Multirotor drone sizing and trajectory optimization within Modelon Impact

Clément Coïc[1] Marc Budinger[2] Scott Delbecq[3]

[1]Modelon, Germany, clement.coic@modelon.com

[2]*Institut Clément Ader (ICA), Université de Toulouse, CNRS-INSA-ISAE-Mines Albi-UPS, Toulouse, France*

[3]ISAE-SUPAERO, Université de Toulouse, France

Abstract

The design of multirotor drones often relies on optimizing its performance in terms of maximum speed requirements and hover time. This is well suited to undefined tasks. In the case of repetitive tasks, the drone trajectory can be added as a third degree of freedom. This paper focuses on the use of Modelon Impact and its dynamic optimization capabilities to reach a multirotor drone design and 1-D trajectory optimization. In comparison to other options investigated by the authors in a separate publication, Modelon Impact based optimization proved to be much simpler, more robust, and faster – for this use case.

Keywords: Multirotor, Drone, Dynamic optimization, Trajectory optimization, Sizing, Modelon Impact, Optimica,

1 Introduction

Multirotor drones are often associated with toys that are fun to pilot. The drone designer does not know in advance who will use the drone and how it will be used. Therefore, these drones are typically designed based on performance requirements. For a toy drone to be fun, the user expects it to be fast and to have a satisfying autonomy. The toy drone designer often takes as requirement a maximum speed and a given hover time.

On the other side, multirotor drones are also being developed for some industry applications for a variety of roles – from packages delivery to military assistance or payload lifting in substitution to cranes. Contrarily to the toy drones, industry drones typically have well defined missions often expressed as:

- Start point: initial elevation, hover time
- End point: final elevation, horizontal distance from the initial point and hover time

Missions including more points can be described as sequences of start and end points.

Getting back to the sole mission of the drone – in comparison to optimizing for performance requirements – relaxes an entire degree of freedom: the drone trajectory. Solving both the drone sizing and trajectory optimizations allows focusing on optimization criteria such as minimizing energy consumption or the cost function, if willing to associate a cost to the parts that compose the drone and its utilization (time of utilization – including potential operator – and energy consumption).

This paper presents how easy it is to perform sizing and trajectory optimization of a system within Modelon Impact. A drone is selected as example system. A selected case study – payload lifting – is introduced in section 2. The drone model is discussed in section 3 of this paper, with a particular focus on the propeller. In section 4, we present the optimization problem, the simplicity of its implementation in Modelon Impact and the associated results. Finally, section 5 is a collection of advantages that come with optimizing using OPTIMICA and Modelon Impact.

2 Case Study – Payload lifting

2.1 Use Case

It is typical on construction sites – mostly when approaching the end of the construction – to either keep a crane operating for a longer period time to lift some minor equipment or material, or to carry this payload by human strength. While the former solution is often expensive, the latter requires manpower and can affect physical health. The use of drones to lift these small payloads (Draganfly, 2022) is an economically attractive solution which has a low impact on physical health.

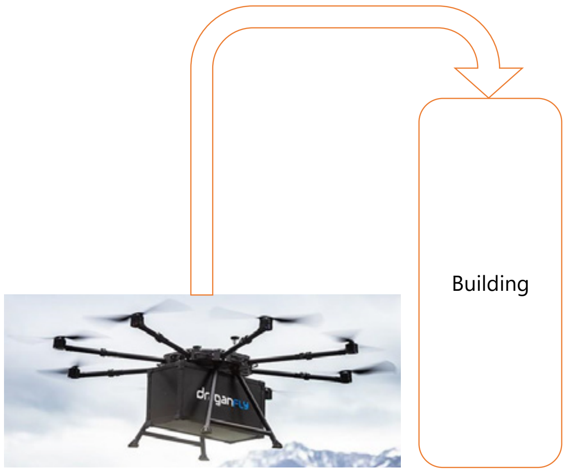

Figure 1. Illustration of the load lifting use case.

This paper focuses on a drone design aimed at a repetitive task: small payload lifting – below 25 kg – from ground to the top of a building.

2.2 Main Requirements

To match the use case presented in section 2.1, the main requirements for the drone designs are related to the lifting operation in terms of payload and endurance:

- Payload: The drone shall lift masses up to 25 kg during its full operation.
- Endurance: The drone shall operate at least 150 climbs of at least 10 m height with 5 seconds hovering – for the handling of the payload.

For this use case, the drone returns to ground without any payload and thus with very little energy consumption. Should the drone carry payload on the descent, the number of climbs would be reduced inevitably.

3 From Drone Architecture to Model

This section details the drone architecture, technological choices, the sizing problem formulation and discusses the associated model.

3.1 Drone Architecture

Different architectural choices can be made depending on the purpose and the mission. For multirotor drones, the main choices are the number of arms and the number of propellers per arms but also the materials and technologies of the components.

As the main usage of such a drone is in urban area, it was decided to select a fully electric drone design. Indeed, electric motor emit less pollution – in terms of emissions, smell, and noise – than the combustion ones. In the presented work, a single architecture is considered for the multirotor drone. The architecture is presented in Figure **2** and is composed of:

1. Four (4) fixed pitch propellers
2. Four (4) out-runner brushless motors
3. Four (4) electronic speed controllers (ESC) mainly made from MOSFET inverters
4. One (1) battery based on Li-Ion cells
5. One (1) mechanical structure (frame) consisting of four (4) arms and one (1) central body

Investigating variable architecture designs is let as perspective to this work.

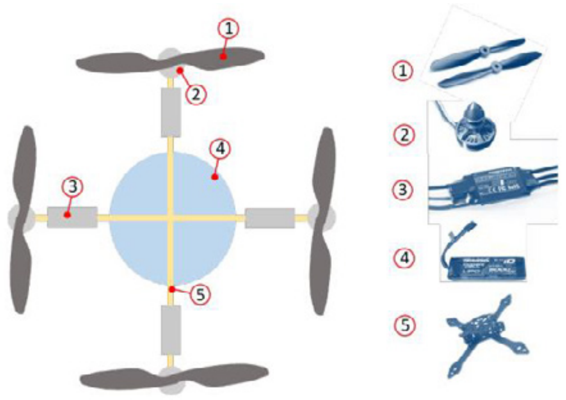

Figure 2. Multirotor drone architecture and components.

3.2 Discussion on the Drone Model Fidelity

As for physical systems, a simulation model is developed for a given purpose. This purpose guides technological choices on the model development, such as general assumptions, the physical effects modeled, the level of details, the smoothing, etc. These are all gathered within the so-called model fidelity.

The purpose of the multirotor drone model for this paper is to perform a component sizing and simultaneously optimize the 1-D trajectory of the drone to perform a well-defined task. From these two purposes, we can extract a few technological choices.

<u>Sizing purpose</u>

Implications on model causality

When it comes to sizing a system based on a desired trajectory – here, imposed by the optimization algorithm – , it is often necessary to reverse the power chain. This is also known as bicausality.

A performance simulation of the drone model would typically require a known load at the propellers and would compute the resulting speed of the drone, for a given command. On the contrary, a sizing scenario would provide both the load at the rotors and the drone speed. These two variables define the required power output, which can be propagated upstream on the power chain to compute the require power at the power generation or storage – here, the battery.

As the design validation would require a performance simulation, the acausality of the Modelica language is clearly a benefit to solve sizing problems. The sizing solving requires propagating the power variables through the component ports in one way or another – e.g. by having *flow* and *non-flow* which product would lead to the power or by adding the power to a connector.

Selection of model complexity

For cost reasons, we assume the drone will be assembled using off-the-shelf components. The sizing problem thus consists mainly of finding a good order of magnitude for each component size. Therefore, physical effects to be modeled should only be the dominant ones and their implementation complexity could be quite low. Consequently, scaling laws are used to design the components and efficiencies are often used to assume losses – instead of overloading the model with multiple separate physical losses, e.g. each separate friction.

Optimization purpose

Most optimization algorithms rely on gradients to define on which direction they should perform the next step. This means that the variables of the model should, at least, be continuous and derivable. A special effort is made in the Modelica code to respect this constrain.

In addition, minimum, maximum and nominal values of the design variables are provided to allow respectively to bound and normalize the variables and equations – key for optimization convergence.

Finally, dynamic optimization benefits from having both an initial and a nominal trajectory that match the specified requirements – without necessarily be optimal. This is easily achieved by simulating first the drone behavior with a smooth trajectory command. Here, the acausality of the Modelica language becomes once more convenient.

3.3 Drone Model – Focus on Propeller

The drone Modelica model serves two purposes:

1. It includes the scaling laws for all components to allow their sizing.
2. It encodes the physics equations to model the flight performance and power consumption.

The component sizing models used are scaling laws, linear regressions of data sheet and surrogate models – detailed by Budinger (2020). The physics equations are well known equations from components and are presented by Delbecq (2021). It is however relevant to present here the propeller model, as a representative component of the drone.

The propeller represents a key component in the drone propulsion chain. Its performance can be expressed as a function of two coefficients C_T and C_P, respectively expressing thrust (1) and mechanical power (2) equations:

$$Thrust = C_T \rho_{air} n^2 D^4 \tag{1}$$

$$Power = C_P \rho_{air} n^3 D^5 \tag{2}$$

where ρ_{air} represents the air density in [kg/m^3], n the rotational speed in [rev/s] and D the propeller diameter in [m].

While C_T and C_P are dimensionless, they are not constant. These depend on further variables such as the blade pitch p, the air Bulk Modulus K, the relative airspeed V (normal to the rotor plane). It is here assumed that:

$$Thrust = f(D, p, \rho, K, n, V) \tag{3}$$

$$\rightarrow C_T = f(D, p, \rho, K, n, V)/(\rho_{air} n^2 D^4) \tag{4}$$

Performing a dimensional analysis on these equations allows identifying a reduced set of dimensionless variables that can define this equation. Note that Buckingham's theorem gives us the insight that these dimensionless variables are in number of 3 (6 variables and 3 units dimensions). The detailed dimensionless analysis is available on request – please email the authors. This gives us the following three dimensionless numbers:

- The pitch to diameter ratio: $\beta = pitch/D$.
- The advance ratio: $J = V/(nD)$.
- The air compressibility indicator: $B = K/(\rho n^2 D^2)$

The analysis on C_P reveals the same dimensionless numbers. As both the thrust and power coefficients are surface responses, these are better fitted with polynomial regression rather than with power regression – as discussed in (Sanchez 2017).

A sensitivity study was conducted in (Budinger 2020) on the three dimensionless numbers within the domain of usage of the drone. It showed that both C_T and C_P are quite insensitive to the air compressibility indicator, within this domain. Finally, the fitting revealed the following equations:

$$\begin{aligned} C_T \approx\ & 0.02791 - 0.06543J - 0.23504J^2 \\ & + 0.02104J^3 + 0.11867\beta \\ & + 0.27334\beta^2 - 0.28852\beta^3 \\ & + 0.18677\beta J^2 \end{aligned} \tag{5}$$

$$\begin{aligned} C_P \approx\ & 0.01813 - 0.00343J - 0.12350J^2 \\ & + 0.06218\beta + 0.35712\beta^2 \\ & - 0.23774\beta^3 + 0.07549\beta J \end{aligned} \tag{6}$$

The surface responses and the corresponding datasets are presented below for both coefficients.

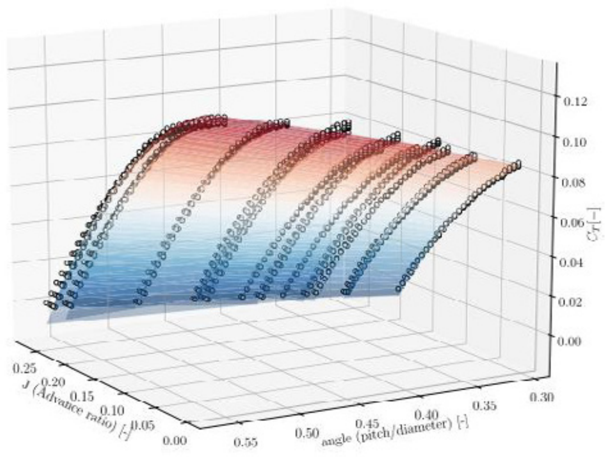

Figure 3. Surface response of C_T and reference dataset (dots).

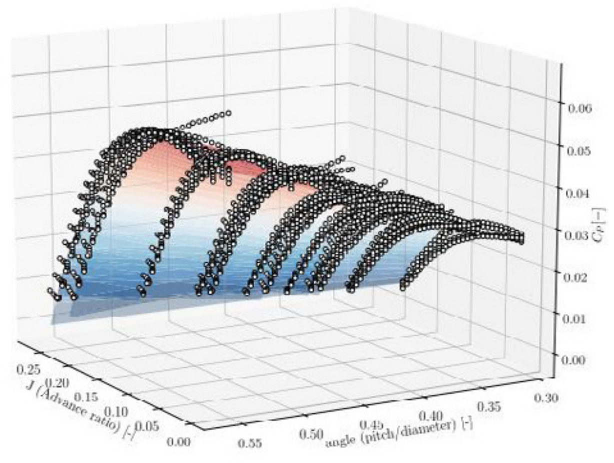

Figure 4. Surface response of C_P and reference dataset (dots).

This gives us the performance model of the propeller, that needs to be completed by its sizing model. For the propeller, that parameters of interest are the mass and the inertia. These are computed based on scaling laws. The propeller mass M_{prop} is found to be proportional to the cube of its diameter. From the mass, the inertia I_{prop} can be computed. Note that scaling laws require similar reference data to scale on to.

$$M_{prop} = \mathrm{M_{ref}}\left(D_{prop}/D_{ref}\right)^2 \tag{7}$$

$$\mathrm{I_{prop}} = \mathrm{M_{prop}}\left(D_{prop}/2\right)^3/3 \tag{8}$$

From this complete set of equations, we get the propeller sizing and associated performance – impacted by the sizing. Similar models are developed for all components and the following parts of the paper will focus on the general sizing problem formulation and optimization problem, rather than detailing each component. As a reminder, these equations are available in (Budinger 2020) and the propeller model fidelity is representative of the rest of the model. If of interest, a more detailed Modelica performance model is presented by Podlaski (2020).

3.4 Sizing Problem Formulation

The main sizing scenarios, design drivers and models for vertical flight applications of multirotor drones are summarized in Figure 5. Such applications consist of three sizing scenarios to be considered in the design problem that are:

1. the hovering flight with the advance ratio of the propeller $J = 0$ – as the air speed V is null.

2. the takeoff phase which requires maximum power to accelerate the drone with an increasing J – increasing air speed V.

3. the climb phase with a constant vertical speed and thus constant J.

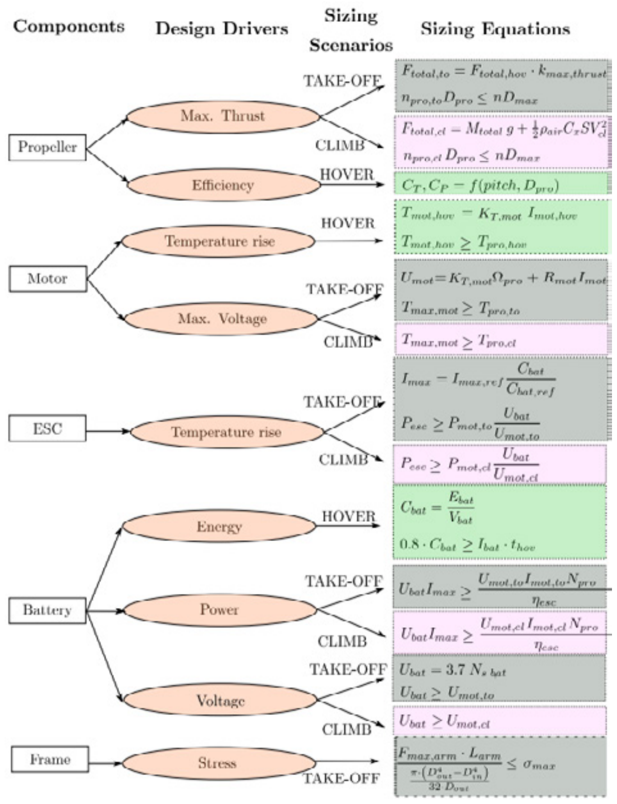
Components
Design Drivers
Sizing Scenarios
Sizing Equations
Propeller
Max. Thrust
Efficiency
Motor
Temperature rise
Max. Voltage
ESC
Temperature rise
Battery
Energy
Power
Voltage
Frame
Stress
TAKE-OFF
CLIMB
HOVER
$F_{total,to} = F_{total,hov} \cdot k_{max,thrust}$
$n_{pro,to} D_{pro} \leq nD_{max}$
$F_{total,cl} = M_{total}\, g + \frac{1}{2}\rho_{air} C_x S V_{cl}^2$
$n_{pro,cl} D_{pro} \leq nD_{max}$
$C_T, C_P = f(pitch, D_{pro})$
$T_{mot,hov} = K_{T,mot} I_{mot,hov}$
$T_{mot,hov} \geq T_{pro,hov}$
$U_{mot} = K_{T,mot}\Omega_{pro} + R_{mot} I_{mot}$
$T_{max,mot} \geq T_{pro,to}$
$T_{max,mot} \geq T_{pro,cl}$
$I_{max} = I_{max,ref}\frac{C_{bat}}{C_{bat,ref}}$
$P_{esc} \geq P_{mot,to}\frac{U_{bat}}{U_{mot,to}}$
$P_{esc} \geq P_{mot,cl}\frac{U_{bat}}{U_{mot,cl}}$
$C_{bat} = \frac{E_{bat}}{V_{bat}}$
$0.8 \cdot C_{bat} \geq I_{bat} \cdot t_{hov}$
$U_{bat} I_{max} \geq \frac{U_{mot,to} I_{mot,to} N_{pro}}{\eta_{esc}}$
$U_{bat} I_{max} \geq \frac{U_{mot,cl} I_{mot,cl} N_{pro}}{\eta_{esc}}$
$U_{bat} = 3.7\, N_{s\,bat}$
$U_{bat} \geq U_{mot,to}$
$U_{bat} \geq U_{mot,cl}$
$\frac{F_{max,arm} \cdot L_{arm}}{\frac{\pi \cdot (D_{out}^4 - D_{in}^4)}{32 \cdot D_{out}}} \leq \sigma_{max}$

Figure 5. Design drivers and equations of the multirotor drone.

The overall sizing model has been adapted from (Delbecq 2021) which is tailored for vertical flight applications of multirotor drones, and should be consulted for a more detailed sizing formulation.

4 Optimization Problem

4.1 Optimization Problem Formulation

The simultaneous trajectory and design optimization problem can be formulated as a mass optimization problem including the trajectory variables (motor torque command and final time):

$$\begin{aligned}
&\textbf{minimize } MTOW \\
&\textbf{with respect to } \beta_{prop}, k_{MTOW}, k_{ND}, k_{mot}, k_{mot,speed} \\
&\qquad k_{bat,mass}, k_{bat,voltage}, k_{arm}, T_{mot}(t), a_{t0}, t_f \\
&\textbf{subject to } T_{prop,t0} - T_{mot,max} \leq 0 \\
&\qquad E_{mission} - E_{bat} \leq 0 \\
&\qquad U_{mot} - U_{ESC} \leq 0 \\
&\qquad U_{ESC} - U_{bat} \leq 0 \\
&\qquad MTOW_f - MTOW \leq 0 \\
&\qquad T_{mot}(t) - T_{mot,max} \leq 0 \\
&\qquad h - z(t_f) \leq 0 \\
&\qquad \dot{z}(t_f) = 0
\end{aligned} \tag{9}$$

The objective is to minimize the weight of the vehicle for the defined mission with respect to design variables such as the propeller pitch β_{pro} or the nominal motor torque T_{mot}. The design has to respect some constraints to respect the technological constraints of components such as motor maximum electromagnetic torque $T_{mot,max}$ as well as others to respect voltages consistency in the power train ($U_{mot} \leq U_{ESC}$ and $U_{ESC} \leq U_{bat}$). Some consistency constraints are used to solve multidisciplinary couplings as suggested by Delbecq (2020) ($E_{mission} \leq E_{bat}$ and $MTOW_f \leq MTOW$). This also requests to add normalized design variables such as k_{MTOW} and $k_{bat,mass}$.

Instead of minimizing the drone weight, minimizing the energy could have been used as objective. Unfortunately, within the time given to investigate this solution, this seemed to be a less robust option. Minimizing the mass is a conscious problem simplification – a lower mass means less energy to carry it. The authors acknowledge that minimizing the energy might lead to a slightly different optimum to this problem.

4.2 Optimica Implementation

Modelon Impact (Coïc 2020-b) is a state of the art, cloud-based modeling and simulation environment, relying on open standards such as Modelica, FMI and Python. Modelica models can be developed within Modelon Impact by composition (drag and drop and connect) of existing models from available Modelica libraries, or by writing Modelica code within the code editor. Modelon Impact compiler, Optimica Compiler Toolkit (OCT), compiles the models – either in steady-state or dynamic simulation mode.

As many engineering problems can be cast as optimization problems – including optimal control, minimum time problems, optimal design, and model calibration – Modelon Impact compiler supports optimization of dynamic and steady state models. This is achieved relying on the extension of Modelica language for optimization: OPTIMICA (Åkesson 2008).

As the OPTIMICA language extends the Modelica language, it is convenient to opt for a similar approach when formulating an OPTIMICA optimization problem. Thus, the optimization model could be built as follow:

1. Create an *optimization* class and provide modifiers to set up the objective and time constraints.
2. Extend the Modelica model and provide modifiers to fix or relax parameters as well as minimum, maximum and nominal values.
3. Optionally add more variables and equations.
4. Add the constraints of the optimization problem.

The Optimica code of the drone optimization is listed in Listing 1.

Listing 1. OPTIMICA Code of the Drone Optimization

```
optimization SizingAndTrajectoryOptim (
  objective=M_total(startTime),
  finalTime(free=true, min=1, max=10, start=5))
// Minimize the total drone mass and relax the final simulation
time within bounds.

  import Modelica.Units.SI.DimensionlessRatio;

  extends Drone(
    x(start = 0, fixed=true),
    xp(start = 0, fixed=true),
    a(start = 0, fixed=true),
    beta(free=true, min=0.3, max=0.6, start=0.4),
    D(free=true, min=0, max=1),
    T_nom_mot(free=true, min=0),
    K_mot(free=true, min=0),
    M_bat(free=true, min=0, max=100),
    P_esc(free=true, min=0),
    k_D(free=true, min=0.01, max=1, start=0.05),
    D_out_arm(free=true, min=0.001, max=1));
// Inherit the Modelica drone model, fix initial conditions and relax
design parameters within bounds.

  Modelica.Blocks.Interfaces.RealInput Traj_in;
// Add input to the trajectory to optimize

  DimensionlessRatio n_norm(start=1, fixed=true)=n/n_hover;

  DimensionlessRatio N_norm(min=-1, max=1,
nominal=0.8)=ND/ND_max;

  DimensionlessRatio T_hov_norm(min=0, max=1,
nominal=0.6) = T_hover/T_nom_mot;

  DimensionlessRatio T_norm(min=-1, max=1, nominal=0.95) =
T/T_max_mot;

  DimensionlessRatio U_norm(min=0, max=1, nominal=0.5) =
U_mot/V_bat;

  DimensionlessRatio P_norm(min=0, max=1, nominal=0.5) =
P_mot/P_esc;

  DimensionlessRatio E_norm(min=0, max=1, nominal=0.25) =
E_drone/E_bat;

  DimensionlessRatio sigma_norm(min=-1, max=1,
nominal=0.15) = sigma/sigma_max;
// Create additional normalized variables with bounds as inequality
constraints

equation

  T=Traj_in; // Bind drone trajectory with optimization input

constraint

  x(finalTime) = 10;

  xp(finalTime) = 0;

  a(finalTime) = 0;
// Define end time constraints.

end SizingAndTrahjectoryOptim;
```

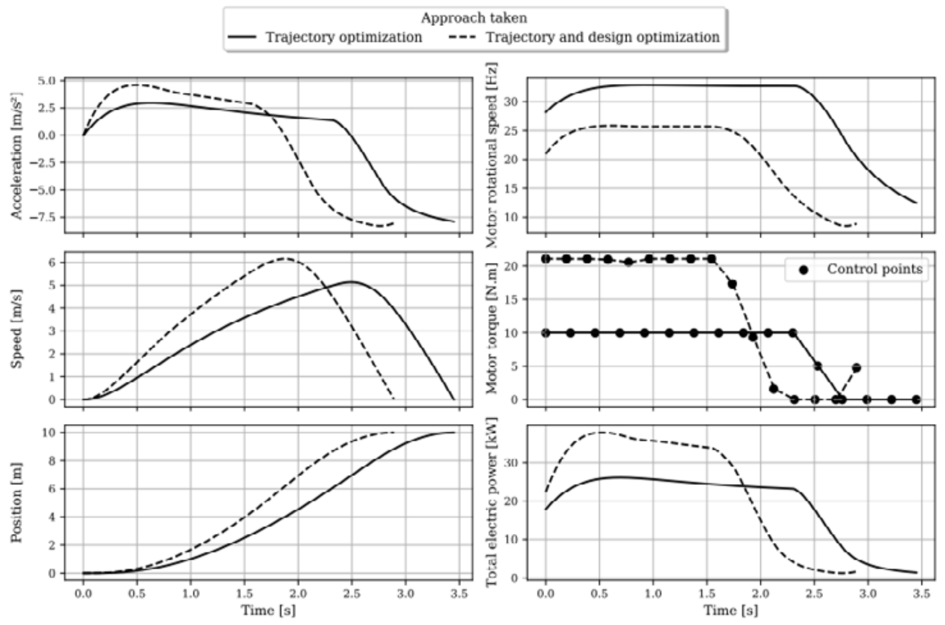

Figure 6. Comparison of resulting trajectories for the trajectory optimization with and without the sizing.

It appears that the OPTIMICA language is straightforward for a person used to the Modelica language – which itself is quite straightforward for many engineers. This way, the threshold to develop an optimization model is minimized.

4.3 Optimization Results

The result of the optimization problem – solving both the design (component pre-sizing) and trajectory optimization problems – are shown in dashed lines, for some variables, in **Figure 6**.

The solution of the sole trajectory optimization – with a separate sizing – is presented in full lines. This highlights the differences in trajectories when sizing is added as part of the optimization. When combining both, the optimizer could assess that it was more energy efficient to increase the size of the propeller, reducing its rotational speed, allow a smoother trajectory and compensating by a bigger battery – that can allow the relevant number of ascents.

After scaling to 150 climbs (endurance requirement), the optimum resulted in a drone weighting about 45 kg (without payload) with a battery contributing to more than half of the weight (about 27 kg). A drone designer might find interesting to investigate an easily replaceable battery pack in order to reduce the endurance requirement – leading to potentially more convenient (and safer) drones to operate.

These results were obtained with similar orders of magnitude with both Modelon Impact and a comparative solution based on FAST-OAD (David 2021).

5 The Benefits of Modelon Impact

There are several advantages in using Modelon Impact compared to a separate optimization with a Python package or in a dedicated optimization platform such as OpenMDAO, even if relying on a FMU for the plant model.

A key advantage of using Modelon Impact is that it relies on its OPTIMICA compiler and the OPTIMCA language, as mentioned previous, extends the Modelica language. Hence, a fair amount of the benefits listed below are Modelica features serving the optimization purposes.

5.1 Solving Initialization Problem

The Modelica language can deal with both Initial Value Problem (IVP) and Boundary Value Problem (BVP) for the model initialization. In the former case, the user provides the initial values for every state. In the latter, the number of independent initial values should match the number of states but is not necessarily their initial values.

Therefore, Modelon Impact compiler solves an initial problem – different from the dynamic problem – to resolve the BVP. In this process, the compiler can solve linear and non-linear systems, which is often not tolerated out of the box with a different solution. For example, in the Python optimization, it was necessary to adapt the code, define iteration variables (coefficient factors) and residuals (equations that should tend to zero) to solve these non-linear problems.

5.2 Acausality – One Model Several Purposes

A second advantage is that the Modelica language is acausal. This way, defining a Drone model based on the equations of the physics makes it useful for several use cases: position, speed or torque command. In our example, we want to optimize the torque trajectory while sizing the drone. Nevertheless, it is convenient to provide the optimizer a start trajectory not too far from our constraints. This is easily achieved by simulating the same model, providing a position trajectory which not optimized at all but matching our requirements.

5.3 Normalization for Convergence

Modelica models and FMUs usually use variables expressed in SI units. Variable values may therefore differ by several orders in magnitude (Coïc 2020-a). A typical example is thermodynamic models containing pressures, temperatures and mass flows. Such large differences in scales may have a severe deteriorating effect on the performance of numerical algorithms and may in some cases even lead to the algorithm failing. In order to relieve the user from the burden of manually scaling variables, Modelica offers the nominal attribute, which can be used to automatically scale a model. Modelon Impact compiler can use these nominal attributes to scale the variables (and thus objective) of the optimization problem. In addition, it is possible to provide a reference trajectory for scaling at every time step of the simulation.

5.4 Derivatives at Hand of the Optimization

Finally, the Optimica language, being an extension of Modelica, has access to all the equations of the model and can process these. Thus, the compiler automatically computes all the derivatives it requires to secure a fast and robust convergence to a global optimum of the problem.

5.5 Simpler, Faster and More Robust

Modelon Impact solution also proved to be much simpler than the python with FAST-OAD approach. The Modelica code wasn't written differently for performance simulation and for optimization purposes as it was the case for its Python version. Also, the optimization problem formulation in OPTIMICA language is very simply expressed, as mentioned above (see Listing 1), and does not diverge much from the Modelica language – which makes it really easy for a Modelica developer to ramp up on OPTIMICA

The same optimization problem was solved in Modelon Impact and using FAST-OAD. For this optimization problem that consists of 10 design variables, 7 inequality and 1 equality constraint, Modelon Impact could solve the problem in less than 30 seconds while it took more than 2 minutes to FAST-OAD. This can be explained by the different level of information on the model the optimizer has – by default FAST-OAD does not have access to the internal derivatives of the model.

Finally, and this might sound unfortunately qualitative, the Modelon Impact solution was more robust, more straightforward to converge. While it took several hours of debugging to get the Python code running and optimization solving with FAST-OAD, it appeared to work almost directly with Modelon Impact. In all transparency, the first attempt did not involve normalization of the added variables for inequality constraints, and it failed to converge. Normalizing solved the issue.

5.6 The Benefits of FAST-OAD

There are many applications for which higher model fidelities are required. FAST-OAD supports easy coupling with Computational Flow Dynamics or Finite Element models. FAST-OAD scales also very well with the number of models involved in the optimization process. Therefore, the authors value both technologies and, indeed, some authors are major contributors to FAST-OAD development.

6 Conclusion

This paper uses a multi-rotor drone (pre-)sizing and trajectory optimization to illustrate the needs for solving such a problem. The model fidelity is not necessarily the highest but constrains on the numerical aspect of the code are highlighted – e.g. acausality, smoothness, etc. The models shall be "optimization-friendly". The propeller model is detailed to emphasize the level of details sufficient for such a purpose.

In a second step, the optimization problem has been expressed, first analytically and then in OPTIMICA language – supported by Modelon Impact. The solving of the problem is achieved, and the benefit of this solution are discussed, in a generic way, and in comparison with another implementation using Python and FAST-OAD.

The results proved that solving the sizing problem in combination with the trajectory optimization leads to a better design, compared to solving both problems sequentially. All industries could benefit from optimizing systems – that are meant to exist – in a more complete manner, e.g. including dynamic optimization of trajectories.

As a perspective of work, we could show how the same model serve the purpose of Model Predictive Control of the multi-rotor drone to actually reach the optimum trajectory it was designed for. Another axis of improvement could be to use one of Modelon's 6 degrees of freedom drone model to be able to include environmental constraints – such as a wind field – in the overall design optimization problem.

References

Åkesson Johan (2008). "Optimica—An Extension of Modelica Supporting Dynamic Optimization". In *6th International Modelica Conference,* Bielefeld, Germany, 2008.

Budinger Marc, Aurélien Reysset, Aitor Ochotorena and Scott Delbecq (2020). "Scaling laws and similarity models for the preliminary design of multirotor drones". In *Aerospace Science and Technology*, 98. 1-15. ISSN 1270-9638.

Coïc Clément, Moritz Hübel and Matthis Thorade (2020). "Enhanced Steady-State in Modelon Jet Propulsion Library, an Enabler for Industrial Design Workflows". In *American Modelica Conference 2020*, Boulder, Colorado, USA.

Coïc Clément, Johan Andreasson, Anand Pitchaikani, Johan Åkesson and Hemanth Sattenapalli (2020). "Collaborative Development and Simulation of an Aircraft Hydraulic Actuator Model". In *Asian Modelica Conference 2020*, Tokyo,Japan.

David Christophe, Scott Delbecq, Sébastien Defoort, Peter Schmollgruber, Emmanuel Benard, Valérie Pommier-Budinger (2021). "From FAST to FAST-OAD: An open source framework for rapid Overall Aircraft Design". In *10th EASN Virtual International Conference on Innovation in Aviation & Space to the Satisfaction of the European Citizens*.

Delbecq Scott, Marc Budinger, Clément Coïc, and Nathalie Bartoli (2021). "Trajectory and design optimization of multirotor drones with system simulation". In *American Institute of Aeronautics and Astronautics, Inc*, *SciTech*, DOI: 10.2514/6.2021-0211.

Delbecq Scott, Marc Budinger and Aurélien Reysset (2020). "Benchmarking of monolithic MDO formulations and derivative computation techniques using OpenMDAO". In *Structural and Multidisciplinary Optimization*, Vol. 62, No. 2, 2020, pp. 645–666. DOI: 10.1007/s00158-020-02521-7.

Draganfly website – heavy lift, accessed in August 2022. https://draganfly.com/heavy-lift/

Podlaski Megan, Luigi Vanfretti, Hamed Nademi and Hao Chang (2020). "UAV Dynamics and Electric Power Systems Modeling and Visualization using Modelica and FMI". In *American Modelica Conference 2020*

Sanchez Florian (2017). "Génération de modèles analytiques pour la conception préliminaire de systèmes multi-physiques : application à la thermique des actionneurs et des systèmes électriques embarqués". Doctoral's thesis. Université Toulouse 3 Paul Sabatier, France. URL: http://thesesups.ups-tlse.fr/3555/1/2017TOU30081.pdf.