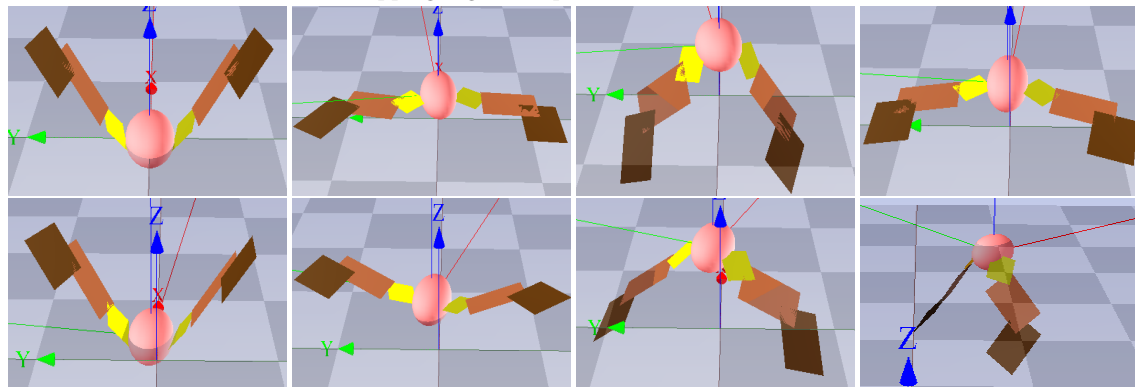


Figure 5.49: Flapping Bat rotating thanks to wing highly complex motion.
 (File: "\Parent_Child_Modelling\Literature_Models\Bat\Bat_Rotate.emx").

5.12 Conclusion

Several case studies were presented along this chapter, each of them with different characteristics. The models are presented starting with the most simple examples to the most complex, introducing new elements progressively. The first example is just the base model that will be used later for all the literature case studies, but it is used to verify the correctness of the modelling approach, showing that the energy and power balance and the dynamics behave as expected. Energy balance is a very important concept in this modelling and simulation methodology, as most simulating tools tend to have energy leaks, leading to breaking at some point. Not having those energy inconsistencies can let us do longer experiments in time. Even though the energy balance is not shown for every case study it is fulfilled, since the subsystems used are exactly the same all the time.

From section 5.3 and on all experiments are performed on case study models with a rather high degree of satisfaction in the (not numerical) results. The potential advantages of the modelling and simulation method noted along the previous chapter of the thesis are somewhat reflected in these model, starting from the compact Child, that is an abstraction of a joint plus a simple child, and it is a port-Hamiltonian subsystem composed by two port-Hamiltonian subsystems.

Furthermore, the so called "world attachment", that, as explained before, acts as a test-bed was of great utility all along the chapter, since it enabled the execution of many different experiment, from allowing only the rotation around one axis in the case of the gull wing (section

5.3), locking the rotations around the different axis when asymmetric morphing had to take place, avoiding the models from rotating like crazy, to the Robird (section 5.9) and the robotic Bat (section 5.11) for testing the inertial effects generated by the morphs.

All three type of joints were used and experimented with, from the rotational joint to the prismatic joint and the multi-dof joint. All three of them with satisfactory results. The performance of all three joints was perfect, leading to super short processing times way below 1s for simulating 20s of experiment (see figures 5.50a and 5.50b), except in the second experiment with the robotic bat in section 5.11, but for a totally different reason, that is the generation of the motion.

```
Starting Simulation
Simulation Running...
Simulation stopped after 0.083 seconds.      Model calculations: 4598      Number of output points: 1032      Average steps per second: 55397
===== Finished Simulation - 0 errors, 0 warnings =====
```

(a) Robird model processing time for a simulation of 20s.

(File: "\Parent_Child_Modelling\Literature_Models\Robird\Robird.emx").

```
Starting Simulation
Simulation Running...
Simulation stopped after 0.27 seconds.      Model calculations: 14620      Number of output points: 3673      Average steps per second: 54148
===== Finished Simulation - 0 errors, 0 warnings =====
```

(b) Robotic Bat model processing time for a simulation of 20s.

(File: "\Parent_Child_Modelling\Literature_Models\Bat\Bat_up_forward.emx").

Figure 5.50: Processing time of two simulations of 20s with very different components and motions (Robird and Robotic Bat).

In fact, also the generation of motions was tested by means of different actuators, including mechanisms like the crank-rod moved by a DC motor, which proved to work perfectly together with the parent-child subsystems. Even flexibility was successfully implemented in a very simple and comprehensible way for the Robird in section 5.10. Only problems were found when trying to implement more complex motion waves, as it happened in the case of the second experiment with the robotic bad in section 5.11, where all four degrees of freedom of the bat were actuated simultaneously, at high frequencies (2.5Hz and 5Hz) and following rather complex paths, at least compared to the rest of the case studies.

Additionally, the Robird model with flexible wings presented in section 5.10 was also used to positively compare the parent-child modelling method against a bond-graph model, exalting the ease of understanding of the parent-child implementation, as the models with this methodology are more abstracted, while in bond-graphs there are too many elements, which also mean too many parameters to tune and interconnect, apart from the use of frame transformation elements that in parent-child approach are directly included in the parent and child subsystems, but in bong-graphs they are explicit.

Nevertheless, the possibility to model such a vast amount of models in a quite agile way exposes the capabilities of the parent-child modelling and simulating approach. While the first model might take a while to build it, once we got one, the rest are pretty straight-forward, enabling a very fast modelling of the wing-morphing UAVs. The most challenging part have proven to be the motion generation.

6 Conclusion

6.1 Conclusions

This thesis was meant to answer three main research questions:

- What's the state-of-the-art of the morphing-wing robots?
- What would be the energy-based atomic modules to support the geometric port-Hamiltonian modelling and simulation framework for morphing-wing aerial robots?
- How applicable is the framework for modelling and simulating literature case studies?

What's the state-of-the-art of the morphing-wing robots?

The state of the art, as analyzed during chapter 2, is considerably reduced. Nevertheless some conclusions can be taken from what was observed.

For starters, there is a noticeable increment of morphing-wing UAVs since 2010 and more importantly an increment in the amount of flapping-wing UAVs developed. However, this is a normal consequence of two factors, the general explosion of the UAVs industry and the growing use-cases for morphing-wing UAVs. In any case, the vast amount of multirotors developed in the last years compared to the little amount of flapping-wing UAVs can be considered a direct consequence of the design and aerodynamics complexity of the flapping-wings.

Fixed-wing bird-mimicking UAVs are quite common to find, but propelled by blades, as these robots are much easier to design since they behave like regular aeroplanes. But the broad existence of these robots make us believe that bird-mimicking robots could be of great importance.

Additionally, an initial research on the actual modelling and simulation tools was performed. Leading to the conclusion that all of them are poorly suited for modelling UAVs different from multirotors and fixed-wings.

All that derives in a inevitable conclusion. There are no more flapping-wing robots in the literature because the modelling and simulation tools aren't properly suited for the design of these highly complex, morphing UAVs at the moment. Therefore, our new and better modelling and simulating framework could help enormously to close the huge gap between aerial robots propelled by rotors and aerial robots propelled by flapping-wings.

The answer to what this first question leads to wondering, how that modelling and simulation framework should be so it can solve the actual problems of current tools and environments and enable the design of those complex aerial robots?.

What would be the energy-based atomic modules to support the geometric port-Hamiltonian modelling and simulation framework for morphing-wing aerial robots?

The modelling and simulation framework proposed in this thesis for solving the problems of current tools and environments is a geometric port-Hamiltonian framework supported on the Screw theory and Lie-group theory. The reasons, although already exposed during the introduction in chapter 1, they are numerous.

Current modelling and simulating tools don't allow energy-based modelling. They use block diagrams together with representations of the bodies using Euler angles, which are not geometric representations, or alternatively they use quaternions. Therefore, algorithms are not energy conservative nor the dynamics of the models configuration independent, meaning, in

the one side, that models tend to have energy leaks that derive in simulation errors and breaks when executing long simulations, and in the other side, that the dynamics model changes when the configuration of the robot changes. Conversely, in a geometric port-Hamiltonian framework these problems have no place. The dynamics of bodies, represented using the Lie Group mathematical tools, include the mass distribution and inertial effects that many modelling approaches ignore, as in most cases these inertial effects are accounted as disturbances by the controller. Whats more, the dynamics are configuration independent, which makes them very easily transformed to any coordinate frame and they are defined by means of twists and wrenches (six-dimensional velocity and force tensors). This results in a highly modular and flexible modelling framework, as the possibilities for defining the bodies are almost limitless. Last but not least, the framework is designed in a port-Hamiltonian style, which grants the model energy preservation and multi-domain qualities, as power becomes the common language between subsystems like the bodies, the air or the actuators.

The modelling and simulation framework proposed along this lines consist on the creation of merely three atomic modules, a parent, a child and a joint subsystems. All three represented as Dirac structures, which means that the sum of the exchanged power and energy in each structure is equal to zero. This is, the energy balance inside each atomic module has to be zero if properly implemented. The parent subsystem represents the main body from which all other bodies hang. The child or children are all other bodies apart from the main body (parent) which are connected to their parents through joints. Therefore, the joint is the atomic module acting as transmitter or translator between bodies. Additionally, a child can have more children hanging from them.

Thanks to the Lie Group particularities the expression of the dynamics of the bodies is very reduced. In fact, the subsystems that represent bodies, the parent and the child, both contain the dynamics of the body they represent in a super compact way, by means of one single equation. Two if we consider the computation of the gravity force (wrench), that in this proposal is calculated inside each body (parent or child). This, together with the independence from configuration makes the solution very powerful and modular.

Furthermore, if two port-Hamiltonian subsystems are connected the result is another port-Hamiltonian system. What this means is that its energy conservation characteristics are preserved when constructing larger systems unifying several port-Hamiltonian subsystems. In fact, the joint and the child atomic modules can be directly put together, as a child cannot be connected to a parent without a joint. Therefore a single subsystem can be created containing both, the child and joint subsystems, that as it is the result of two port-Hamiltonian subsystems it will be a port-Hamiltonian power conservative subsystem as well. Joining both subsystems imply many benefits, as the number of connections (ports) per model are greatly reduced, improving the readability and reducing the possibilities of an error.

As the joint and child are put together it can be considered that the modelling and simulation approach consists on **just two atomic modules**, the parent and a compact child, which contains the joint already in it. Moreover, the atomic modules were designed in a very contained manner, with the lesser amount of power-ports and tuneable parameters possible. In fact, the number of power-ports in a parent is reduced to two, while the compact child (joint + child) has four.

Similarly, the amount of parameters is as little as two for the parent (inertia tensor and initial configuration wrt the world), one for the child (inertia tensor) and two for the joint (Unit twist and initial configuration of the child wrt the parent), this is, three in total for the compact child.

The presented joint module contains the description of the degree of freedom that relates a parent with a child (or a child with another child). Such degree of freedom is actuated through the actuator port, which is a power port that expects an input twist (velocity). The fact that the

actuator port is also a power port means that the power can be transmitted by any source of any physical domain in a power conservative manner, which should also allow for the use of energy-based control laws.

Nevertheless, a joint with multiples degrees of freedom (multi-dof joint) was also proposed, which instead of actuating one single dof can actuate six, three rotations and three translations. While with a simple joint it is possible to describe rotational, screw or prismatic joints, with the multi-dof joint it becomes possible the definition of those three plus spherical, universal or cylindrical joints, offering a maximum level of flexibility and even simplifying more the specification of the subsystem's parameters.

How applicable is the framework for modelling and simulating literature case studies?

The potential of the parent-child modelling and simulation methodology is greatly proven in this document. A large set of case studies extracted from the literature were modelled satisfactorily. The focus was placed on the flexibility of the approach and not on the accuracy of the models in comparison to those of the literature.

Modularity was constantly demonstrated, showing the ease of use of the parent and child atomic modules, adding and removing new bodies to the models. Also the fact that bodies configuration are defined relative to the previous body in the tree-branch makes it really easy to define the models. While the first model might take some time to build, once the person is familiar with the subsystems and their configuration creating new models becomes super fast and easy.

Ten models in total are presented, of which the only the first doesn't correspond with any literature example, but it is just used to validate and verify the correct functioning of the subsystems. For the other nine different constructions, actuators and experiments are implemented.

The experiments developed intended to imitate those also developed in the original papers, nevertheless the parent-child models didn't include the air, which makes a huge difference in the behaviour of the robotic birds and which is the reason why it is very difficult to obtain numerical results. Nonetheless, some basic experiments with moving surfaces and the inertial effects provoked by those dynamically moving surfaces were held. As a matter of fact, the inertial forces generated by the movement of the bodies were in the expected direction. Furthermore experiments with the model of a robotic bat proved the generation of a forward and upward movement thanks to merely inertial forces generated by its four dofs morphing wings. It was tested the minimum pitch angle α to start generating the forward and upward movement and compared to that obtained by [Colorado et al. (2012)], which in our case was a minimum pitch angle of $\alpha=8.35^\circ$ instead of the $\alpha=5.5^\circ$ obtained by the authors. Additionally, rotations were also generated in several models thanks to asymmetrically morphing either the wings or elements in the tail.

The rotational, prismatic and spherical/universal type of joints were tested, both using the simple joint and the multi-dof joint, which proved to be very convenient.

Also several joint actuation methods were used, including mechanisms like the crank-rod moved by a DC motor and a source of voltage, demonstrating the multi-domain capabilities of the methodology, and a flexible joint, which enables the modelling of flexible wings. The approach allows for easily defining a wing as a set of wing sections, that in between have flexible unions.

Furthermore, wings and bodies can also be defined as sets of independent sections but rigidly attached for aerodynamics purposes. Most models in the literature treat all non-moving surfaces as a whole, but with the parent-child paradigm it is possible to define fixed sub-structures or surfaces with particular aerodynamic effects independently. Whats more, for some aero-

dynamic implementations, like the stripes theory approach proposed by [Abdelbadie (2021)], the possibility of dividing the wing in many subsections might ease the implementation of the aerodynamics.

Finally, it can be stated that the geometric port-Hamiltonian modelling and simulation approach proposed in this thesis solves many of the problems detected in the methods used in the literature, together with the current modelling and simulation tools, and it opens a new world of possibilities with such fast and modular approach, as it becomes a lot easier, and more faithful to dynamics, to develop and test new morphing-wing UAV configurations.

6.2 Limitations and Future Work Recommendations

The biggest limitation of this thesis, as made clear from the beginning, is the lack of air. The air was never modelled, as that is beyond the scope of this project. Nonetheless, not implementing the air and therefore not implementing any aerodynamics make the obtaining of relevant numerical data very challenging. This is the main reason why comparing the results in the absence of air from the parent-child models against those from the literature which include the air becomes so arduous. For instance, the Robird with flexibility model cannot be compared to its analogous from the literature in terms of flexibility, as if we want to appreciate that the model is flexible without aerodynamic forces it is necessary to make the model a lot more flexible so inertial forces become sufficient to deform the wing. This limitation lead to many vague comparisons between software models and study cases. Consequently, an inevitable future recommendation would be including the aerodynamic forces. A proposition would be joining the modelling and simulation framework developed in this thesis with the work from [Abdelbadie (2021)], as they already implement the aerodynamics of a wing in a port-Hamiltonian fashion.

A secondary limitation was 20-Sim. This modelling and simulation tool, while it is one of the few that allow power ports and energy-based modelling it is very counter intuitive. The internal code optimization algorithms are unpredictable, so sometimes it might be better to disable them when possible. The tool has potential, but it is very difficult to track errors or the causes of drifts in the graphs as a result of numerical errors. Moreover, algebraic loops are difficult to avoid. A good way to validate the framework would be implementing it in a different programming language or environment.

On the other hand, one limitation of the modelling and simulation method itself is the use of integrations, that will inevitably lead to some errors, which can be neglected in most cases, but its something to keep in consideration when performing long simulations. A future work recommendation related to this issue would be checking the exact origin of the error and to reduce it. For instance, the position can be introduced "manually" as a signal instead of integrating the velocity, since if we can provide a velocity path we can also provide the position and acceleration paths computed in advance.

Additionally, before continuing developing the modelling and simulation framework, an additional recommendation would be to implement a very basic example of which the dynamic behaviour is very well known that permit performing an experiment for validating the accuracy of the methodology. Not a morphing wing example, but a model with the minimum elements possible, this is, just a parent and a child. A possible example would be a pendulum. Truth is, this was done at the very beginning but it was not properly documented in time, but it is included in appendix B.1.

Moreover, the inertial forces are proven to exert a force in the direction we would expect them to do, but there are no numerical results that allow us to conclude if those forces constitute a large percentage of the overall forces exerted by those morphing elements when aerodynamic forces are also present or, if on the contrary, they can be neglected once the air is modelled.

The presented experiments let us believe that they imply a considerable amount of force, but this should be verified with numerical values. Thus, future work should demonstrate the relevance of the inertial moments in the overall forces. Not only that, but also the definition of the inertia tensors in the constructed case study models are in most cases erroneous, as the inertia is included with equal magnitude in all the directions of the body ($I_x = I_y = I_z = I$), resulting in spherical definitions of all the bodies. Also the inertias of the wings were defined as if the rotation was around the center of the body, which is also wrong. The topic is also commented in appendix B.1. This is something to take into account during the construction of future models, but it doesn't make a difference on the conclusions of the framework.

Finally, the modelling and simulation framework is intended for ultimately allowing the design of the complex control laws for the flapping-wing robots. Furthermore, the approach should permit the design and implementation of energy-based control solutions. This is undoubtedly a very important field for future research and development within the geometric port-Hamiltonian modelling and simulation framework.

A Appendix 1: 20-Sim Implementation

A.1 Before building a model

Building a model should be very straight forward if the subsystems are properly designed. 20-Sim allows the creation of libraries of subsystems, so it is very easy to add these libraries to the interface to then pick and place the new subsystems into the new model. Let's first explain very briefly the steps to follow to add the library of subsystems. In appendix A.6 it is also explained how to create the library.

Libraries are basically folders containing subsystems and models. Steps are illustrated in figure A.1. First go to *Settings»Options»Folders»Library Folders* and in here click on *Add* to select the folder containing the library. Finally click *OK*.

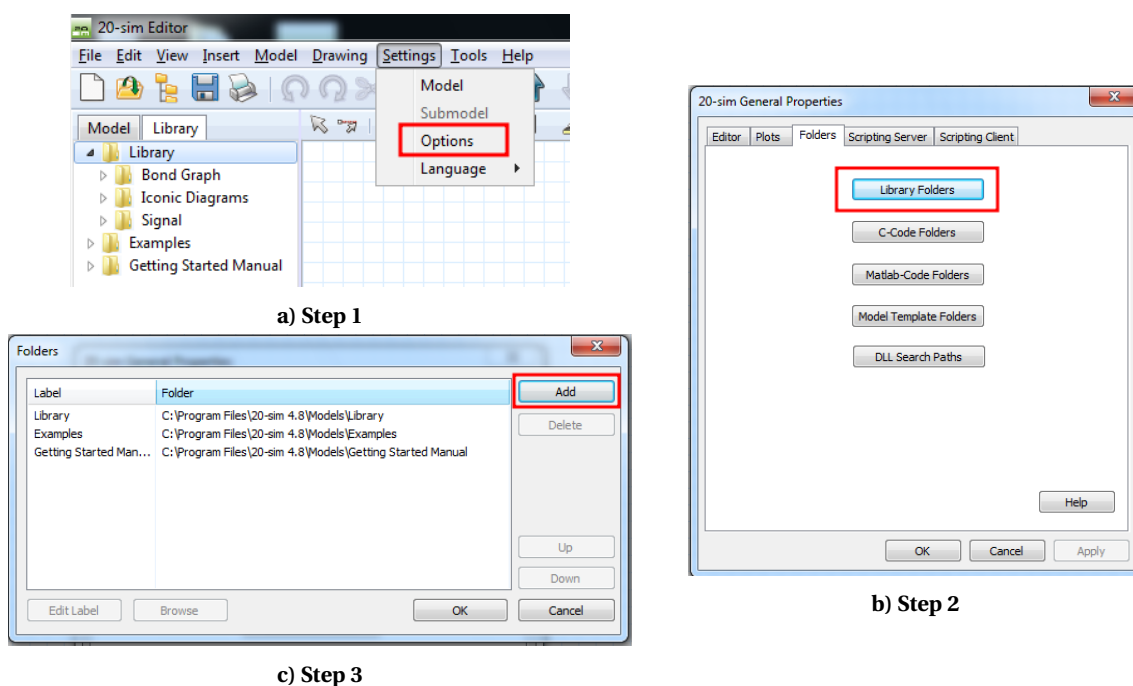


Figure A.1: Steps to add a submodels library in 20-Sim.

One very important thing the user should always do when building a model using these subsystems is specifying the program 20-Sim not to solve the algebraic variables. This solver introduce problems to the computation for the simulations, elongating the simulation time or even making it break. See in figure A.2 the steps to remove the option. This is done by simply going to *Settings»Model»Processing* and disable the option *Solve Algebraic Variables*.

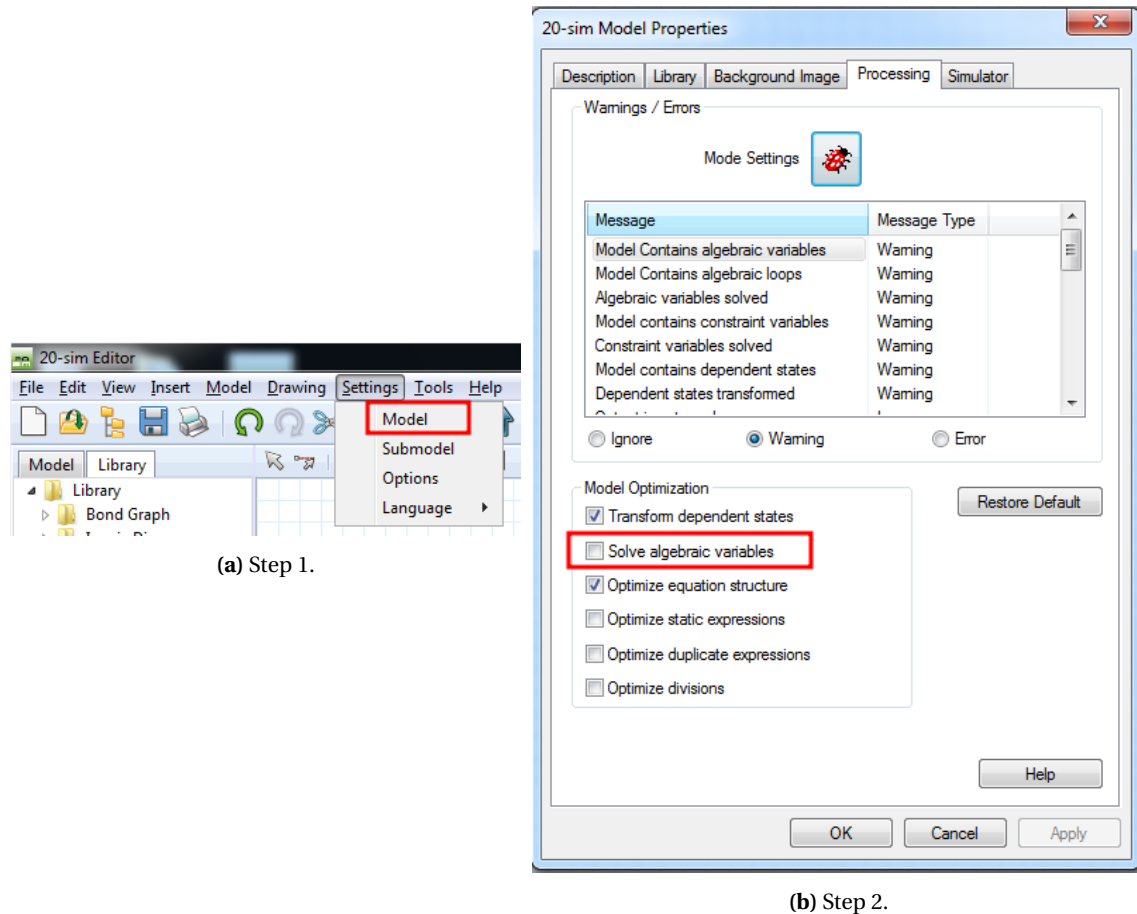


Figure A.2: Remove "Solve algebraic variables" option.

A.2 Building a basic model

Building a model should be very straight forward if the subsystems are properly designed. 20-Sim allows the creation of libraries of subsystems, so it is very easy to add these libraries that allow to easily pick and place the new subsystems into the new model.

A brief explanation on the steps to follow to create and add the library of subsystems is presented in appendices A.1 and A.6.

Let's consider a robotic bird with one single flapping wing as the most basic model to start with. Once the libraries are added and the algebraic variables solver is disabled the modeling can start by adding the sub-models or subsystems, taking into consideration that the *Child* is in compact form, this is, it comprises a *joint+child* (as previously shown in figure 4.13). The process for building a model model would be as follows:

- The first step is always to add a **Parent**, that will represent the main body of the flapping-UAV. The user is in charge of defining the shape and the initial configuration of each body, by means of the inertia tensor G and the initial homogeneous matrix H_p^0 . The code for such definition of the body can be included directly inside the global relations editor or it can be written inside a helping code block containing all global parameters and the definitions of all the components of the system. This way, if we have for example a body with two wings (three bodies), all three elements are defined together in the same block and the code inside the global relations editor is kept cleaner. This is the way the model in figure A.3 is built, where all bodies parameters are defined inside the block "globals", but this is completely up to the user.

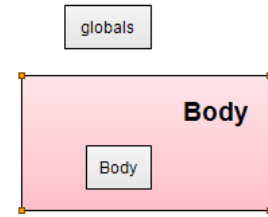


Figure A.3: Add a *Parent* as the main body.

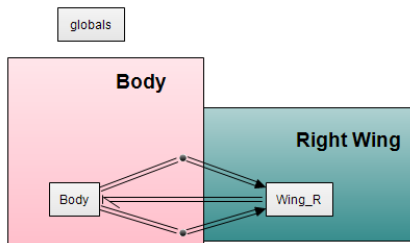


Figure A.4: Add a *Child* as wing and connect it to the *Parent*.

- Either way, the next step in the modelling is adding the **Child** representing the wing and connect it to the body. This connection can be done directly port to port if we only have one *Child*, as shown in figure A.4, or through a *One-junction* in case we have more elements, as in figure A.9, where the *One-junction* represent the sum of the *Children* exerted wrenches onto the body. Also the *Parent* transmits its actual configuration H_p^0 directly to the *Child*, together with the rate of change of the parent's twist $\dot{V}_p^{p,0}$. As recommended before for the *Parent*, the shape (G) and initial configuration (H)

of the *Child* can be coded inside the same helping block (i.e. "globals") used for the *Parent* or directly in their global relations editor, with the addition in the *child's* case of the unit twist ($S_c^{p,p}$) that defines the dof between bodies.

- Both body and wing elements, *Parent* and *Child*, have an energy input port for the **aerodynamic wrenches and other external wrenches**. In figure A.5 the aerodynamic wrenches are represented by sources of effort or modulated sources of effort. If it is the case that the *Child* block is the last element in the hierarchy then a **zero source of effort** should be connected to the "children" energy port of this last *Child*, as in figure A.5.

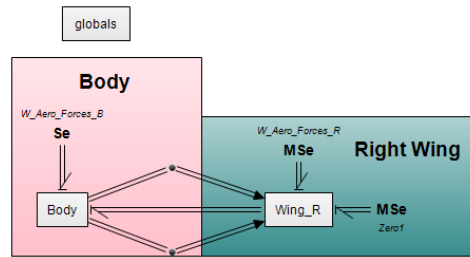


Figure A.5: Include the aerodynamic external wrenches.

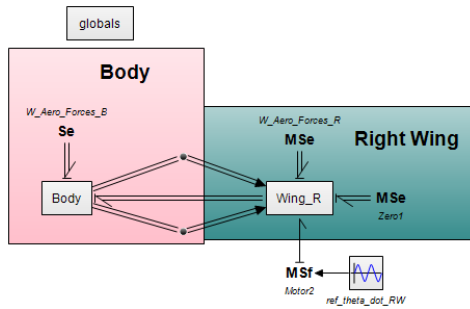


Figure A.6: Connect the actuator.

- At last, the *Child* require an **actuator**, active or passive, in order to move wrt the precedent parent. The design of the actuator is completely up to the user, but the output of the actuator has to be a joint velocity (ω or v). In figure A.6 the actuator is simply wave generator connected to a modulated source of flow.

- Additionally, a so called **World Attachment** can be connected to the parent through the previously mentioned *One-junction*, as shown in figure A.7. This is an element that imitates a test-bed platform. It enables/disables the 6 degrees of freedom of the whole model. It permits performing very different kind of tests liberating only the degrees of freedom of interest. The subsystem is very simple, consisting on a resistance-capacitor combination, as shown in figure A.8, which allow the control of the compliance of the test bed.

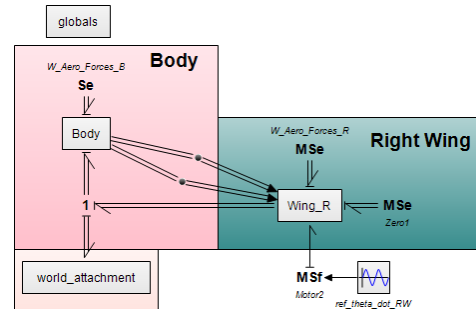


Figure A.7: Addition of a "World attachment".

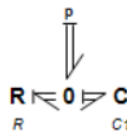


Figure A.8: World attachment/test-bed modelling.

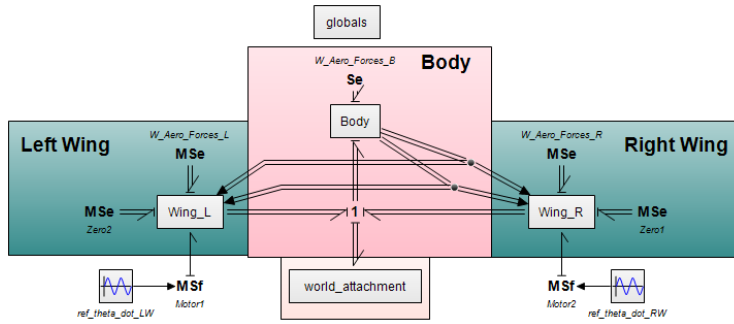


Figure A.9: Basic flapping robot model.

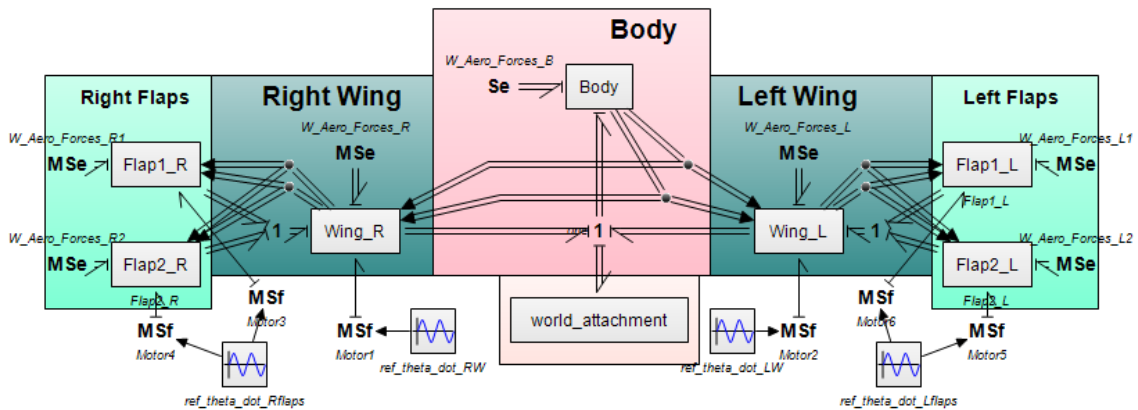
- The flapping-wing UAV construction will be finished once the parameters are defined consistently. Nonetheless, if we would like to expand the model to a two flapping wing UAV (bird-like) the process would be as easy as copy pasting, connecting and defining the new wing

parameters, resulting in the model represented in figure A.9.

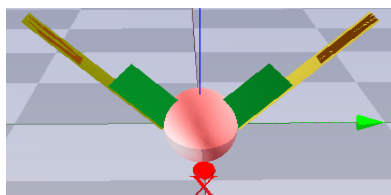
(File: "\\Parent_Child_Modelling\Other Models (tests)\Building_a_Model.emx").

A.2.1 Building a model with a Child with multiple children

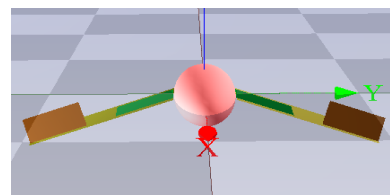
Child submodels can act as *child* and *parent* at the same time. Thus, any *Child* can be a parent of several children, like it is shown for the case of the main body's *parent* in figure A.9. In the same manner, if a child has more than one child, these need to be connected to the *Child* through a *one-connection* as done in figure A.10a. This model is considerably more complex than the one from figure A.9, as it is a big step forward, but it is just meant for illustrating the branched hierarchy of bodies in the modelling approach. It is built so it represents a flapping-wing UAV with two flaps per wing, as depicted in figures A.10b and A.10c. It comprises a *parent* with two *children* which in turn have two more *children* each. The children "Wing R" and "Wing L" act as children and parent at the same time, while "Flap1 R", "Flap2 R", "Flap1 L" and "Flap2 L" are merely children.



(a) 20-Sim model



(b) Flaps 1 up. Flaps 2 down.



(c) Flaps 1 down. Flaps 2 up.

Figure A.10: 20-Sim model with child-parent subsystems with two children each connected under them. The model represents a flapping-wing UAV with two wings and two flaps per wing. (File: "\\Parent_Child_Modelling\Other Models (tests)\Bird_basic_multichild.emx").

These last *child* blocks require 0-effort sources connected to the "children" power port (outermost power port). While signals can remain open (disconnected), power ports cannot.

A.2.2 Building a model with multi-DOF joints

When building a model with an universal joint or a *multi-dof joint* the procedure is basically the same as building a normal model, with the only difference of the parameters that the user have to introduce, as explained in section 4.4, and the "actuator" input, which becomes a 6x1 array.

To further illustrate this, in figure A.11 it's represented a flapping wing model where one wing uses a *multi-dof joint* and the other two regular (single) *joints*. In this particular case the movement requires two dof, a rotation around X and a rotation around Z. The added complexity of using single dof *joints* when we need more than one dof is clear from the picture.

With the one single dof *joint* if we would like to implement a, let's say, shoulder with three rotational dof, it would require three *Child* subsystems plus three (single-)actuators, symbolically located at the same physical point. This 3x (*Child* + actuator) can be reduced introducing the universal joint to one single *Child* plus one actuator. Whats more, not only there is a reduction in blocks, but also in configuration parameters.

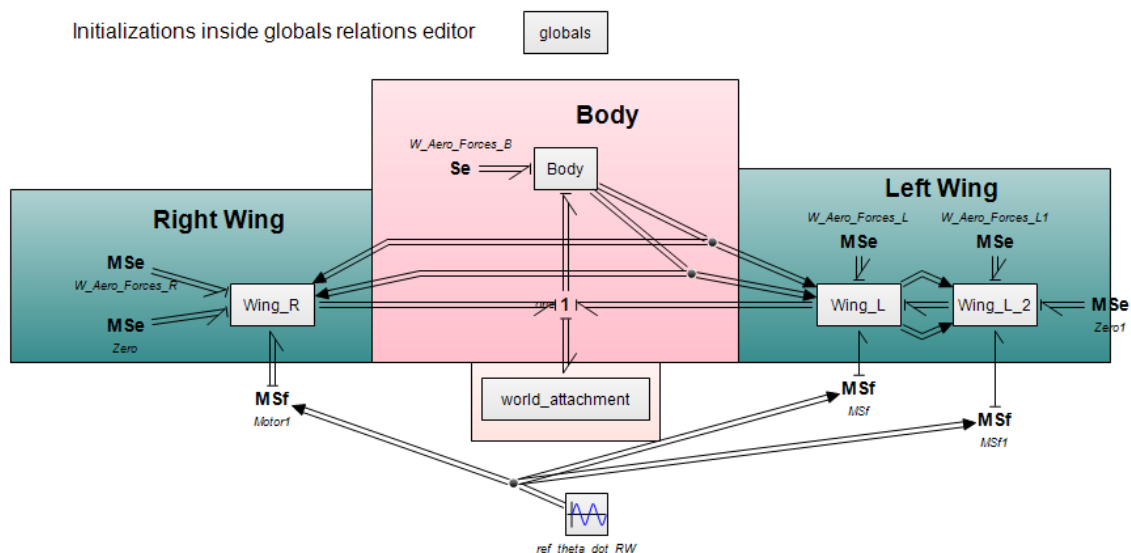


Figure A.11: Flapping bird model comparing joint types. On the "Right Wing" (left side of picture) the joint is an universal joint. On the "Left Wing" (right side of picture) the joints are simple. Using 2 DOF only.

(File:

"\Parent_Child_Modelling\Other Models (tests)\Multidof_efficiency_test\Robird_1dof_vs_3dof.emx").

A.2.3 Building a model with gravity compensation

Gravity compensation can be of great use for simulating different experiments with the models. The air won't be implemented in any of the models, which means that there won't be any external forces thrusting the UAV upwards, but the inertia only. Therefore, if we would like the UAV to "float" to properly verify some properties of the models there are several options. First of all, it is possible to set the gravity term to zero ($g=0$). Second, we can counteract the gravity, in a rough manner, by applying an external force on the main body as an upwards thrust (in body's coordinate frame) equivalent to the weight of the whole UAV. Nonetheless, the body can rotate so its orientation need to be taken into account all the time for properly counteracting

the gravity in the world's coordinate. Adding the "world attachment" (test-bed) block can help a lot to simplify this problems, since it allows the restriction of the UAV's dof.

Finally, if we would like to get rid of the "world attachment", the most accurate way to counteract gravity is by applying an external wrench opposite to the gravity that is computed inside each body as:

$$W_{grv}^{i,i} = G_i^i [Ad_{H_0}^i] A_{grv}^0$$

That uses the updated configuration of the body wrt to the world. All the components of the gravity wrench calculation are accessible because those are either configuration parameters or is information transmitted via signal ports, so that the opposite wrench can computed outside the block and applied through the external wrenches port of each body as:

$$W_{grv}^{i,i} = -G_i^i [Ad_{H_0}^i] A_{grv}^0$$

Notice that the only change compared to the previous equation is the minus sign. Such way of counteracting gravity is demonstrated in figure A.12, where the configuration of the body wrt the world signal is introduced into a modulated source of effort (MSe) that computes the opposite gravity wrench and inserts it back into the body.

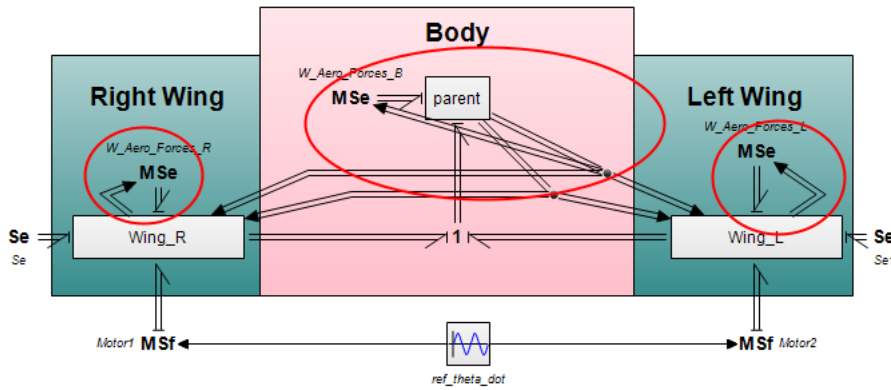


Figure A.12: Model with gravity wrench compensation. The red circles indicate the elements through which the gravity wrench is counteracted.

(File: "\Parent_Child_Modelling\Other Models (tests)\Bird_Basic_Gcomp_test.emx").

A.3 Global Relation Editor (GRE)

The subsystems are meant to be standard for modelling any flying vehicle, thus, the inside of such subsystems should remain untouched. 20-Sim allows to easily configure the blocks/subsystems from the "global relations editor". In this editor, the internal block parameters are related to the user specific parameters. This way, the user will only need to change the name of the parameters with their own to configure the subsystem. All the subsystems are design to require the minimum parameters as possible, while remaining flexible.

The "global relations editor" is very convenient for both the creator of the block and the user, since it allows the user to modify parameters of the block without accessing it. This means, for the creator, that the block can be encrypted and protected, and the interface for the user becomes rather simple at the very same time. The way to access the "global relations editor" is shown in figure A.13. Right click on the block -> Global Relation Editor.

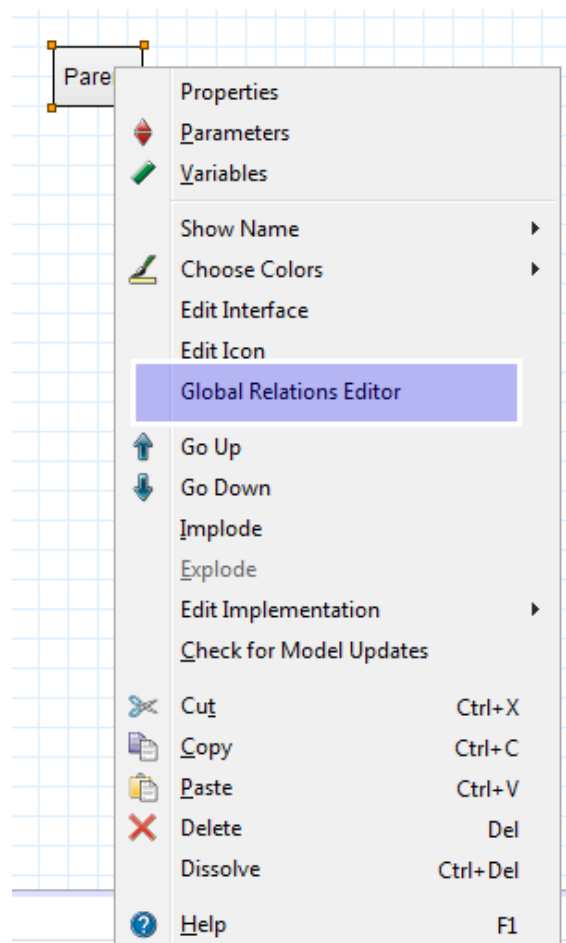


Figure A.13: Global relations editor.

For this parameter tuning method to work the parameters that we want to have access from the "global relation editor" have to be declared as globals inside the underlying blocks and as normal (not global) parameters inside the "global relations editor". This is, all parameters declared in the "global relations editor" are seen as global parameters from the underlying blocks. All blocks can access all parameters that are declared above them in the hierarchy. This is very useful to guarantee modularity without naming conflicts.

A.4 Parent, Child and Joint 20-Sim blocks Implementation

Along this section the implementation of the different elements required for constructing a complete flapping-wing robot model are explained, starting with the parent and continuing with the child and joint structures, that constitute the core of this type of modelling.

The modelling approach is designed in such a way that the user doesn't require to dive into the blocks to tune any parameters, but instead they can do it from the so called, "global relation editor" (GRE), as a way to simplify the work, so parameters won't be a topic until the specific section A.3 dedicated to the GRE.

A.4.1 Parent subsystem

The *parent* subsystem represents the main body, thus it contains the dynamics of that, following equation (4.1). The subsystem's ports, shown in figure A.14, were already explained in section 4.2.

The pseudo-code of this subsystem is presented in algorithm 1.

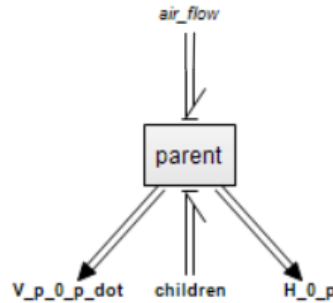


Figure A.14: Parent subsystem

Algorithm 1: Parent pseudo-code.

- 1 $W_{grv} = G \cdot \text{Adjoint}(\text{inverseH}(H_{0_p})) \cdot A_{grv}$;
 - 2 $V_{p_0_p} = \text{int}(G_{\text{inv}} \cdot (\text{transpose}(\text{adjoint}(\text{children.f})) \cdot G \cdot \text{children.f} + W_{grv} + \text{air_flow.e} + \text{children.e}))$;
 - 3 $V_{p_0_p_dot} = \text{ddt}(V_{p_0_p})$;
 - 4 $\text{children.f} = V_{p_0_p}$;
 - 5 $\text{air_flow.f} = V_{p_0_p}$;
 - 6 $H_{0_p} = \text{int}(H_{0_p} \cdot \text{tilde}(V_{p_0_p}), H_{0_p_initial})$;
-

A.4.1.1 GRE Parent

The only parameters the *parent* requires are the three shown in figure A.15 declared under the line "Do not modify". These were also introduced during section 4.2.


```

parameters
//USER SPECIFIC PARAM (MODIFY):
  real global H_0_p_initial_user[4,4]; //initial homogeneous matrix parent
                                     //wrt to inertial frame (Phi0) user defined

  real global mass_user; //mass user defined
  real global I_user; //inertia matrix user defined
  real global g; //gravity user defined
////////
//DO NOT MODIFY:
  real H_0_p_initial[4,4];
  real G[6,6]; //generalized inertia tensor expressed in Psi_b
  real A_grv[6,1];
//////
initialequations
//USER SPECIFIC PARAM (MODIFY right hand side):
  H_0_p_initial = H_0_p_initial_user;
  G = [I_user, 0, 0, 0, 0, 0 ;
       0, I_user, 0, 0, 0, 0 ;
       0, 0, I_user, 0, 0, 0 ;
       0, 0, 0, mass_user, 0, 0 ;
       0, 0, 0, 0, mass_user, 0 ;
       0, 0, 0, 0, 0, mass_user] ;
  A_grv = [ 0; 0; 0; 0; 0; -g];
////////

```

Figure A.15: Global relations editor of the Parent subsystem.

A.4.1.2 Parent parameters declaration (User definitions)

There are many ways to use the *parent*, but it is highly recommended to use a separate code block where all global variables are declared and initial parameters are initialized. This way of proceeding is shown in figure A.16. The initial homogeneous matrix $H_0^P(0)$ is created from the combination of an initial orientation $R_0^P(0)$ and position $O_0^P(0)$. The orientation matrix is created from the combination of rotation matrices, while the position is just the displacement from the world origin. Parameters are used to define the shape of the body, necessary to obtain the inertia of it. As it is explicit in figure A.16, the only parameters needed by other blocks are the homogeneous matrix, the inertia and the mass, all three declared as "global".

```

parameters
// Main Parent Body initial pose
real global H_0_b_initial[4,4] ; //homogenous matrix (hmatrix) Psi body wrt world
real R_0_b_initial[3,3] ;
real R_0_b_X0[3,3] ;
real R_0_b_Y0[3,3] ;
real R_0_b_Z0[3,3] ;
real O_0_b_initial[3,1] ;
//Main body parameters
real global mass_body = 1; // 1 kg
real r_body = 0.2; // 20 cm
real global I_body;
real roll_body_initial; //around x axis [rad]
real pitch_body_initial; //around y axis [rad]
real yaw_body_initial; //around z axis [rad]

```

(a) Parent globals declaration ("parameters").

```

initialequations

//body w.r.t. main frame(0).
roll_body_initial = 0*pi/180;
pitch_body_initial = 0*pi/180;
yaw_body_initial = 0*pi/180;
|
R_0_b_X0 = [1,0,0; 0,cos(roll_body_initial),-sin(roll_body_initial); 0,sin(roll_body_initial),cos(roll_body_initial)];
R_0_b_Y0 = [cos(pitch_body_initial),0,sin(pitch_body_initial); 0,1,0; -sin(pitch_body_initial),0,cos(pitch_body_initial)];
R_0_b_Z0 = [cos(yaw_body_initial),-sin(yaw_body_initial),0; sin(yaw_body_initial),cos(yaw_body_initial),0; 0,0,1];
R_0_b_initial = R_0_b_X0 * R_0_b_Y0 * R_0_b_Z0; //initial orientation of the body
O_0_b_initial = [ 0 ; 0 ; 0 ]; //starting position of the body
H_0_b_initial = homogeneous(R_0_b_initial , O_0_b_initial) ; //initial pos + rot. of body

//Inertias:
I_body = 2/5*mass_body*r_body*r_body ; // assuming an egg is a sphere with moment of inertia 2/5 M R^2

```

(b) Parent globals initialisation ("initialequations").

Figure A.16: Parent globals

A.4.2 Compact Child + Joint

A *child* structure will always require a *joint* to be attached to a parent, thus, it makes sense to put both subsystems together under one single standardized subsystem.

The *child-joint* subsystem comprehends, in plain words, a *child* plus a *joint* together. The code of the blocks is exactly the same as when separate, this is, the unification only means a simplification to the user's eyes, an abstraction. Instead of requiring adding two subsystems there is only need for one *child-joint*, saving time and space when building a model, and, additionally, all user parameters can be modified through one single GRE (explained in section A.3). This *child-joint* subsystems are depicted in figure A.17. Two slightly different *child* subsystem were implemented, a *child*, shown in figure A.17a, and a leaf-*child*, shown in figure A.17b, for the case when the *child* is the last element in the hierarchy (leaf). The *child-joint* subsystem will be simply referred to as **Child** from now on for simplicity.

Finally, what the user is offered its just the two compact options shown in figures A.17c and A.17d, named Child-Parent and Child in the figures (Child and leaf-Child respectively). At the end all flapping-UAV systems should be modelable using only the *Parent* and *Child* subsystems (plus the actuators and external elements).

A.4.2.1 Child subsystem

The *child* subsystem is used to represent any body different from the main body, which dynamic behaviour is ruled by equation (4.3), as explained previously on section 4.3.

The *child* subsystem's ports were already also explained in 4.3. Additionally, a slightly different subsystem was implemented for the case when the child is the last element in the hierarchy (leaf), with the only difference that the "children" power port is removed, as it wouldn't be

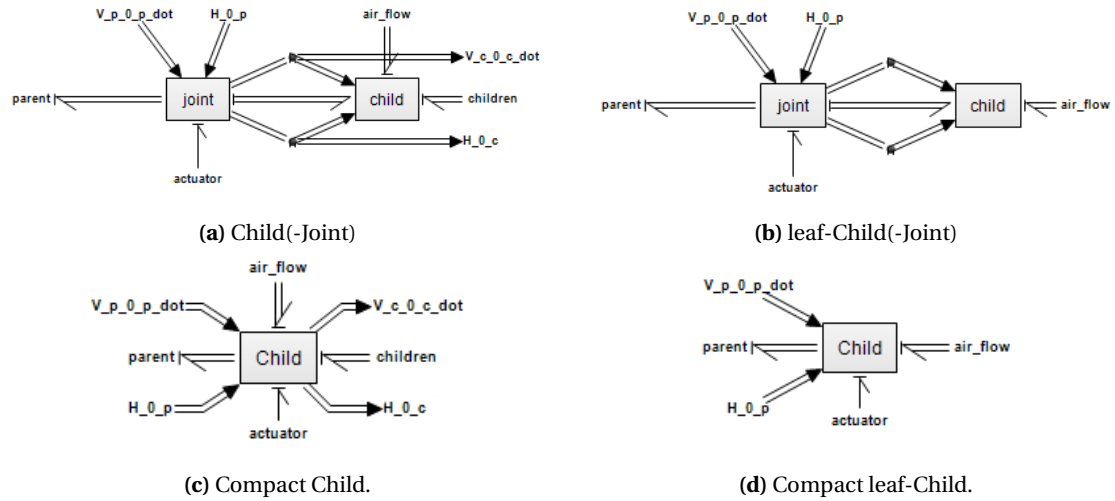


Figure A.17: Joint-Child subsystem models.

used. The difference in the code between the *child* and the leaf-*child* is minimal, as it can be seen by comparing algorithms 2 and 3.

It would be possible to build any model only using the *child-parent* subsystem with the extra use of one 0-effort source. It is up to the user what they prefer to use, but both *child* and *child-parent* blocks are created for ease of construction of the models.

Algorithm 2: Child pseudo-code.

- 1 $W_{grv} = G \cdot \text{Adjoint}(\text{inverseH}(H_{0_c})) \cdot A_{grv}$;
 - 2 $\text{joint.e} = G \cdot V_{c_0_c_dot} - \text{transpose}(\text{adjoint}(\text{joint.f})) \cdot G \cdot \text{joint.f} - W_{grv} - \text{air_flow.e} - \text{child.e}$;
 - 3 $\text{air_flow.f} = \text{joint.f}$;
 - 4 $\text{child.f} = \text{joint.f}$;
-

Algorithm 3: leaf-Child pseudo-code.

- 1 $W_{grv} = G \cdot \text{Adjoint}(\text{inverseH}(H_{0_c})) \cdot A_{grv}$;
 - 2 $\text{joint.e} = G \cdot V_{c_0_c_dot} - \text{transpose}(\text{adjoint}(\text{joint.f})) \cdot G \cdot \text{joint.f} - W_{grv} - \text{air_flow.e}$;
 - 3 $\text{air_flow.f} = \text{joint.f}$;
-

A.4.2.2 Joint subsystem

The *joint* is the element that determines the relation between *parent* and *child* through the *actuator*, that inserts a relative velocity term. All the ports of the *joint* subsystem were introduced during section 4.4.

The block first processes the joint's relative velocity ("actuator" input, ω or v) to get the joint's relative displacement ($\int \omega$ or $\int v$) and acceleration ($\frac{d}{dt} \omega$ or $\frac{d}{dt} v$). Next, that displacement is used to obtain the homogeneous matrix that relates the configuration of the *child* wrt to the *father* H_p^c . After this, H_p^c is used to form the "joint matrix". Finally the *child* configuration H_c^0 is computed, but more importantly, the rate of change of the *child's* twist $\dot{V}_c^{c,0}$, explained in equation (4.9), using the joint's acceleration $\ddot{\theta}$.

There are three types of joints that can be defined with this same block, a revolute joint, a screw joint and a prismatic joint. The differentiation between them is done by the user indirectly, as it

is intrinsic in the unit twist, that specifies the dof between bodies and which is inserted by the user via the GRE as it will be explained in section A.4.2.3. If the unit twist has both, a linear \hat{v} and a rotational $\hat{\omega}$ velocity components, then the joint can be either a screw joint or a revolute joint, but if there is only a linear velocity \hat{v} component, it will be a prismatic joint.

$$\begin{cases} \text{Screw or Revolute joint} \rightarrow S_c^{p,p} = (\hat{\omega}, \hat{\omega} \wedge \hat{v})^T \\ \text{Prismatic joint} \rightarrow S_c^{p,p} = (0, 0, 0, \hat{v})^T \end{cases}$$

The joint is constructed to automatically detect the type of joint, so if the unit twist has a $\hat{\omega}$ component the equation used will be eq. (3.7). On the other hand, if there is only the \hat{v} component eq. (3.9) will be used instead. The pseudo-code is presented in algorithm 4.

If we would like to fix the *child* to the *parent* then a source of 0-flow ($Sf = 0$) should be connected to the "actuator" port (see figure ??/A.17a) in order to constrain any movement. But if on the contrary, the *child* moves with respect to the *parent* this movement is also controlled through the *actuator's* port by means of active (i.e. motor) or passive (i.e. spring) actuators.

Algorithm 4: Joint pseudo-code.

```

1  q = int(actuator.f, q0);
2  q̇ = ddt(actuator.f, q̇0);
3  if ω = 0 then
4    R = eye(3);
5    O = v̂q;
6  else
7    R = (eye(3)+skew(ω̂) · sin(q)+skew(ω̂)2(1-cos(q)));
8    O = (eye(3) - R) · r + λω̂;
9  end
10 H_p_c = homogeneous(R,O) · H_p_c_initial;
11 AdH = Adjoint(inverseH(H_p_c));
12 parent.e = - transpose(AdH) · child.e;
13 child.f = AdH · (parent.f + S_p_p_c · actuator.f);
14 actuator.e = transpose(S_p_p_c) · transpose(AdH) · child.e;
15 V_c_0_c_dot = adjoint(child.f) · S_c_p_c · actuator.f + AdH · V_p_0_p_dot + S_c_p_c · q̇;
16 H_0_c = H_0_p · H_p_c;

```

A.4.2.3 GRE compact Child

The *Child* subsystem require some configuration from the user. For this, all it is needed is adding the user parameters in the "global relations editor", in a very similar way as explained just before for the *parent* structure. Nevertheless, these parameters are configured together with the *joint* parameters, since a *Child* will always require a *joint* to be connected to the system. Figure A.18 shows the exact parameters required for the subsystem to work, which were already mentioned separately for *child* and *joint* in sections 4.3 and 4.4 respectively.

A.4.2.4 Child parameters declaration (User definitions)

The *Child* parameter declaration is very similar to the *parent's*, but with the addition of the unit twists S_p^c , that define the degree of freedom of the *Child* with respect to the *parent*. Also, it is important to notice that the *Child's* initial homogeneous matrix $H_p^c(0)$ is defined relative to the *parent*, not to the world. For the computation of the unit twists some dummy variables are used. This is considered the easiest way to compute them. All this is visualized in figure A.19.

```

parameters
//USER SPECIFIC PARAM (MODIFY):
//Unit twist & initial homogeneous matrix
real global S_p_p_c_user[6,1]; //Unit twist child wrt parent user defined
real global H_p_c_initial_user[4,4]; //initial homogeneous matrix child wrt parent user defined
real global mass_user; //mass user defined
real global I_user; //inertia user defined
//real global G_user[6,6]; //inertia matrix user defined
//real global g; //gravity user defined //g_n is predefined from 20-sim
/////
//DO NOT MODIFY:
real S_p_p_c[6,1];
real H_p_c_initial[4,4];
real G[6,6];
real A_grv[6,1];
real q_init; //initial state, default = 0;
real q_dot_dot_init; //initial state, default = 0;
/////
initialequations
//USER SPECIFIC PARAM (MODIFY right hand side):
S_p_p_c = S_p_p_c_user;
H_p_c_initial = H_p_c_initial_user;
G = [I_user, 0, 0, 0, 0, 0 ;
     0, I_user, 0, 0, 0, 0 ;
     0, 0, I_user, 0, 0, 0 ;
     0, 0, 0, mass_user, 0, 0 ;
     0, 0, 0, 0, mass_user, 0 ;
     0, 0, 0, 0, 0, mass_user] ;
//G = G_user;
A_grv = [ 0; 0; 0; 0; 0; 0; -g_n ] ; //g_n is predefined from 20-sim

//q_init = ...; //default = 0;
//q_dot_dot_init = ...; //default = 0;
/////

```

Figure A.18: Global relations editor of the Child-Joint subsystem.

A.4.3 Child with multi-dof joint

When using a *multi-dof joint* the only visible difference compared to the simple *joint* is the actuator, that instead of being a single power port input it is an array power port input. This implies that all elements of the array different from 0 will actuate a degree of freedom, including three rotations and three translations ($[\omega_x, \omega_y, \omega_z, v_x, v_y, v_z]$). The theory behind this was introduced in section 4.4.

The multi-dof joint not only means a reduction in blocks, but also in configuration parameters. The most noticeable one, the unit twists, don't require the user to pay attention to them anymore, going from the need to code several 6x1 unit twists to requiring just the position of the joint wrt the parent, r_{joint}^P . The code implementation of this block is exemplified in algorithm 5.

A.4.3.1 GRE Child with multi-dof Joint

For the case of a *Child* with a multi-DOF joint the configuration parameters are mostly the same as for the regular *Child* subsystem with exception of the unit twist. The unit twist disappears, as now the six possible unit twists (three rotations and three translations) are already considered inside the block's code.

```

//wing
real global H_b_w_initial[4,4] ; //Hmatrix of Psi_w w.r.t. Psi_b
real R_b_w_initial[3,3] ;
real R_b_w_X0[3,3] ;
real R_b_w_Y0[3,3] ;
real R_b_w_Z0[3,3] ;
real O_b_w_initial[3,1] ;
//wings parameters
real global mass_wing = 0.02 {kg};
real length_wing = 1.0 {m};
real global I_wing;
real init_Xrot_w;
real init_Yrot_w;
real init_Zrot_w;
//unit twist
real global S_b_b_w[6,1];
real a[3,1], b[3,1], c[3,1]; // dummy variables

(a) Child globals declaration ("parameters").

//wing left config w.r.t main body
init_Xrot_w = 30*pi/180;
init_Yrot_w = 0*pi/180;
init_Zrot_w = 0*pi/180;
R_b_w_X0 = [1, 0, 0; 0, cos(init_Xrot_w), -sin(init_Xrot_w); 0, sin(init_Xrot_w), cos(init_Xrot_w)];
R_b_w_Y0 = [cos(init_Yrot_w),0,sin(init_Yrot_w); 0,1,0; -sin(init_Yrot_w),0,cos(init_Yrot_w)];
R_b_w_Z0 = [cos(init_Zrot_w),-sin(init_Zrot_w),0; sin(init_Zrot_w),cos(init_Zrot_w),0; 0,0,1];
R_b_w_initial = R_b_w_X0*R_b_w_Y0*R_b_w_Z0; //initial rotation matrix of the wing
O_b_w_initial = [ 0 ; r_body+(length_wing/2)*cos(init_Xrot_w) ; (length_wing/2)*sin(init_Xrot_w)]; //starting position
H_b_w_initial = homogeneous(R_b_w_initial , O_b_w_initial) ; //initial pos + rot. of wing

//Unit twists
a=[1;0;0]; //turn around x
b=[ 0 ; r_body ; 0];
c=cross(b,a); //left wing
S_b_b_w = [a[1];a[2];a[3];c[1];c[2];c[3]];

//Inertias:
I_wing = 1/12*mass_wing*length_wing*length_wing ;

(b) Child globals initialisation ("initialequations").

```

Figure A.19: Child globals.

The previously required unit twist, that already included the distance from the parent to the joint in its definition, is substituted in the parameters by simply that distance from the parent to the joint as already mentioned in section 4.4. See figure A.20.

A.4.3.2 Child with multi-dof joint parameters declaration (User definitions)

When using a multi-dof joint the parameters are tuned very similarly to the normal *Child* but removing the unit twists S_p^c . These are changed for an array containing the relative positions of the joint wrt the *parent*. This array declaration is shown in figure A.21. The *Child's* initial homogeneous matrix $H_p^c(0)$ is still defined relative to the *parent*, not to the world.

Initialitation of the six unit twists is shown in figure A.22. So that the movement is not constrained by the unit twists, but by the actuators input array. Actuating a degree of freedom or not will depend solely of the actuation, so if the input to a certain degree of freedom is zero, then that dof will be constraint.

Figure A.23 shows the obtaining of the updated position and orientation from the unit twists and the inputs, q ($\int \dot{q}$) and \dot{q} ("actuator.f"). The theory behind this operations is explained in section 4.4.

Algorithm 5: multi-DOF Joint pseudo-code.

```

1  $q = \text{int}(\text{actuator.f}, q_0)$ ;
2  $\ddot{q} = \text{ddt}(\text{actuator.f}, \dot{q}_0)$ ;
3 for  $i$  in 3 do
4    $R\_rot[i] = (\text{eye}(3) + \text{skew}(\hat{\omega}) \cdot \sin(q[i]) + \text{skew}(\hat{\omega})^2 (1 - \cos(q[i])))$ ;
5    $O\_rot[i] = (\text{eye}(3) - R\_rot[i]) \cdot r + \lambda \hat{\omega}$ ;
6    $H\_p\_c\_rot[i] = \text{homogeneous}(R\_rot[i], O\_rot[i])$ ;
7 end
8  $H\_p\_c\_trans = \text{homogeneous}(\text{eye}(3), q[4:6])$ ;
9  $H\_p\_c = H\_p\_c\_trans \cdot H\_p\_c\_rot[3] \cdot H\_p\_c\_rot[2] \cdot H\_p\_c\_rot[1] \cdot H\_p\_c\_initial$ ;
10  $\text{AdH} = \text{Adjoint}(\text{inverseH}(H\_p\_c))$ ;
11  $\text{parent.e} = - \text{transpose}(\text{AdH}) \cdot \text{child.e}$ ;
12  $\text{child.f} = \text{AdH} \cdot (\text{parent.f} + S\_p\_p\_c \cdot \text{actuator.f})$ ;
13  $\text{actuator.e} = \text{transpose}(S\_p\_p\_c) \cdot \text{transpose}(\text{AdH}) \cdot \text{child.e}$ ;
14  $V\_c\_0\_c\_dot = \text{adjoint}(\text{child.f}) \cdot S\_c\_p\_c \cdot \text{actuator.f} + \text{AdH} \cdot V\_p\_0\_p\_dot + S\_c\_p\_c \cdot \ddot{q}$ ;
15  $H\_0\_c = H\_0\_p H\_p\_c$ ;

```

```

20-sim Global Relations Editor
Specify the global relations for submodel: child_DOF

parameters
//USER SPECIFIC PARAM (MODIFY):
  real global H_p_c_initial_user[4,4]; //initial homogeneous matrix child wrt parent user defined
  real global r_joint_user[3,1]; //position joint user defined
  real global mass_user; //mass user defined
  real global I_user; //inertia user defined
  //real global G_user[6,6]; //inertia matrix user defined //g_n is predefined from 20-sim
  real global g; //gravity user defined
////////
//DO NOT MODIFY:
  real r_joint[3];
  real H_p_c_initial[4,4];
  real G[6,6];
  real A_grv[6,1];
  real q_init[6,1]; //initial state, default all = 0;
  real q_dot_dot_init[6,1]; //initial state, default all = 0;
/////
initialequations
//USER SPECIFIC PARAM (MODIFY right hand side):
  r_joint = r_joint_user;
  H_p_c_initial = H_p_c_initial_user;
  G = [I_user, 0, 0, 0, 0, 0 ;
        0, I_user, 0, 0, 0, 0 ;
        0, 0, I_user, 0, 0, 0 ;
        0, 0, 0, mass_user, 0, 0 ;
        0, 0, 0, 0, mass_user, 0 ;
        0, 0, 0, 0, 0, mass_user] ;

  //G = G_user;
  A_grv = [ 0; 0; 0; 0; 0; -g_n] ; //g_n is predefined from 20-sim

  //q_init = ...; //default = 0s;
  //q_dot_dot_init = ...; //default = 0s;
/////

```

Figure A.20: Global relation editor of the Child with multi-dof joint subsystem.

```

//wing
real global H b_wL_initial[4,4] ; //Hmatrix of Psi_wL w.r.t. Psi_b
real global r_joint_wL_b[3,1]; //joint position
real R_b_wL_initial[3,3] ;
real R_b_wL_X0[3,3] ;
real R_b_wL_Y0[3,3] ;
real R_b_wL_Z0[3,3] ;
real O_b_wL_initial[3,1] ; |
//wing parameters
real global mass_wing = 0.1 {kg}; //0.018
real global length_wing = 1.0 {m};
real init_Xrot_wL;
real init_Yrot_wL;
real init_Zrot_wL;

```

(a) Child with multi-DOF joint globals declaration ("parameters").

```

//wing config w.r.t main body
init_Xrot_wL = 45*pi/180;
init_Yrot_wL = 0*pi/180;
init_Zrot_wL = 0*pi/180;
R_b_wL_X0 = [1, 0, 0; 0, cos(init_Xrot_wL), -sin(init_Xrot_wL); 0, sin(init_Xrot_wL), cos(init_Xrot_wL)];
R_b_wL_Y0 = [cos(init_Yrot_wL), 0, sin(init_Yrot_wL); 0, 1, 0; -sin(init_Yrot_wL), 0, cos(init_Yrot_wL)];
R_b_wL_Z0 = [cos(init_Zrot_wL), -sin(init_Zrot_wL), 0; sin(init_Zrot_wL), cos(init_Zrot_wL), 0; 0, 0, 1];
R_b_wL_initial = R_b_wL_X0*R_b_wL_Y0*R_b_wL_Z0; //initial rotation matrix of the wing
O_b_wL_initial = [ -(length_wing/2)*sin(init_Zrot_wL) ; r_body+(length_wing/2)*cos(init_Xrot_wL)*cos(init_Zrot_wL) ; (length_wing/
H b_wL_initial = homogeneous(R_b_wL_initial , O_b_wL_initial) ; //initial pos + rot. of wing
r_joint_wL_b = [0;r_body;0];

```

(b) Child with multi-dof joint globals initialisation ("initialequations").

Figure A.21: Child with multi-DOF joint.

```

initialequations
//Unit twists
aX=cross(r_joint, [1;0;0]);
aY=cross(r_joint, [0;1;0]);
aZ=cross(r_joint, [0;0;1]);
S_p_p_c = [1,0,0,0,0,0;
           0,1,0,0,0,0;
           0,0,1,0,0,0;
           aX[1],aY[1],aZ[1],1,0,0;
           aX[2],aY[2],aZ[2],0,1,0;
           aX[3],aY[3],aZ[3],0,0,1];
S_p_p_c_T = transpose(S_p_p_c);
S_c_p_c = Adjoint(inverseH(H_p_c_initial))*S_p_p_c;
skew_w_p_p_c_X = skew(S_p_p_c[1:3,1]);
skew_w_p_p_c_Y = skew(S_p_p_c[1:3,2]);
skew_w_p_p_c_Z = skew(S_p_p_c[1:3,3]);
skew_w_p_p_c_X_squared = skew_w_p_p_c_X^2;
skew_w_p_p_c_Y_squared = skew_w_p_p_c_Y^2;
skew_w_p_p_c_Z_squared = skew_w_p_p_c_Z^2;

```

Figure A.22: Unit twists initialization with multi-dof joint.

A.4.4 World / Test-bed modelling

This is an element that imitates a test-bed platform. It enables/disables the 6 degrees of freedom of the whole model by simply specifying with 'true' or 'false' if a degree is free or constrained.

The subsystem is very simple, consisting on a resistance-capacitor combination, as shown in figure A.25. The code inside the R and C elements is modified though, in order to allow the user to enable/disable the degrees of freedom at will. The pseudo-code is shown in algorithm 6.

Figure A.24 shows an example of the declaration of the world/test bench constraints vector in 20-Sim. Here the rotation around X is freed together with the movement in the Z axis. The vector positions represent the rotation around X, around Y, around Z, movement along the X


```

//Pure Rotation: 1. I+ wsin(q) + w^2(1-cos(q)) / 2. (I-R)*r
R_p_c_Xrot = eye(3)+skew_w_p_p_c_X*sin(q[1])+skew_w_p_p_c_X_squared*(1-cos(q[1])) ;
o_p_c_Xrot = (eye(3) - R_p_c_Xrot)*r_joint;
H_p_c_Xrot = homogeneous(R_p_c_Xrot,o_p_c_Xrot);

R_p_c_Yrot = eye(3)+skew_w_p_p_c_Y*sin(q[2])+skew_w_p_p_c_Y_squared*(1-cos(q[2])) ;
o_p_c_Yrot = (eye(3) - R_p_c_Yrot)*r_joint;
H_p_c_Yrot = homogeneous(R_p_c_Yrot,o_p_c_Yrot);

R_p_c_Zrot = eye(3)+skew_w_p_p_c_Z*sin(q[3])+skew_w_p_p_c_Z_squared*(1-cos(q[3])) ;
o_p_c_Zrot = (eye(3) - R_p_c_Zrot)*r_joint;
H_p_c_Zrot = homogeneous(R_p_c_Zrot,o_p_c_Zrot);

//Pure translation: 2. q_dot
H_p_c_translation = homogeneous(eye(3),q[4:6]);

```

Figure A.23: Position and orientation (configuration) computation with multi-dof joint.

axis, Y axis and Z axis respectively. The modification of the constraints is done via the GRE, as explained next.

```

//Test bench - Body attachment
boolean global tb_constraints[6] = [true,false,false,false,false,true]; //unconstraint(free)=TRUE, constraint=FALSE.

```

Figure A.24: World attachment constraints vector.

Algorithm 6: World Attachment pseudo-code. TRUE=Free (enable dof), FALSE=Constrained (disable dof).

```

1 for i = 1 to 6 do
2   if constraints[i]==TRUE then
3     r[i]=r_free;
4     c[i]=c_free;
5   else
6     r[i]=r_cstr;
7     c[i]=c_cstr;
8   end
9 end
10 R = diag(r);
11 C = diag(c);

```

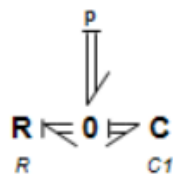
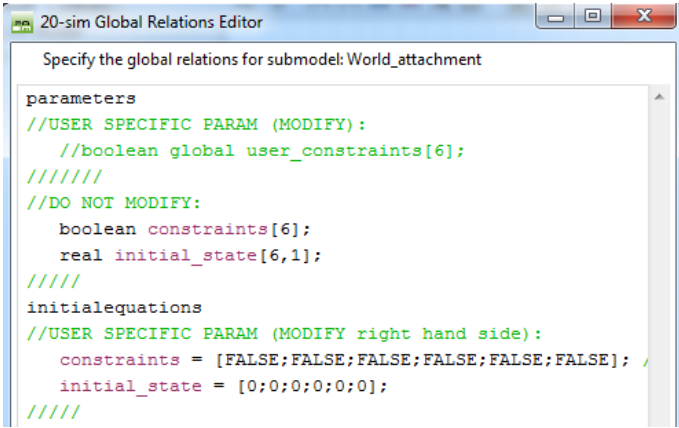


Figure A.25: World attachment model.

A.4.4.1 GRE World Attachment

Finally, the *wold attachment* subsystem also employs the "global relations editor" to indicate the degrees of freedom of the model wrt the world through the mentioned constraints array shown in figure A.24. Its particular "global relations editor" would look like in figure A.26 with the constraints vector directly declared in it. An "initial state" vector is included in case the initial frame of the body wrt the world differs, but by default it should be all 0's.



```
20-sim Global Relations Editor
Specify the global relations for submodel: World_attachment

parameters
//USER SPECIFIC PARAM (MODIFY):
//boolean global user_constraints[6];
//////
//DO NOT MODIFY:
boolean constraints[6];
real initial_state[6,1];
//////
initialequations
//USER SPECIFIC PARAM (MODIFY right hand side):
constraints = [FALSE;FALSE;FALSE;FALSE;FALSE;FALSE];
initial_state = [0;0;0;0;0;0];
//////
```

Figure A.26: Global relation editor of the World Attachment subsystem.

A.5 The Numerical Differentiation and the Algebraic Loops Problem

Numerical differentiation tend to be computationally very expensive, so it is something to be avoided as much as possible, since it can make the method impracticable. The presented approach require the differentiation of the twist, $\dot{T}_c^{c,0}$, which can be an inconvenient. It is possible though to get the rate of change of the child's twist mathematically with eq. 4.9:

$$\dot{T}_c^{c,0} = [ad_{T_c^{c,0}}] S_c^{c,p} \dot{\theta}_c + [Ad_{H_p^c}] \dot{T}_p^{p,0} + S_c^{c,p} \ddot{\theta}_c$$

Nevertheless, this solves the numerical differentiation in the child, but requires the rate of change of the parent's twist $\dot{T}_p^{p,0}$, which can be obtained from eq. 4.1 as follows:

$$\dot{T}_p^{p,0} = (G^p)^{-1} ([ad_{T_p^{p,0}}]^T G^p T_p^{p,0} + W_{grv} + W_{air_flow} + W_{children})$$

The issue with implementing this equation in 20-Sim is that it uses $T_p^{p,0}$ to compute $\dot{T}_p^{p,0}$, generating an algebraic loop, which is a huge problem for performance.

Since 20-Sim's behaviour with these loops is a bit unpredictable, the program has to be "cheated". The first code that was tested, shown in algorithm 7, did solve the algebraic loop in the parent and looks the most intuitive, so it was considered good at first. Nonetheless, it did provoke all the children underneath to suffer algebraic loops instead, which is worse than having only the one of the parent.

Algorithm 7: Parent initial pseudo-code.

- 1 `V_p_0_p_dot = G_inv · (transpose(adjoint(V_p_0_p)) · G · V_p_0_p + W_grv + air_flow.e + children.e);`
 - 2 `V_p_0_p = int(V_p_0_p_dot);`
-

The second version, shown in algorithm 8, reverted the problem. It did solve the algebraic loop in the children, improving the efficiency of the models enormously, but moved the algebraic loop to, only, the parent. This is technically a much better solution, since there is only one parent per model, thus only one algebraic loop.

Algorithm 8: Parent improved pseudo-code.

- 1 `V_p_0_p = int(G_inv · (transpose(adjoint(children.f)) · G · children.f + W_grv + air_flow.e + children.e));`
 - 2 `V_p_0_p_dot = G_inv · (transpose(adjoint(children.f)) · G · children.f + W_grv + air_flow.e + children.e);`
-

Finally, the elimination of absolutely all algebraic loops was achieved with the algorithm 9, increasing again the efficiency. The solution is simply using the derivative function from 20-Sim, that seems to be better for 20-Sim's engine even though it looks absolutely counter-intuitive.

Algorithm 9: Parent final pseudo-code.

- 1 `V_p_0_p = int(G_inv · (transpose(adjoint(children.f)) · G · children.f + W_grv + air_flow.e + children.e));`
 - 2 `V_p_0_p_dot = ddt(V_p_0_p);`
-

Not only that, but using `children.f` or `Vp0p` changes the result!!

It is noticed that if the differentiation of $V_{p_0_p}$ is written before $V_{p_0_p} = \text{inf}()$, as in algorithm 10, then $V_{p_0_p_dot}$ remains all 0s for the whole simulation, which is erroneous. But if the $V_{p_0_p_dot}$ is before $V_{p_0_p}$ with children.f inside, as in algorithm 11, then it works just perfect.

Algorithm 10: Parent pseudo-code with $V_{p_0_p_dot}$ always 0s, so NOT working.

- 1 $V_{p_0_p_dot} = \text{ddt}(V_{p_0_p});$
 - 2 $V_{p_0_p} = \text{int}(G_{\text{inv}} \cdot (\text{transpose}(\text{adjoint}(\text{children.f})) \cdot G \cdot \text{children.f} + W_{\text{grv}} + \text{air_flow.e} + \text{children.e}));$
-

Algorithm 11: Parent pseudo-code with $V_{p_0_p_dot}$ before computation of $V_{p_0_p}$, but as a function of children.f and working perfectly. Equivalent to algorithm 9.

- 1 $V_{p_0_p_dot} = \text{ddt}(\text{children.f});$
 - 2 $V_{p_0_p} = \text{int}(G_{\text{inv}} \cdot (\text{transpose}(\text{adjoint}(\text{children.f})) \cdot G \cdot \text{children.f} + W_{\text{grv}} + \text{air_flow.e} + \text{children.e}));$
-

A.6 Creation of the sub-models library

In 20-Sim a library is merely a folder/directory with models and/or sub-models. Models are meant as examples or base projects, while sub-models are elements that can be added to a model but don't have a function by themselves alone. In this case the focus is put on the creation of sub-models, since the parent, the child and the world are sub-models.

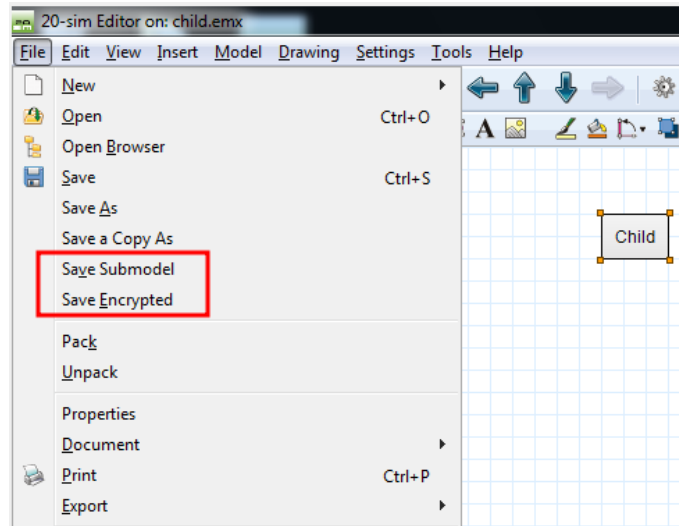


Figure A.27: Save sub-model as sub-model or as encrypted sub-model.

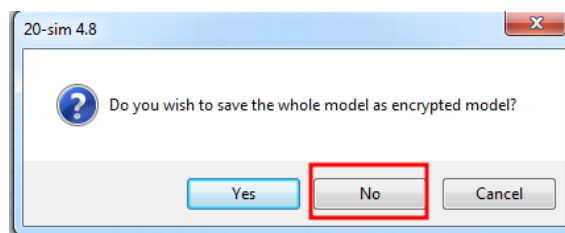


Figure A.28: Save sub-model as encrypted.

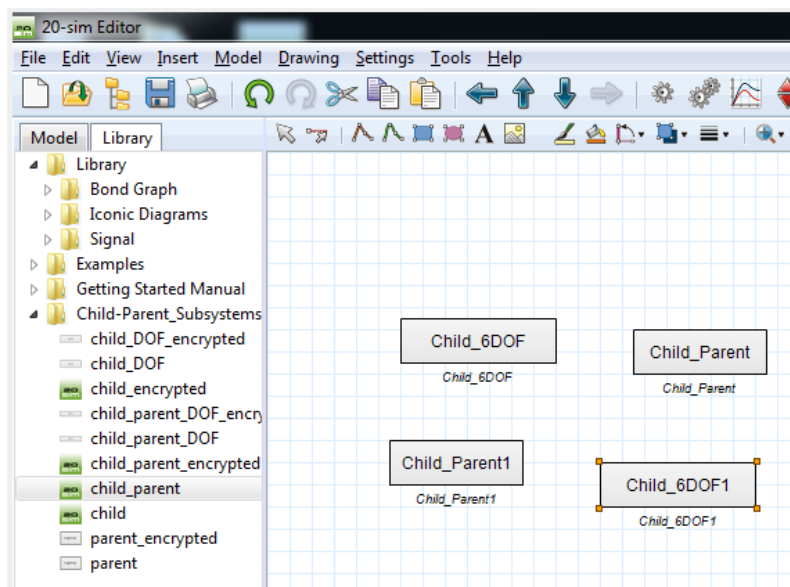


Figure A.29: 2.

B Appendix 2: Extra models for testing, validation and verification.

B.1 Pendulum model

The pendulum model can be used for validation and verification of the modelling and simulation framework implementation. On the one hand it is a bad example because of its non-linear behaviour, but at the same time it is a very good example because it also requires the inertia tensor to be properly specified in order to provide the expected results.

The pendulum non-linear dynamic equation is:

$$\ddot{\theta} + \frac{g}{L} \sin\theta = 0$$

Where θ is the angle, g is the gravity and L the length of the pendulum. That linearized around its equilibrium point, this is, considering $\sin\theta = \theta$ for very small angle variations around the equilibrium, results in:

$$\ddot{\theta} + \frac{g}{L} \theta = 0$$

From which we can obtain the natural frequency, w_n as $w_n = \sqrt{\frac{g}{L}}$. The period relates to the natural frequency as:

$$T = \frac{2\pi}{w_n} = 2\pi\sqrt{\frac{L}{g}}$$

Nevertheless this is the exact period at the equilibrium point only. The broader the angle of oscillation, this is, the further from the equilibrium point the further from that equation the period will be.

For this test the model presented in figure B.1 is built. It simply consists on a body with a wing that will act as pendulum. The body is completely constrained by the "world attachment". The wing is actuated by a free-joint, consisting on a very low resistance ($R = 1e^{-6}$) a very low inductance ($I = 1e^{-10}$) and a very compliant capacitor ($C = 1e^{12}$), so the wing will fall free due to gravity and the joint actuation won't exert almost any resistance.

The length of the pendulum in this case is the distance from the joint to the CG of the wing, that is located at the middle. This means, for gravity $g = 9.80665ms^{-2}$ and $L = 0.5m$ that the theoretical period of oscillation of the wing should be:

$$T = 2\pi\sqrt{\frac{L}{g}} = 2\pi\sqrt{\frac{0.5}{9.80665}} = 1.4187s$$

In order to obtain a similar result the oscillation must be very reduced, so the wing is let to oscillate approximately an amplitude of 0.1 radians or 6 degrees, as shown in the graph B.2, where the oscillation angle is shown (displaced, so amplitude is half). The obtained period is also shown in the same figure, is:

$$T = 1.4122s$$

Which is very close, but smaller, which should be impossible, but it is probably because of the time steps of the simulator or a side-effect of the actuator. The result in fact verifies the dynamics of the model.

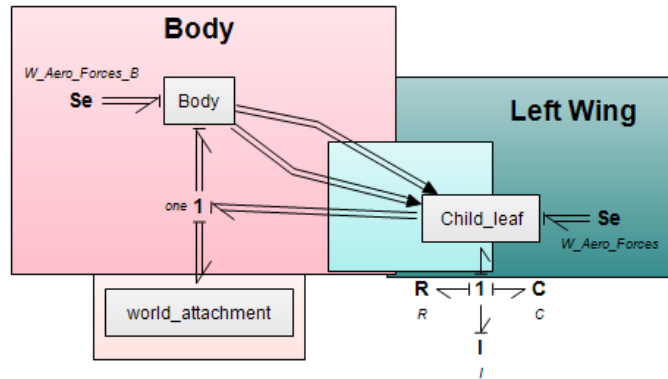


Figure B.1: Pendulum (wing pendulum) 20-sim model.

(File: "\Parent_Child_Modelling\Other Models (tests)\Pendulum\Pendulum_wing.emx").

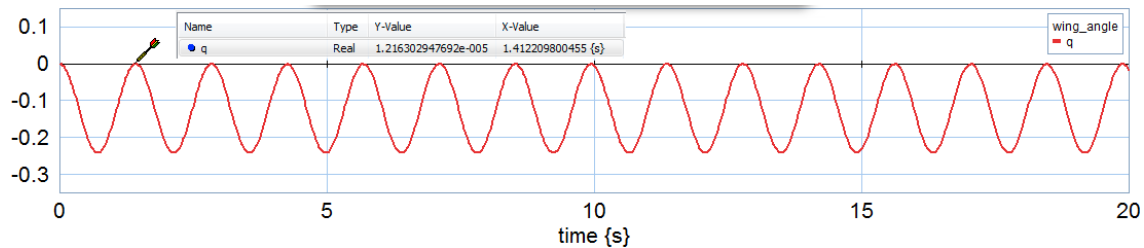


Figure B.2: Pendulum oscillation graph. Period (T) of oscillation shown on graph.

(File: "\Parent_Child_Modelling\Other Models (tests)\Pendulum\Pendulum_wing.emx").

Additionally, for obtaining this result the inertia tensor required special attention. The inertia need to be properly defined as a rod rotating around an extreme, as:

$$I = \frac{1}{3} m_{wing} L_{wing}^2$$

Where m_{wing} is the mass of the wing and $L_{wing} = 2L$ is twice the length of the pendulum. And the inertia tensor, as the inertia in the case of the pendulum affects only in the Y axis of the wing, it would finally look like:

$$G = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & I & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & m_{wing} & 0 & 0 \\ 0 & 0 & 0 & 0 & m_{wing} & 0 \\ 0 & 0 & 0 & 0 & 0 & m_{wing} \end{pmatrix}$$

B.2 Transformer Aircraft (Flapping + Prismatic) with gravity compensation

The model, presented in figure B.3, was built as an alternative to the model in section 5.4 to test if removing the "world attachment" would help reduce the drifting error in the energy balance and the position. Nevertheless, the position and the energy still accumulates error, only in a

different manner, as shown in figures B.4 for the position and B.5 for the energy balance. There is no pattern found in the error drift, so it will be considered numerical error.

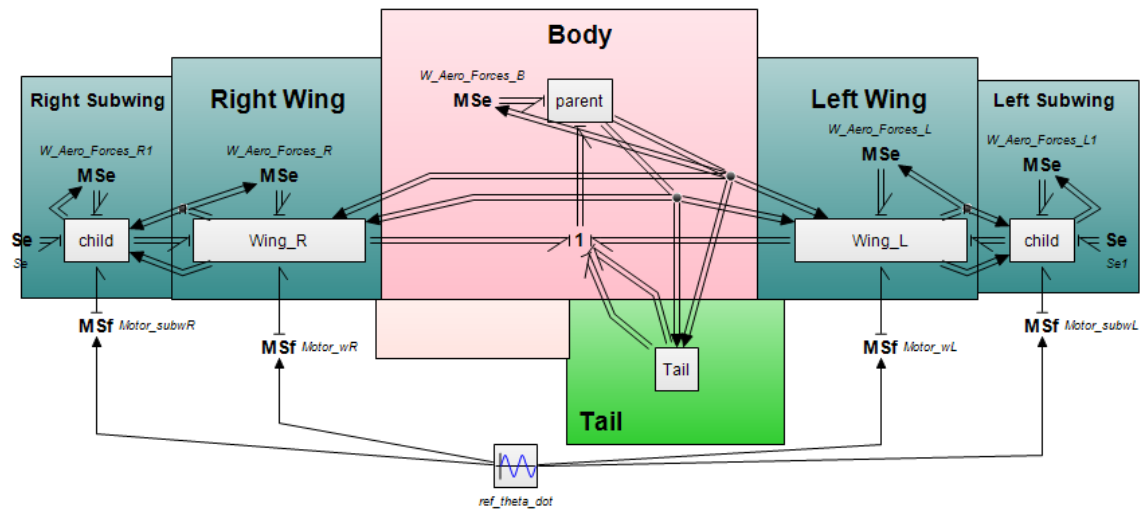


Figure B.3: Flapping wing and prismatic subwing 20-sim model with gravity compensation. (File: "\Parent_Child_Modelling\Literature_Models\Bird_prismatic\Bird_prismatic_Gcomp.emx").

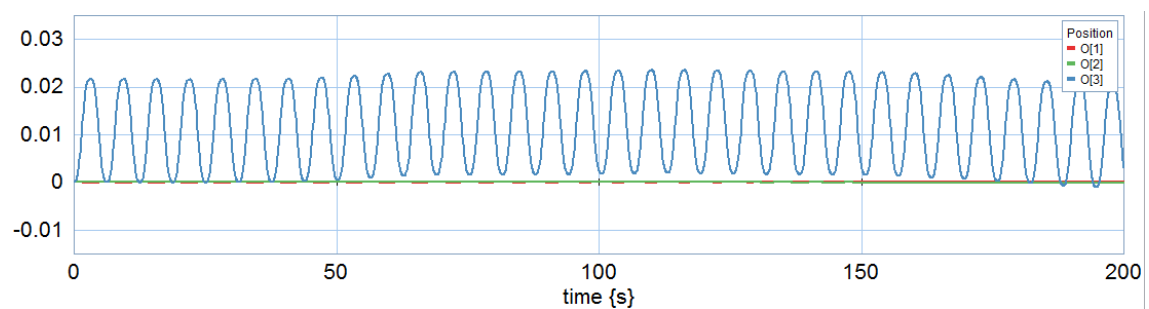


Figure B.4: Flapping wing and prismatic subwing model with gravity compensation position graph. (File: "\Parent_Child_Modelling\Literature_Models\Bird_prismatic\Bird_prismatic_Gcomp.emx").

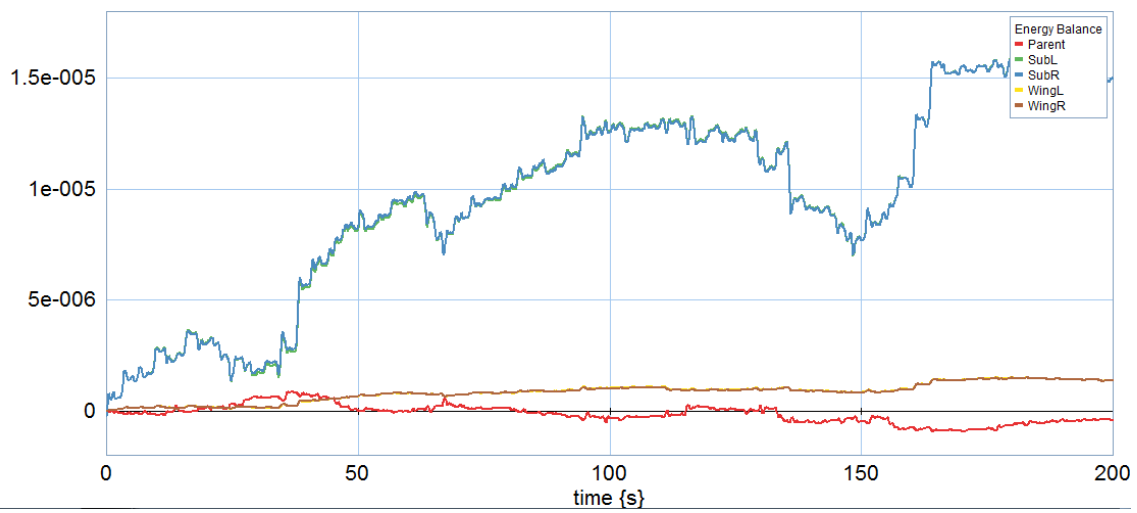


Figure B.5: Flapping wing and prismatic subwing model with gravity compensation energy balance graph.

(File: "\Parent_Child_Modelling\Literature_Models\Bird_prismatic\Bird_prismatic_Gcomp.emx").

Bibliography

- Abbasi, S. H., A. Mughal and A. Khaliq (2021), Bioinspired Feathered Flapping Wing UAV Design for Operation in Gusty Environment, *Journal of Robotics*, **vol. 2021**.
- Abdelbadie, I. M. (2021), Aerodynamic modeling of flapping-wing UAVs in the Port Hamiltonian Framework.
<http://essay.utwente.nl/86162/>
- Abdulrahim, M. and R. Lind (2004), Flight Testing and Response Characteristics of a Variable Gull-Wing Morphing Aircraft, **vol. 5113**.
- Ajaj, R. and G. Jankee (2018), The Transformer aircraft: A multimission unmanned aerial vehicle capable of symmetric and asymmetric span morphing, *Aerospace Science and Technology*, **vol. 76**, pp. 512–522.
- Ajanic, E., M. Feroskhan, S. Mintchev, F. Noca and D. Floreano (2020), Bioinspired wing and tail morphing extends drone flight capabilities, **vol. 5**, no.47, p. eabc2897.
- Baraff, D. (1996), Linear-time dynamics using Lagrange multipliers, *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*.
- Barbarino, S., O. Bilgen, R. Ajaj, M. Friswell and D. Inman (2011), A Review of Morphing Aircraft, *Journal of Intelligent Material Systems and Structures - J INTEL MAT SYST STRUCT*, **vol. 22**.
- Berg, C. and J. Rayner (1995), The moment of inertia of bird wings and the inertial power requirement for flapping flight, **vol. 198**, no.8, pp. 1655–1664, ISSN 0022-0949.
- Bisht, I. S. (2022), France Unveils Flapping-Wing Military Drone.
<https://www.thedefensepost.com/2022/02/15/france-flapping-wing-drone/>
- Caetano, J. V., M. Weehuizen, C. De Visser, G. Croon and M. Mulder (2015), Rigid-Body Kinematics Versus Flapping Kinematics of a Flapping Wing Micro Air Vehicle, *Journal of Guidance, Control, and Dynamics*, **vol. 38**.
- Cardoso Ribeiro, F. (2016), *Port-Hamiltonian modeling and control of a fluid-structure system*, Ph.D. thesis.
- Chen, A., B. Song, Z. Wang, D. Xue and K. Liu (2021), A Novel Actuation Strategy for an Agile Bio-inspired FWAV Performing a Morphing-coupled Wingbeat Pattern.
- Colorado, J., A. Barrientos, C. Rossi and C. Parra (2012), Corrigendum: Inertial attitude control of a bat-like morphing-wing air vehicle, *Bioinspiration biomimetics*, **vol. 8**, p. 016001.
- Controllab Products (2008), 20-sim.
<http://www.20-sim.com/>
- Cyberbotics Ltd. (1998), Webots, Open-source Mobile Robot Simulation Software.
<http://www.cyberbotics.com>
- Da Ronch, A., L. Yongchao, L. Zhang, R. De Breuker, J. Kirn, S. Storm, H. Monner and M. Kintscher (2018), A Review of Modelling and Analysis of Morphing Wings, *Progress in Aerospace Sciences*.
- Delft, U. and U. Wageningen (2007), RoboSwift.
<https://RoboSwift.nl/>
- Dietl, J., T. Herrmann, G. Reich and E. Garcia (2011), Dynamic Modeling, Testing, and Stability Analysis of an Ornithoptic Blimp, **vol. 8**, no.4, pp. 375–386, ISSN 1672-6529.
- Duindam, V., A. Macchelli, S. Stramigioli and H. Bruyninckx (2009), *Modeling and Control of Complex Physical Systems: The Port-Hamiltonian Approach*, ISBN 978-3-642-03195-3.

- Festo AG & Co. KG (2011), SmartBird: Bird flight deciphered.
- Folkertsma, G. A., W. Straatman, N. Nijenhuis, C. H. Venner and S. Stramigioli (2017), Robird: A Robotic Bird of Prey, **vol. 24**, no.3, pp. 22–29.
- Gerdes, J., A. Holness, A. Perez-Rosado, L. Roberts, A. Greisinger, E. Barnett, J. Kempny, D. Lingam, C.-H. Yeh, H. A. Bruck and S. K. Gupta (2014), Robo Raven: A Flapping-Wing Air Vehicle with Highly Compliant and Independently Controlled Wings, **vol. 1**, no.4, pp. 275–288.
- Gill, J. S., M. Saeedi Velashani, J. Wolf, J. Kenney, M. R. Manesh and N. Kaabouch (2021), Simulation Testbeds and Frameworks for UAV Performance Evaluation, in *2021 IEEE International Conference on Electro Information Technology (EIT)*, pp. 335–341.
- Grant, D., M. Abdulrahim and R. Lind (2006), Flight Dynamics of a Morphing Aircraft Utilizing Independent Multiple-Joint Wing Sweep, ISBN 978-1-62410-045-1.
- Guo, T., Z. Hou and Z. Liu (2016), Modelling and simulation of flight dynamics for a gull-wing, in *2016 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 2181–2186.
- Hentati, A., L. Krichen, F. Mohamed and L. Fourati (2018), Simulation Tools, Environments and Frameworks for UAV Systems Performance Analysis, pp. 1495–1500.
- Hong, Y., R. Rashad, S. Noh, T. Lee, S. Stramigioli and F. Park (2021), A Geometric Formulation of Multirotor Aerial Vehicle Dynamics.
- Jafferis, N., E. Helbling, M. Karpelson and R. Wood (2019), Untethered flight of an insect-sized flapping-wing microscale aerial vehicle, *Nature*, **vol. 570**, pp. 491–495.
- Jahanbin, Z., A. Ghafari, A. Ebrahimi and A. Meghdari (2015), Multi-body simulation of a flapping-wing robot using an efficient dynamical model, *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, **vol. 38**.
- Jahanbin, Z. and S. Karimian (2018), Modeling and parametric study of a flexible flapping-wing MAV using the bond graph approach, *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, **vol. 40**.
- ji, B., Q. Zhu, S. Guo, F. Yang, Y. Li, Z. Zhu, S. Chen, R. Song and Y. Li (2020), Design and experiment of a bionic flapping wing mechanism with flapping–twist–swing motion based on a single rotation, *AIP Advances*, **vol. 10**, p. 065018.
- Karásek, M. and A. Preumont (2011), Control of longitudinal flight of a robotic hummingbird model.
- Kate, B., J. Waterman, K. Dantu and M. Welsh (2012), Simbeeotic: A simulator and testbed for micro-aerial vehicle swarm experiments, in *2012 ACM/IEEE 11th International Conference on Information Processing in Sensor Networks (IPSN)*, pp. 49–60.
- Mairaj, A., A. I. Baba and A. Y. Javaid (2019), Application specific drone simulators: Recent advances and challenges, *Simulation Modelling Practice and Theory*, **vol. 94**, pp. 100–117, ISSN 1569-190X.
- Martinez, J., D. Scopelliti, C. Bil, R. Carrese and P. Marzocca (2017), Design, Analysis and Experimental Testing of a Morphing Wing.
- Maschke, B. and A. Van der Schaft (1992), Port-Controlled Hamiltonian Systems: Modelling Origins and Systemtheoretic Properties, **vol. 25**, no.13, pp. 359–365, ISSN 1474-6670, 2nd IFAC Symposium on Nonlinear Control Systems Design 1992, Bordeaux, France, 24-26 June.
- Matloff, L., E. Chang, T. Feo, L. Jeffries, A. Stowers, C. Thomson and D. Lentink (2020), How flight feathers stick together to form a continuous morphing wing, *Science*, **vol. 367**, pp. 293–297.
- Meguid, S., Y. Su and Y. Wang (2017), Complete morphing wing design using flexible-rib system, *International Journal of Mechanics and Materials in Design*, **vol. 13**.

- Merzouki, R., A. Samantaray, P. Pathak and B. Bouamama (2013a), *Intelligent Mechatronic Systems - Modeling, Control and Diagnosis*, ISBN 978-1-4471-4628-5.
- Merzouki, R., A. K. Samantaray, P. M. Pathak and B. O. Bouamama (2013b), *Intelligent Mechatronic Systems: Modeling, Control and Diagnosis*, ISBN 978-1-4471-4628-5.
- Mirtich, B. (1996), Impulse-based dynamic simulation of rigid body systems.
- Neal, D., M. Good, C. Johnston, H. Robertshaw, W. Mason and D. Inman (2004), Design and Wind-Tunnel Analysis of a Fully Adaptive Aircraft Configuration.
- Nearman, A. and D. Van Engelsdorp (2022), Water provisioning increases caged worker bee lifespan and caged worker bees are living half as long as observed 50 years ago.
- Nunn, G. (2021), 'They're territorial': can birds and drones coexist?
<https://www.theguardian.com/environment/2021/oct/01/theyre-territorial-can-birds-and-drones-coexist>
- Obradovic, B. and K. Subbarao (2011), Modeling of Flight Dynamics of Morphing Wing Aircraft, *Journal of Aircraft*, **vol. 48**, pp. 391–402.
- Ollerton, J., R. Winfree and S. Tarrant (2011), How many flowering plants are pollinated by animals?, **vol. 120**, no.3, pp. 321–326.
- Pankonien, A. and D. Inman (2013), Experimental testing of spanwise morphing trailing edge concept, *Proceedings of SPIE - The International Society for Optical Engineering*, **vol. 8688**, pp. 15–.
- Pfeiffer, A. T., J.-S. Lee, J.-H. Han and H. Baier (2010), Ornithopter Flight Simulation Based on Flexible Multi-Body Dynamics, *Journal of Bionik engineering*.
- Previtali, F., A. Arrieta and P. Ermanni (2014), Performance of a Three-Dimensional Morphing Wing and Comparison with a Conventional Wing, *AIAA Journal*, **vol. 52**, pp. 2101–2113.
- Ramezani, A., X. Shi, S.-J. Chung and S. Hutchinson (2016), Bat Bot (B2), a biologically inspired flying machine, in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3219–3226.
- Rashad, R., F. Califano, F. P. Schuller and S. Stramigioli (2021a), Port-Hamiltonian modeling of ideal fluid flow: Part I. Foundations and kinetic energy, *Journal of Geometry and Physics*, **vol. 164**, p. 104201, ISSN 0393-0440.
- Rashad, R., F. Califano, F. P. Schuller and S. Stramigioli (2021b), Port-Hamiltonian modeling of ideal fluid flow: Part II. Compressible and incompressible flow, *Journal of Geometry and Physics*, **vol. 164**, p. 104199, ISSN 0393-0440.
- Robotics, O. (2002), Gazebo.
<https://gazebosim.org/>
- Roderick, W. R. T., M. R. Cutkosky and D. Lentink (2021), Bird-inspired dynamic grasping and perching in arboreal environments, **vol. 6**, no.61, p. eabj7562.
- Rodrigue, H., S. Cho, M.-W. Han, B. Bhandari, J.-E. Shim and S.-H. Ahn (2016), Effect of twist morphing wing segment on aerodynamic performance of UAV, *Journal of Mechanical Science and Technology*, **vol. 30**, pp. 229–236.
- Rohmer, E., S. P. N. Singh and M. Freese (2013), V-REP: A versatile and scalable robot simulation framework, *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1321–1326.
- Samuel, D., S. Grondel, A. Bontemps, C. Eric and C. Daniel (2014), Bond graph model of a flapping wing micro-air vehicle.
- Sanchez-Laulhe, E., R. Fernandez-Feria and A. Ollero (2022), Simplified Model for Forward-Flight Transitions of a Bio-Inspired Unmanned Aerial Vehicle, **vol. 9**, no.10.

- Send, W., M. Fischer, K. Jebens, R. Mugrauer, A. Nagarathinam and F. Scharstein (2012), Artificial hinged-wing bird with active torsion and partially linear kinematics, *28th Congress of the International Council of the Aeronautical Sciences 2012, ICAS 2012*, **vol. 2**, pp. 1148–1157.
- Serrani, A., B. Keller, M. Bolender and D. Doman (2010), Robust Control of a 3-DOF Flapping Wing Micro Air Vehicle*, ISBN 978-1-60086-962-4.
- Shyy, W., H. Aono, S. Chimakurthi, P. Trizila, C.-K. Kang, C. Cesnik and H. Liu (2010), Recent progress in flapping wing aerodynamics and aeroelasticity, **vol. 46**, no.7, pp. 284–327, ISSN 0376-0421.
- Sibilski, K., L. Lorocho, W. Buler and A. Zyluk (2004), Modeling and Simulation of the Nonlinear Dynamic Behavior of a Flapping Wings Micro-Aerial-Vehicle, ISBN 978-1-62410-078-9.
- Siddall, R., A. O. Ancel and M. Kova (2017), Wind and water tunnel testing of a morphing aquatic micro air vehicle, *Interface Focus*, **vol. 7**.
- Smith, D., M. Lowenberg, D. Jones and M. Friswell (2014), Computational and Experimental Validation of the Active Morphing Wing, *Journal of Aircraft*, **vol. 51**, pp. 925–937.
- Stramigioli, S. (2015), Energy-Aware Robotics, in *Mathematical Control Theory I*, Springer, pp. 37–50, ISBN 978-3-319-20987-6.
- Stramigioli, S. (2021), Lecture notes in Modern Robotics, University Of Twente.
- Stramigioli, S., F. Califano, R. Rashad, A. Dijkshoorn, L. G. Koerkamp, R. Sneep and A. Brugnoli (2018), The PortWings Project.
<http://www.portwings.eu/>
- Suzuki, A. (2022), Experts warn drone pilots of bird attack risks.
<https://japannews.yomiuri.co.jp/society/general-news/20220925-60544/>
- Van der Schaft, A. and J. Cervera (2002), Composition of Dirac structures and control of Port-Hamiltonian systems, in *Proceedings of the 15th International Symposium on the Mathematical Theory of Networks and Systems*, Eds. D. Gilliam and J. Rosenthal, University of Notre Dame, United States, pp. –, ISBN not assigned.
- Vasista, S., F. Nolte, H. Monner and P. Horst (2017), 3D Design of a Large-Displacement Morphing Wing Droop Nose Device, *Journal of Intelligent Material Systems and Structures*, **vol. 29**.
- Vos, R. and M. Abdalla (2010), Mechanism for Warp-Controlled Twist of a Morphing Wing, *Journal of Aircraft - J AIRCRAFT*, **vol. 47**, pp. 450–457.
- Wilkinson, I. (2022), Spanish police disguise drones as birds of prey to spy on smugglers.
<https://www.thetimes.co.uk/article/police-release-bird-of-prey-drones-to-spy-on-smugglers-6fj8tfl8f>
- Yang, W., L. Wang and B. Song (2018), Dove: A biomimetic flapping-wing micro air vehicle, **vol. 10**, no.1, pp. 70–84.
- Yoshimura, H. and J. E. Marsden (2006), Dirac structures in Lagrangian mechanics Part I: Implicit Lagrangian systems, **vol. 57**, no.1, pp. 133–156, ISSN 0393-0440.
- Zheng, L., T. Hedrick and R. Mittal (2013), A comparative study of the hovering efficiency of flapping and revolving wings, *Bioinspiration biomimetics*, **vol. 8**, p. 036001.
- Zufferey, R., J. Barbero, M. Garcia, F. Fernandez, E. Sanchez-Laulhe, P. Grau, M. Capote, J. Acosta and A. Ollero (2021), Design of the High-Payload Flapping Wing Robot E-Flap, *IEEE Robotics and Automation Letters*, **vol. PP**, pp. 1–1.
- Zufferey, R., J. Tormo-Barbero, D. Feliu-Talegón, S. Nekoo, J. Acosta and A. Ollero (2022), How ornithopters can perch autonomously on a branch, *Nature Communications*, **vol. 13**.