This is a preprint of: Rodríguez, F., Díaz-Báñez, J. M., Sanchez-Laulhe, E., Capitán, J., & Ollero, A. (2021). Kinodynamic planning for an energy-efficient autonomous ornithopter. Computers & Industrial Engineering, 107814

Kinodynamic Planning for an Energy-Efficient Autonomous Ornithopter

Fabio Rodríguez[1] José-Miguel Díaz-Báñez[2,*] Ernesto Sanchez-Laulhe[1]
Jesús Capitán[1] Aníbal Ollero[1] ‡

December 1, 2021

Abstract

This paper presents a novel algorithm to plan energy-efficient trajectories for autonomous ornithopters. In general, trajectory optimization is quite a relevant problem for practical applications with *Unmanned Aerial Vehicles* (UAVs). Even though the problem has been well studied for fixed and rotatory-wing vehicles, there are far fewer works exploring it for flapping-wing UAVs, like ornithopters. These are of interest for many applications where long-flight endurance, but also hovering capabilities, are required. We propose an efficient approach to plan ornithopter trajectories that minimize energy consumption by combining gliding and flapping maneuvers. Our algorithm builds a tree of dynamically feasible trajectories and it applies heuristic search for efficient online planning, using reference curves to guide the search and prune states. We present computational experiments to analyze and tune the key parameters, as well as a comparison against a recent alternative probabilistic planner, showing best performance. Finally, we demonstrate how our algorithm can be used for planning perching maneuvers online.

Keywords: Trajectory optimization, ornithopter, motion planning, nonlinear dynamics.

1 Introduction

Unmanned Aerial Vehicles (UAVs) are spreading quite fast for many applications due to their versatility and autonomy. However, they present two main barriers to reach a wider range of applications: (i) flight endurance; and (ii) safety during interactions with people and objects in the environment. For instance, flight endurance is essential in applications like long-range inspection of infrastructures (e.g., power lines). Conventional multi-rotor UAVs do not achieve competitive flight times for those scenarios; and the use of fixed-wing UAVs does not solve the problem completely either, as capabilities for *Vertical Take-Off and Landing* (VTOL) and hovering in-place are required for accurate inspections.

Additionally, most of the aforementioned platforms are not safe enough to interact with people, due to their powerful rotor systems, their blades and the hard materials of their airframes. In order to cope with both issues, endurance and safety, bio-inspired UAVs, like flapping-wing vehicles or ornithopters, are of interest Grauer & Hubbard (2010); Q.-V. Nguyen & Chan (2018); de Croon et al. (2009); Arabagi et al. (2012). These try to imitate birds flying, as birds can travel long distances efficiently. Thus, the main objective of the GRIFFIN project [1], which is the one inspiring this work, is the design of bio-inspired flapping-wing UAVs that are able not only to fly but also to perch in order to interact with the environment through manipulation.

A key aspect for the development of these ornithopters is to make them able to combine efficiently gliding and flapping phases, as birds do. Gliding allows the UAV to save energy and extend its flight endurance, but flapping is still necessary to increase altitude and to perform perching operations. Therefore, during the trajectory planning process, ornithopters should consider when to transition optimally between flapping and gliding, in order to save as much energy as possible.

In general, planning optimal trajectories for autonomous ornithopters is a complicated problem. First, these types of vehicles present nonlinear and complex dynamics that need to be taken into account

*[1]GRVC Robotics Lab, University of Seville, Spain. frodriguex@us.es, jcapitan@us.es, esanchezlaulhe@us.es, aollero@us.es

†[2]Departamento de Matemática Aplicada II, University of Seville, Spain. dbanez@us.es

‡*Corresponding author

[1]GRIFFIN is an Advanced Grant of the European Research Council (https://griffin-erc-advanced-grant.eu).

Figure 1: View of our ornithopter prototype.

when computing feasible trajectories. Second, the state space should include the vehicle's position, velocities and attitude, which are relevant for gliding and perching operations. Due to these complexities, there is a need for model-based but efficient methods that allow us to compute optimal trajectories with real-time performance. Some works use numerical methods for model-based trajectory planning Posa et al. (2014); Hoff et al. (2019). For instance, numerical solutions of the Navier-Stokes equations have been used Paranjape et al. (2012), but they are too expensive computationally for a real-time implementation. Other approaches use probabilistic motion planners Webb & van den Berg (2013); Karaman & Frazzoli (2010) integrating kinodynamic constraints or evolutionary algorithms Menezes & Kabamba (2016) but, again, tractable models are necessary not to exceed computational requirements.

Contrary to other related work, in this paper, we propose a novel optimization algorithm for energy-efficient trajectory planning with an autonomous ornithopter. Our approach can be used for online planning and it is based on the following novel strategy. Instead of generating nodes in a random way, we only consider a small set of selected waypoints, following two prescripts to prune states: (1) the optimal path lives into a specific space corridor and, (2) we can prevent redundancy by grouping the nodes and selecting one witness per group. This strategy allows us to tune the parameters of the pruning stage depending on the particular application and hence, it is a more efficient and accurate problem-specific approach.

We envision the use of flapping-wing UAVs for tasks like surveillance or inspection due to their ability to perform long-endurance flights. Therefore, our main objective is to compute online trajectories that minimize the energy consumption of the ornithopter. Those trajectories have to comply with the ornithopter dynamics, being thus flyable. Then, we assume the existence of lower-level algorithms to control the ornithopter tracking the computed trajectory. For example, control laws for stable longitudinal and lateral flight with actual flapping-wing UAVs in perching operations have been proposed Paranjape et al. (2013).

We compute a tree of dynamically feasible trajectories by using a nonlinear model for the ornithopter motion and applying segmented tail angles and flapping frequencies. The two flight modes of the ornithopter, i.e., flapping and gliding, are modeled with different aerodynamic coefficients, which implies a more complex nonlinear model that depends on the flight mode. Then, we run a heuristic tree search pursuing optimal solutions. In order to achieve online planning, the computational complexity is alleviated twofold: (i) we propose an ornithopter model with simplified dynamics, to make it computationally tractable; and (ii) some pruning operations to reduce the tree search space and keep it bounded. Even though we use in this paper a 2D model that constraints the ornithopter movement to a longitudinal plane, our algorithm is general and could be applied to 3D trajectory planning given a proper ornithopter model. Moreover, our heuristic solver finds approximate solutions with minimal energy consumption, but we also demonstrate the efficacy of those trajectories regarding final target state achievement.

In summary, our main contributions are the following:

- We propose a dynamic model for the 2D motion of our prototype ornithopter (see Figure 1). The model is nonlinear and complex, combining the aerodynamic behavior from both gliding and flapping phases. Nonetheless, we show how the model is computationally tractable for online trajectory planning.

- We contribute a new tree-based algorithm for the computation of trajectories that minimize energy consumption and that comply with the ornithopter's dynamics. Gliding and flapping operations are integrated for efficient trajectory planning.

- Our algorithm can compute online approximated solutions by means of fast heuristic operations that prune the tree.

- We design energy-efficient curves for the ornithopter to create corridors around them that guide the search procedure. The advantage is twofold: these corridors allow us to bound the

tree search space, hence improving the algorithm's efficiency; but at the same time, the solutions' quality is degraded as less as possible, as we assume the optimal solution to lie within those energy-efficient corridors.

- We run extensive computational experiments to analyze our algorithm and tune its key parameters. Moreover, we demonstrate its performance in comparison with another relevant approach from the literature and show a special case study for planning perching maneuvers.

The remainder of the paper is organized as follows: Section 2 surveys the related work; Section 3 introduces our trajectory planning problem; Section 4 describes the dynamic model for our ornithopter; Section 5 details our algorithm for trajectory planning; Section 6 presents computational experiments to select the values of the parameters; Section 7 shows experimental results to better assess the performance of our algorithm; and Section 8 discusses our results and explores future work.

2 Related Work

In the literature, there have been different attempts to design flapping-wing UAVs. Many of these works Chirarattananon et al. (2014); Sihite & Ramezani (2020); Qin et al. (2014) develop new aerodynamic models and specific controllers to operate the vehicles. In some cases, the design of the flapping mechanism is optimized to achieve more efficient trajectories. Other similar works focus more on controllers for specific maneuvers, like diving Rose et al. (2016) or perching Maldonado et al. (2020). Even though the aforementioned works address the problem of designing and controlling non-linear dynamic models for flapping-wing UAVs, most of them do not concentrate on efficient trajectory planning for medium distance flights, which is our goal.

Generally speaking, motion planning for UAVs is a different problem to ours but somehow related. In the literature, the terms motion planning and path planning are usually employed indistinctly to refer to the same problem: given a robot in a workspace with obstacles, find a collision-free path from an initial to a goal configuration.

Many methods for motion planning make use of the differential flatness of multicopter systems to generate optimal, continuous-time trajectories represented as polynomials Mellinger & Kumar (2011); Mueller et al. (2015); Oleynikova et al. (2016). It can be assured that those trajectories are dynamically feasible given simplified multicopter dynamics. Others Brescianini & D'Andrea (2018); Paranjape et al. (2015) use motion primitives to discretize the UAV state space into a connected graph. Then, standard graph search algorithms like A* can be used to find efficient solutions through the graph. However, the common assumptions made by the previous works do not hold for flapping-wing UAVs, where nonlinear, complex dynamics are of utter importance when planning trajectories. Therefore, instead of exploring motion planning approaches more tailored to collision avoidance, we focus on trajectory optimization methods able to take into account nonlinear dynamics in a computationally tractable manner.

In general, the trajectory optimization problem for UAVs consists of finding the sequence of control inputs that minimizes a certain cost index, such as the energy consumption or the flight time; but fulfilling at the same time constraints on the vehicle dynamics. Thus, trajectory planners use the UAV motion equations (typically differential equations) to provide as output time-indexed variables such as positions, velocities and accelerations. A survey on trajectory optimization for UAVs can be found in Coutinho et al. (2018).

A common approach for UAV trajectory planning are the numerical methods. These methods can be classified into direct and indirect methods. Direct methods rely on discretizing an infinite-dimensional optimization problem into a finite-dimensional problem, to apply then nonlinear programming solvers Posa et al. (2014). Indirect methods do the opposite, they first determine the optimal control necessary conditions for the problem, and then use a discretization method to solve the resulting equations Betts (2010). For instance, Dietl & Garcia (2013) propose a discrete-time optimization method (with fixed time-step) for ornithopter trajectory optimization, where the objective is minimizing travelled distance. Another discrete numerical framework for solving constrained optimization problems using gradient-based methods is presented in Wang et al. (2017). The technique is applied to energetically optimal flapping using frequency and pitching/heaving trajectories as optimization parameters. A longitudinal model for the dynamics of a bat-like prototype has also been proposed recently Hoff et al. (2019). The authors use that model with direct collocation to plan dynamically feasible trajectories in simulation and to track them with their actual prototype. The objective is reducing the control efforts by minimizing accelerations. The main concern with these numerical methods is that they can face computational and convergence issues for highly nonlinear problems, as

the one addressed in this paper. This makes them less suitable for online trajectory planning.

An alternative approach to the numerical methods are the probabilistic planners, like the *Probabilistic Road-Maps* (PRM) or the *Rapidly-exploring Random Trees* (RRT). These algorithms are able to tackle high-dimensional planning problems in a reasonable computation time by increasingly sampling the state space. Besides, they are probabilistically complete, i.e., they converge to a solution (if it exists) with a probability tending to 1. There are also versions like RRT* Karaman & Frazzoli (2011) that achieve optimal solutions in the same asymptotically manner.

Algorithms based on RRT* build a tree by connecting the samples from the state space through optimal trajectories (i.e., solving the two-point boundary value problem). However, computing feasible trajectories for kinodynamic systems is an issue. Some works have extended RRT* focusing on simple specific instances of kinodynamic systems Karaman & Frazzoli (2010). In Webb & van den Berg (2013), for example, they propose a kinodynamic RRT* that can cope with any system with controllable linear dynamics. They even apply the algorithm to nonlinear dynamics through linearization, but without convergence guarantees.

Other probabilistic methods deal with nonlinear systems more specifically. For instance, connecting tree nodes using trajectories based on splines that are optimized via a nonlinear program solver Stoneman & Lampariello (2014). Conversely, there exists the alternative of using exclusively control sampling to handle dynamic constraints, rather than resorting to a numerical two-point boundary value problem solver. This approach is followed by the variants *Stable-Sparse RRT** Li et al. (2016) and *AO-RRT* Hauser & Zhou (2016). Convergence is not proven for any of these RRT* modifications. Moreover, the aforementioned probabilistic planners try to be generic solvers. Contrary to that, we propose problem-specific heuristics that allow us to guide more quickly the search of energy-efficient trajectories for an ornithopter. It is should be note that although there are also metaheuristic algorithms Boussaid et al. (2013); Dokeroglu et al. (2019); Hussain et al. (2019), they are designed to solve a wide range of hard optimization problems with minor adaptation. Our problem is very specific from an engineering point of view, and we need to integrate complex and non-linear dynamics, which makes it hard to find a general metaheuristic that works properly.

In summary, most trajectory planning methods related to our application fit into one of the presented categories. Trajectory planning methods based on motion primitives can yield trajectories dynamically feasible, but they do not address non-linear models properly. Exact numerical methods for trajectory optimization can cope with complex, non-linear models, but they present computational issues that make them not recommendable for online planning. Last, probabilistic planners can be adequate to tackle generic high-dimensional problems with competitive computational cost. This is why we selected this alternative for comparison with our application-specific planner.

3 Problem Description

In this section, we introduce the optimization problem addressed in this paper, which aims to plan trajectories with an autonomous ornithopter, as well as the main assumptions we made.

We assume that we have an autonomous ornithopter with a known model of its dynamics. Then, we are interested in planning optimal trajectories to navigate the ornithopter from an initial to a target state. The goal is to compute the sequence of control actions that produces a trajectory connecting the two given states that: (i) is dynamically feasible; and (ii) minimizes the total energy consumed by the aircraft. This is done by combining flapping and gliding maneuvers. Moreover, we assume that the UAV is flying in an open space and, hence, we do not consider collisions with obstacles.

More specifically, let us define the *states* and control *maneuvers* for our problem:

Definition 3.1. *(Flight state) A flight state* $s = (x, z, u, w, \theta, q)$ *describes an ornithopter configuration in a given instant of time, where* x *and* z *are the positional values in the plane* XZ *of the Earth reference frame,* u *and* w *are velocity components in the body reference frame,* θ *is the pitch angle and* q *is the pitch angular velocity.*

Definition 3.2. *(Flight maneuver) A flight maneuver is a control action performed by the ornithopter during its flight at a given flight state. We consider two degrees of freedom to define the flight maneuvers: tail deflection, determined by the deflection angle* δ *(up and down); and wing flapping, determined by the flapping frequency* f *(including a zero value for gliding).*

According to the previous definitions, a trajectory consists of a sequence of interleaved flight states and flight maneuvers. Our trajectory planning problem is constrained to a 2D movement (XZ plane), as we

use a longitudinal motion model for the ornithopter [2]. In particular, we define the Earth reference frame as a global frame with the Z axis (pointing downwards) representing the ornithopter altitude and the X axis its longitudinal motion. We also define a body reference frame attached to the ornithopter with X_b pointing forward and Z_b downwards. Both reference frames, together with the state and control variables, are depicted in Figure 2. Finally, we make some additional assumptions to simplify the ornithopter dynamics and derive its model: (i) we use the hypothesis of punctual mass; (ii) we assume small flapping amplitudes and thin airfoils to model aerodynamics; and (iii) we consider the aerodynamics centers to be in a fixed position, as movements are of small amplitude.

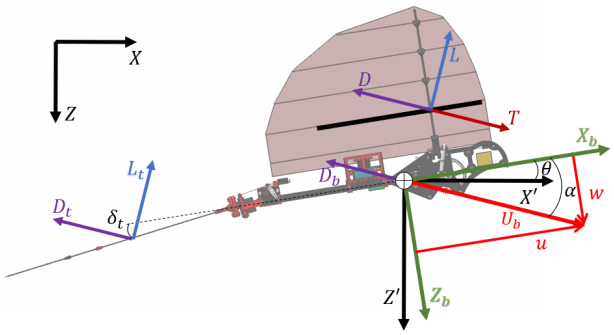


Figure 2: Schematics of the ornithopter with the forces acting on it. Axis XZ represents the Earth frame, axis $X'Z'$ a translation of the Earth frame, and axis X_bZ_b the body frame.

4 Ornithopter Dynamic Model

We describe in this section a motion model based on the one defined in Martín-Alcántara et al. (2019), and specifically developed for bio-inspired, flapping-wing UAVs. The model is used to describe the longitudinal motion of our ornithopter prototype.

4.1 Non-dimensional Newton-Euler equations

The Newton–Euler equations describe the combined translation and rotational dynamics of a rigid body, considering all existing forces. For a flapping-wing UAV as the one in Figure 2, the equations can be described as follows:

$$\begin{aligned} 2\mathcal{M}\frac{du}{dt} &= U_b^2[(C_L + \Lambda C_{Lt})\sin\alpha \\ &\quad + (C_T - C_D - Li - \Lambda C_{Dt})\cos\alpha] \\ &\quad - \sin\theta - 2\mathcal{M}qw \end{aligned} \tag{1}$$

$$\begin{aligned} 2\mathcal{M}\frac{dw}{dt} &= U_b^2[-(C_L + \Lambda C_{Lt})\cos\alpha \\ &\quad + (C_T - C_D - Li - \Lambda C_{Dt})\sin\alpha] \\ &\quad + \cos\theta + 2\mathcal{M}qu \end{aligned} \tag{2}$$

$$\begin{aligned} \frac{1}{\chi U_b^2}\frac{dq}{dt} &= C_L\cos(\alpha) - (C_T - C_D)\sin(\alpha) \\ &\quad + \mathcal{L}\Lambda[C_{Lt}\cos(\alpha) + C_{Dt}\sin(\alpha)] \\ &\quad - \mathcal{R}_{HL}[C_L\sin(\alpha) + (C_T - C_D)\cos(\alpha)] \end{aligned} \tag{3}$$

$$\frac{d\theta}{dt} = q, \tag{4}$$

[2]Note that our trajectory planning method could be applied to 3D as long as there were a complete 3D motion model for the ornithopter.

where α is the angle of attack, defined as $\alpha = \arctan(w/u)$, and $U_b = \sqrt{u^2 + w^2}$ the velocity module. $\mathcal{M}$, χ, Λ, $\mathcal{L}$ and $\mathcal{R}_{HL}$ are characteristic non-dimensional parameters of the UAV. These parameters are obtained by scaling the variables with the characteristic speed, length and time:

$$U_c = \sqrt{\frac{2mg}{\rho S}}, \qquad L_c = \frac{c}{2}, \qquad t_c = \sqrt{\frac{\rho S c^2}{8mg}}, \tag{5}$$

where m is the mass of the UAV, ρ the air density, S the wing surface, c the mean aerodynamic chord and g the gravity acceleration.

Figure 2 shows the forces acting on the vehicle, as well as the representative variables and reference frames used. In particular, L, T and D represent the lift, thrust and drag forces due to the wings; L_t and D_t the lift and drag from the tail; and D_b the drag due to the body friction. All these forces are considered in Equations (1)-(4) by means of the non-dimensional aerodynamic coefficients: C_L, C_T and C_D for the wing, C_{Lt} and C_{Dt} for the tail, and Li for the body. $Li = \frac{S_b}{S}C_{Db}$ is the Lighthill's number, with S_b the body surface and C_{Db} its friction drag coefficient. Section 4.2 explains how to compute the rest of coefficients for the aerodynamic forces of the wing and the tail.

4.2 Aerodynamic models

First, let us concentrate on the computation of the lift and thrust forces from the wings. Our model considers two modes of flight for the ornithopter: flapping and gliding; and the computation of the aerodynamic coefficients differs from one to another. In both cases, we make approximations assuming very thin airfoils. For gliding, the Prandtl's lifting line theory, combined

with unsteady aerodynamic terms, is used. For flapping, the Theodorsen solution Theodorsen (1935) defines the lift given a wing movement with the form $h(t) = h_0 \cos(2\pi f t)/2$, being h the vertical position of the reference wing chord during the flapping movement and h_0 the movement amplitude. In order to model the existing thrust (only existing in the flapping mode), we use the Garrick coefficient Garrick (1936) corrected by Fernandez-Feria (2016, 2017). Moreover, we consider finite wing effects by making aspect ratio corrections based on Ayancik et al. (2019), which leads to:

$$C_{L_{glide}} = 2\pi \left[\alpha + \left(\frac{1.5\dot{\alpha} - \frac{2l_w}{c} q}{U_b} \right) \right] \frac{\mathit{Æ}}{\mathit{Æ} + 2} \tag{6}$$

$$\begin{aligned} C_{L_{flap}} &= 2\pi \{ (kh_0) \left[G(k) \cos 2\pi f t \right. \\ &\quad \left. + F(k) \sin(2\pi f t) \right] + \alpha \} \frac{\mathit{Æ}}{\mathit{Æ} + 2} \\ &\quad + \pi k^2 h_0 \cos(2\pi f t) \frac{\mathit{Æ}}{\mathit{Æ} + 1} \end{aligned} \tag{7}$$

$$\begin{aligned} C_{T_{flap}} &= 4 (kh_0)^2 \sin(2\pi f t) [F_1(k) \cos(2\pi f t) \\ &\quad - G_1(k) \sin(2\pi f t)] \frac{\mathit{Æ}}{\mathit{Æ} + 2} - \alpha C_{L_{flap}}, \end{aligned} \tag{8}$$

where $\mathit{Æ}$ is the aspect ratio of the wing, given by $\mathit{Æ} = b^2/S$, being b the wingspan and S the surface. l_w is the distance between the center of gravity and the aerodynamic center of the wing, being positive when the center of gravity is behind the wing. $k = 2\pi f / U_b$ is the reduced frequency, $F(k)$ and $G(k)$ are the real and imaginary parts of the Theodorsen's function $C(k)$, and $F_1(k)$ and $G_1(k)$ are the real and imaginary parts of the function $C_1(k)$ defined in Fernandez-Feria (2016).

Regarding the lift force generated by the tail, as our bio-inspired design leads to triangular surfaces Thomas (1993), we use an approximation for delta wings:

$$C_{L_t} = \frac{\pi \mathit{Æ}_t}{2} \left[(1 - \varepsilon_\alpha)\alpha + \delta + \left(\frac{1.5\dot{\alpha} - \frac{2l_t}{c} q}{U_b} \right) \right], \tag{9}$$

where δ is the deflection angle of the tail, ε_α models the interference caused by the wing, and $\mathit{Æ}_t$ and l_t are, respectively, the aspect ratio of the tail and the distance between the center of gravity and the aerodynamic center of the wing, being defined in the same manner as it was for the wing. In order to consider near stall effects, we saturate all the lift coefficients (C_L and C_{L_t}) for angles greater than 10^o for the wing and 25^o for the tail.

Finally, we model the drags from the wings and the tail, C_D and C_{Dt}, as the addition of the constant friction drags, C_{D0} and C_{D0t}, and the induced drags, provided by:

$$C_{D_i} = \frac{C_L^2}{\pi \mathit{Æ}}, \qquad C_{D_{it}} = \frac{C_{L_t}^2}{\pi \mathit{Æ}_t}. \tag{10}$$

5 Ornithopter Segmentation-based Planning Approach

In this section, we present our method to solve the problem stated in Section 3, i.e., how to plan optimal trajectories for an ornithopter that are both dynamically feasible and energy-efficient. As discussed in Section 2, there exist in the literature numerical optimization solvers that can deal with nonlinear systems. They obtain dynamically feasible trajectories by discretizing and integrating the model dynamics, but they can suffer from computational complexity and convergence issues for highly nonlinear models. Graph-based approaches are an alternative where a graph with discrete, connected states is built in order to search for optimal trajectories. In particular, probabilistic planners sample the state space increasingly to build these graphs Karaman & Frazzoli (2011). If the objective is to generate trajectories that are dynamically feasible, the method must ensure that the sampled states are reachable, which can be complex for nonlinear systems.

In this paper, we propose a novel graph-based approach that builds a tree to search for energy-efficient trajectories. Instead of taking random samples from the state space, we segment the ornithopter's actions and we integrate its dynamics to generate and connect feasible states, which can later become nodes of the tree. We name our algorithm *OSPA*, which stands for *Ornithopter Segmentation-based Planning Approach*.

The general idea is as follows. In our approach, we consider a discrete set M with the possible flight maneuvers for the ornithopter. Then, given a fixed time step t_s and an initial state, we generate a discrete set of reachable flight states by integrating the ornithopter dynamics for time t_s and for each maneuver in M. Doing this iteratively, we build a tree T whose vertices (or nodes) are flight states, and each edge has associated the corresponding maneuver to navigate from one state (node) to the next one. Each edge has also associated the energy consumed by the corresponding transition maneuver between its vertices. We store at each node the energy consumption needed to reach it from its predecessor state. The

final goal is to find a path τ through the tree T that connects the initial and target states and that minimizes the total energy required by the ornithopter. Figure 3 illustrates an example on how to segment the trajectory of an ornithopter and the resulting flight states. A landing operation is achieved by a maneuver with flapping involved, preceded by two different maneuvers where the ornithopter is only gliding.

Figure 3: Segmentation of the ornithopter motion for a landing trajectory. Top view, sequence of flight states followed by a real bird. Bottom view, trajectory computed by our method, with three consecutive maneuvers along 250 meters, before landing. The trajectory connects the flight states by integrating the dynamic model.

Algorithm 1 provides an overview of the procedure followed by OSPA. The algorithm receives as input:

- the initial and final states s_0 and s_f, respectively;
- the discrete set of maneuvers M;
- the time step t_s;
- the pruning parameters k_d and k_w.

The set M is generated by combining a discrete set of flapping frequencies with a discrete set of tail angles, i.e., $M = D \times F = \{(\delta, f) \mid \delta \in D \wedge f \in F\}$. The initial state is inserted as the tree's root. Then, at each iteration, all leaf states are expanded with all possible maneuvers. Given a state s_i, a flight maneuver defined by a tail angle δ and a frequency f, and a time step t_s, the function $\texttt{ODE}(s_i, \delta, f, t_s)$ integrates the equations stated in Section 4 for time t_s, to produce a flight state that becomes a new node of the tree. The states that are eventually inserted into the tree are selected according to two pruning procedures that will be detailed in Section 5.1.

Algorithm 1: OSPA

```
   Input  : (s_0, s_f, M, t_s, k_d, k_w)
   Output: τ*
1  tree ← Tree()
2  tree.root ← s_0
3  leaves ← GetLeaves(tree)
4  corridor ← GetCorridor(s_0, s_f)
5  while leaves.length > 0 do
6      states ← List()
7      for s_i in leaves do
8          for (δ, f) in M do
9              s' ← ODE(s_i, δ, f, t_s)
10             if GetDist(corridor, s') ≤ k_d and
                s'.x ≤ s_f.x then
11                 s'.parent ← s_i
12                 states.Add(s')
13         end
14     end
15     partitions ← GetPartitions(states, k_w)
16     for c in partitions do
17         s* ← GetOptimalState(c)
18         s_i ← s*.parent
19         s_i.AddChild(s*)
20     end
21     leaves ← GetLeaves(tree)
22 end
23 τ* ← SearchOptimalPath(tree, s_f)
24 return τ*
```

We assume that the ornithopter has forward motion in the XZ plane, so the method generates states with increasing X-axis values from one time step to the following. The tree computation ends when all reached states have greater X-axis value than the final state s_f. In that case, the optimal sequence of maneuvers is returned and the algorithm terminates. This is done by the function `SearchOptimalPath()`, that computes the tree path τ^* with minimum energy consumption. The last node of the solution τ^* is chosen as the one with lowest energy consumption among those within a *tolerance* distance (set by the user) around s_f.

Figure 4 illustrates how Algorithm 1 works with 3 maneuvers ($|M| = 3$) and 2 witness nodes ($k_w = 2$). Crosses represent the pruned nodes and dots the remaining nodes in the tree. The red line is the solution

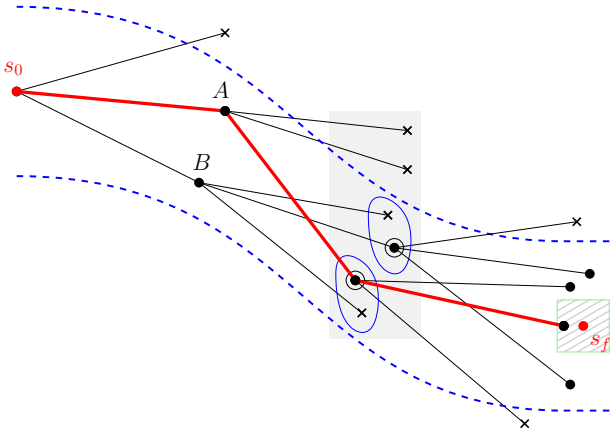


Figure 4: Graphical representation of the procedure followed by OSPA to create the tree and search for optimal trajectories.

trajectory output by OSPA, given the initial and final states, s_0 and s_f; while the dashed blue lines indicate the corridor around the reference curve. An entire iteration of Algorithm 1 is demonstrated in the gray rectangle. First, the nodes within the rectangle are generated from the nodes A and B, by using using an ODE integrator applying each of the 3 maneuvers, and pruning those nodes out of the corridor (lines 6 to 14 of Algorithm 1). Second, two (k_w) partitions are created with the remaining nodes (line 15 of Algorithm 1). Third, for each partition a witness node with the best energy cost (the one within a circle) is kept in the tree and the others pruned (lines 16 to 20 of Algorithm 1). Then, another iteration of the algorithm would be performed. The whole algorithm finishes when a node falls within a target region around the final state (dashed square), and the optimal trajectory (red path) is returned (line 23 of Algorithm 1).

Now let us show a small numerical example using the data corresponding to the trajectory depicted in Figure 3. In that example, the input of the algorithm is the following: $s_0 = (0m,\ 0m,\ 4.26m/s,\ 0m/s,\ 0°,\ 0°/s)$, $s_f = (250m,\ 95m,\ 0\ m/s,\ 0\ m/s,\ 0°,\ 0°/s)$, $t_s = 12s$, $k_d = 15m$, $k_w = 20$, and as set of possible maneuvers M, the one in Table 5. After running OSPA, the output is a sequence of interleaved flight states and flight maneuvers. Specifically, the three selected maneuvers (δ, f) are: $\{(-2°,\ 0Hz), (-1°,\ 0Hz), (0°,\ 0.16Hz)\}$. Then, the pseudo-optimal trajectory to be followed by the ornithopter is computed by integrating its dynamic model for each time step, using the control inputs of each corresponding flight maneuver. The (x, z) waypoints in meters for this numerical example (see Figure 3) are: $\{(0, 0), (90.65, 12.94), (145.57, 69.27), (249.99, 96.31)\}$. In terms of precision, note that the ornithopter would end up at a 1.31 m distance from the target state with this solution.

OSPA searches for optimal trajectories in terms of energy and, hence, whenever a new node is added, the algorithm computes and stores the accumulated energy at that node. In order to model the energy consumed by the ornithopter performing a certain maneuver for a time step t_s, we use the following formula:

$$E = t_s(K_{aero} f^3 + c_r). \tag{11}$$

The first term represents the dominant energy consumption, which is due to the flapping wings. It has been empirically proven that this consumption is proportional to the cube of the flapping frequency T. A. Nguyen et al. (2016), with a constant coefficient K_{aero} that depends on several physical characteristics of the ornithopter, such as the wings' profile, their inertia and their movement amplitude. However, as modelling all those effects precisely is complicated, we opted for estimating the value of K_{aero} empirically [3]. The second term models the residual energy consumption c_r when the ornithopter is not flapping, mainly due to the onboard electronics. While gliding, we measured empirically that, for our ornithopter, the cost of moving the tail was negligible compared to the average consumption of the electronics. Therefore, we consider the cost c_r constant [4]. As expected, it is important to note that Equation 11 indicates that the main energy consumption is produced by the flapping maneuvers. While gliding, the energy efficiency is just related with the temporal length of the maneuver.

Finally, let us analyze the size of the tree T generated by our method. If we have $|M| = |D| \times |F| = m$ different maneuvers that can be selected at each iteration, the whole tree construction without pruning operations takes $O(h^{rm})$ time, where h is the tree height [5] and r is the average time needed by the integrator `ODE()`. Note that the height h depends on the time step t_s: the smaller t_s, the larger the tree will be to reach the final state. OSPA can achieve energy-efficient trajectories if we take values of the time step short enough and we use enough number of

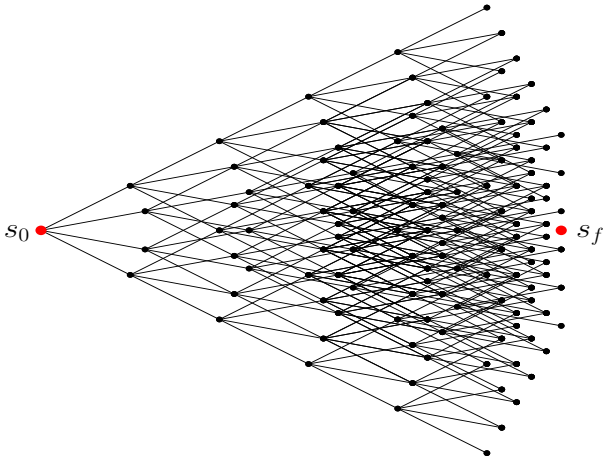
[3] All the experiments in this paper used a value $K_{aero} = 2.5\ W/Hz^3$, obtained empirically for our ornithopter prototype.

[4] We estimated empirically an upper bound of $c_r = 5W$.

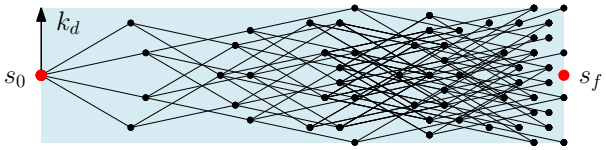
[5] The height of T is the maximum distance (number of edges) from the root to any node in T.

maneuvers. However, the computational complexity increases exponentially with the number of maneuvers, and a more reduced set of states may be enough to achieve competitive approximate solutions to the final state. Therefore, in the next section we propose two pruning procedures which alleviate the computational cost of the algorithm and that yield an efficient planner for short and medium distance flights.

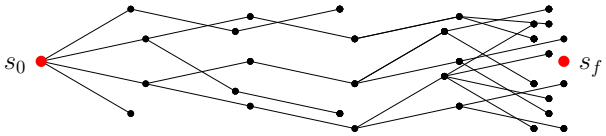
5.1 Tree reduction



(a) Example of an exhaustive tree T considering 4 maneuvers. Nodes are represented as positional values in the XZ plane.



(b) Tree T' after applying to T the pruning operation based on a limited corridor. k_d is the width or clearance of the corridor.



(c) Tree T'' after applying to T' the pruning operation based on clustering. $k_w = 5$ is the upper bound on the number of considered partitions at each step (the last layer of leaves is not pruned).

Figure 5: Overview of the pruning operations in the OSPA planner.

OSPA includes two specific procedures to reduce the tree size and speed up the algorithm. First, we reduce the original tree T to a pruned tree T' that only keeps nodes whose position is close to a hypothetical optimal trajectory $\mathcal{P}^*$. This pruning operation relies on the idea that the tree T' will produce optimal solutions similar to those in T, as long as the pruned nodes are not in the vicinity of $\mathcal{P}^*$. In principle, this optimal trajectory is unknown, but we propose a parametric curve to estimate $\mathcal{P}^*$ that acts as guide in the tree pruning. Second, considering many states that are very close in the tree may lead to a redundant calculation of similar trajectories. This fact motivates our second procedure to reduce further the tree size, where we obtain a new tree T'' by creating partitions with the nodes in T', and keeping only a witness node for each partition. Note that our pruning procedures assume that there is an optimal trajectory $\mathcal{P}^*$ which presents a robust clearance, that is, the nodes in T'' are enough to compute a near optimal trajectory. We support this assumption on the experimental study of Section 6.

Figure 5 shows an example of an original tree T and the effect of applying our pruning operations to obtain the trees T' and T''. Note that some of the leaves in T'' are still irrelevant flight states for a suitable solution, since they are far away from the final state. Thus, OSPA only considers for the selection of the last state those within a desired tolerance distance to the target. In the following, we elaborate on our two pruning operations to perform the incremental construction of T''.

5.1.1 First pruning operation: the corridor

The first procedure to prune the tree consists of imposing physical constraints on the admissible trajectories. We speed up the operations in the algorithm avoiding pathological states that would be unlikely. More precisely, we define a corridor region C connecting the initial and final states, and we only consider tree nodes within that corridor, discarding those out of the corridor. The key idea is building C in such a manner that the (unknown) optimal trajectory $\mathcal{P}^*$ is likely to lie within that corridor.

We generate the corridor C as follows. First, we take a parametric curve on the XZ plane that connects the initial and final states, and we adjust its parameters so that the curve is likely to resemble the optimal trajectory $\mathcal{P}^*$. Let this *reference curve* be denoted as $\widehat{\mathcal{P}^*}$, then C is defined as the region of points in the XZ plane whose minimum Euclidean distance to $\widehat{\mathcal{P}^*}$ is not greater than a parameter k_d. The tree T' results from pruning all the nodes in the original T that fall outside C. Figure 5b illustrates an example for a line segment $\widehat{\mathcal{P}^*}$ connecting the initial and final states. The functions `GetCorridor()` and `GetDist()` in Algorithm 1 compute the corridor C and the minimum Euclidean distance of a flight state

to $\widehat{\mathcal{P}^*}$, respectively.

Recall that the optimal $\mathcal{P}^*$ is unknown and that we are interested in computing a fairly reasonable estimation $\widehat{\mathcal{P}^*}$, so that OSPA finds solutions that are close to the optimum. An option to learn those curves would be a totally bio-inspired approach, i.e., observing bird flights in order to borrow the type of curves they follow; as they are assumed to be energy-efficient. This would require gathering large datasets which are not easy to obtain in general, so we followed a different empirical strategy to design our reference curves. Particularly, we ran extensive computational experiments in different situations, using complete trees computed by OSPA without pruning in order to search for trajectories with the lowest energy consumption. We observed that trigonometric curves were well suited to these optimal trajectories. Therefore, we build our reference curve $\widehat{\mathcal{P}^*}$ as a special type of trigonometric curve that connects the initial and final states, and we assume that it is a fairly good approximation of the actual optimal trajectory. The procedure to compute and adjust these reference curves is further detailed in Section 6.2.

5.1.2 Second pruning operation: witness states

The second procedure to prune the tree focuses on preventing redundancy in order to improve further the time complexity, but without significantly degrading the solution quality. The idea is the following. Note that at each tree expansion step, the neighboring nodes generate M new flight states each, and some of them may be similar. Indeed, the density of close states will grow as the tree height increases (see example in Figure 5a). Therefore, we implement a simple partitioning technique to group close nodes and select only the best ones. At each iteration of the tree building process, we partition the new leaf nodes into k_w disjoint subsets. Since the nodes at the same tree level present more significant differences in the vertical coordinate than in the horizontal one, the z coordinate is used to order the leaves and split them into k_w equally separated sets. This is done by the function `GetPartitions()` in Algorithm 1. As all the nodes in each partition will represent close flight states, we keep alive only a *witness* node for each partition, throwing away the rest. Since we search for energy efficiency, the selected witness nodes are those with the minimum accumulated energy (function `GetOptimalState()` in Algorithm 1). Figure 5c illustrates an example of this pruning operation.

Finally, note that this witness pruning operation alleviates considerably the computational complexity of OSPA, as the number of inserted leaf nodes at each iteration of Algorithm 1 is bounded. More specifically, the original computation time to build the tree, $O(h^{rm})$, is reduced to $O(hk_w m(r+\log(k_w m))$, which is almost linear in the number of maneuvers. The second term in the time complexity is due to the leaves ordering in z performed by `GetPartitions()`.

Parameter	Interval
δ	$[-6, 0]^\circ$
f	$[0, 6]\ Hz$
t_s	$[8, 20]\ s$
k_d	$[10, 25]\ m$
k_w	$[10, 40]$

Table 1: Used intervals for the parameter values in the tuning experiments.

5.2 Planning in dynamic scenarios

OSPA can be used for trajectory planning in dynamic scenarios, recomputing trajectories online as the environment changes. For example, the approach can be easily modified to face collision avoidance. The idea is to integrate some procedure for collision checking within the planner. More specifically, in line 10 of Algorithm 1, assuming a map representation of the environment with the existing obstacles, we could add a new condition to avoid the generation of nodes within a fixed distance from the obstacles. In cases where the obstacles overlap the entire corridor, its width could be increased to ensure a solution. In this paper, we have not explored these options for collision avoidance, as we consider it out of the scope and we leave it for future work.

6 Selection of Parameters

In this section, we describe a series of computational experiments to analyze the effects of the key parameters in OSPA. We aim to obtain the most appropriate values for: the time step t_s, the reference curve $\widehat{\mathcal{P}^*}$, the set of maneuvers M, the threshold k_d to compute the corridor C, and the parameter k_w to create partitions. All experiments were run with a version of OSPA coded in Python 3.7 [6] on a CPU with a 1.60 GHz processor and 8 GB RAM. We used the `odeint` module from the *SciPy* library for numerical integration.

[6] The code is available at https://github.com/fragnarxx/kinodynamic-planning.

Table 1 shows the used parameter intervals for all experiments. We consider a discrete set M of maneuvers that results from the combination of 7 tail angles uniformly selected between the bounds in Table 1 and the frequency values $\{0, 4, 5, 6\}$ Hz, hence $|M| = 28$. These frequency values were empirically selected to provide positive net thrust and consider the mechanical engine limitations of our ornithopter prototype. The interval for the tail deflections was selected to ensure that the ornithopter flies without reaching aerodynamic stall, which would not be interesting in general flight conditions.

$\mathcal{M}$	Λ	$\mathcal{L}$	$\mathcal{R}_{HL}$	χ
6.85	0.278	-15.5	1.92	0.0132
C_{D_0}	$C_{D_{0t}}$	$\mathcal{AR}$	$\mathcal{AR}_t$	Li
0.018	0.021	4.44	2.35	0.0051

Table 2: Values for the dimensionless characteristic parameters of the ornithopter.

U_c	L_c	t_c
4.26 m/s	0.135 m	0.0317 s

Table 3: Values for the characteristic dimensions.

In all experiments, we used the physical properties of our actual prototype in Figure 1 to determine the parameters of the dynamic model from Section 4. Table 2 depicts the characteristic dimensionless parameters, while Table 3 shows the characteristic dimensions of the problem. Finally, we built the same set of 80 scenarios for all experiments, with the ornithopter starting at $(0, 0)$ position in the XZ plane, and $0°$ for the initial and final pitch angles. The final state positions are taken from a uniform grid within the rectangle $R = \{(x, z) : 200 \leq x \leq 250, -20 \leq z \leq 100\}$. Recall that positive and negative values for z mean descending and ascending, respectively.

6.1 Time step

The first critical parameter in OSPA is the time step t_s, as it affects the computation time, as well as the quality of the solution. In order to select an adequate value for this parameter, we ran OSPA at the 80 experimental instances using the set of maneuvers M and varying the value of t_s. Figure 6 shows the average results as t_s increases. We consider three metrics: the computation time to plan a trajectory, the total energy E consumed throughout the trajectory, and the *accuracy* Δ, defined as the Euclidean distance between the last node of the trajectory and the target[7]. As expected, smaller values of t_s yield a larger computation time but a higher accuracy, decreasing the error with respect to the target. According to this trend, we selected a value of $t_s = 12$ s as a trade-off solution, since it favors energy consumption but, at the same time, produces acceptable accuracy values.

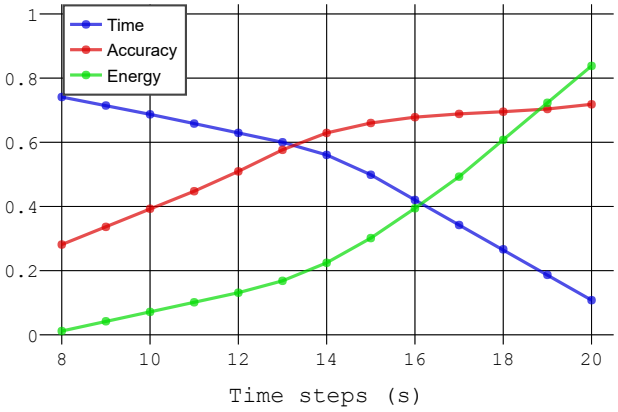


Figure 6: Performance results varying the time step t_s. Average values over 80 experiments are shown for the computation time, the accuracy with respect to the target state (Δ) and the solution energy (E). The values are scaled to the $[0, 1]$ interval.

6.2 Reference curve

In this section we describe the empirical process to design the reference curve $\widehat{\mathcal{P}^*}$ in the planner. Recall that this curve is used to define a corridor C that guides the tree search. As we prune all the states with distance greater than k_d from the reference curve, we should select a curve that is as close as possible to the typical waypoints in optimal trajectories. For that, we ran the 80 experiments obtaining optimal trajectories using OSPA with $t_s = 12$ s and the set of maneuvers M, without pruning operations. We obtained these optimal trajectories using exhaustive trees containing an exponential number of reachable states, so we believe that they represent a good approximation of the optimal solution of the continuous problem. This computation is expensive to be performed online, but we ran it offline to have a good reference of the optimal behavior of our ornithopter, energetically speaking.

As a first option, we attempted a simple straight line connecting the origin and the target point. Then, we measured the *Maximum Error* (ME) and the *Root Squared Mean Error* (RSME) between the waypoints

[7]Recall that the nodes are vectors including positions, angles and velocities.

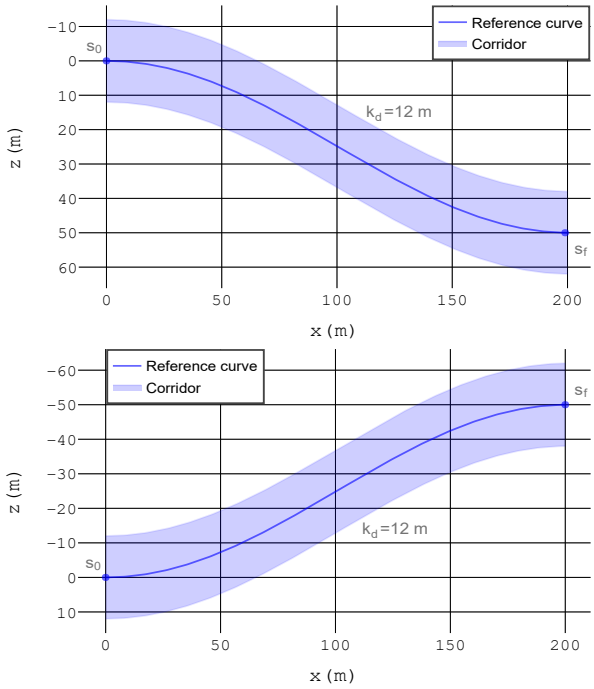


Figure 7: Examples of the reference curve for two cases with the ornithopter descending (top) and ascending (bottom). The corresponding corridor C is also shown for $k_d = 12$.

obtained in the trajectories computed by OSPA and the straight line, obtaining the average values 15.40 m and 9.24 m, respectively. However, we observed in our results that trigonometric curves were well suited to the generated trajectories in the experiments. Therefore, we built our reference curve $\widehat{\mathcal{P}^*}$ as a scaled cosine connecting the initial and final states. The formula for this curve depends on the distance between the initial (x_0, z_0) and final (x_f, z_f) positions. Provided that $x_d = x_f - x_0$ and $z_d = z_f - z_0$, the formula for the proposed reference curve is:

$$\widehat{\mathcal{P}^*}(x) = \frac{1}{2}\left[z_d + z_d cos\left(\pi + \frac{\pi x}{|x_d|}\right)\right]. \qquad (12)$$

Figure 7 depicts examples of this reference curve and the corresponding corridors, resembling typical optimal trajectories obtained with our ornithopter model. Moreover, we computed the ME and RMSE metrics for our 80 experiments, obtaining the average values 15.30 m and 8.92 m, respectively. As expected, we determine that our trigonometric choice of reference curve is also closer to the optimal trajectories in the experiments than straight lines, and we use it in the remaining experiments.

ξ^*	$\lvert M_r\rvert$	Δ	E (W)
0	$28 = \lvert M\rvert$	0.49 ± 0.06	3979 ± 343
0.01	23	0.51 ± 0.07	3693 ± 298
0.02	17	0.53 ± 0.07	3393 ± 317
0.03	10	1.31 ± 0.08	2268 ± 254

Table 4: Results with different sets of maneuvers. Average values and deviations over 80 simulations are shown for the accuracy (Δ) and the energy (E).

6.3 Study of maneuvers

We have already defined an initial set of maneuvers ($|M| = 28$) that make sense from the physical point of view of the ornithopter to plan trajectories. However, as it was explained in Section 5, the number of maneuvers highly affects the computation time of OSPA. Therefore, we present in this section a statistical analysis to find out which maneuvers are really more useful, with the purpose of further reducing the final set. For this study, according to our previous time step analysis, we fixed $t_s = 12$ s. Then, we ran OSPA (without pruning operations) at the 80 scenarios to find optimal trajectories. We can compute the occurrence rate of a maneuver as follows:

$$\xi(m_i) = n_i/|n^*|, \qquad (13)$$

where n_i is the number of times that maneuver m_i was selected in any of the optimal solutions, and n^* is the total count of maneuvers in all the optimal solutions. Larger values of ξ indicate that the maneuver is commonly used, while lower values correspond to rarely used maneuvers. Therefore, we select our reduced set of maneuvers M_r by defining a threshold value ξ^* and removing the maneuvers with a lower occurrence rate, i.e., $M_r = \{m_i : m_i \in M, \xi(m_i) \geq \xi^*\}$.

We tested different values of ξ^* to create the subset M_r, and we solved again the 80 scenarios with each resulting M_r. Table 4 depicts average results for the accuracy and the energy as ξ^* increases. The computation time is not included because it is not representative (the experiment uses the whole set of maneuvers M without pruning operations and, hence, it is not efficient). According to the results, we took $\xi^* = 0.02$, as it offers a fairly good trade-off, considerably reducing the set of maneuvers without a great degradation in terms of energy and accuracy. Table 5 shows the final subset M_r of maneuvers, obtained with the selected threshold $\xi^* = 0.02$. From now on, we will consider this set M_r of maneuvers in all the experiments.

ξ	δ (°)	f (Hz)
0.111	−2	0
0.109	0	4
0.077	0	5
0.076	−3	0
0.074	−6	0
0.072	−5	0
0.057	−4	0
0.053	−1	0
0.053	0	6
0.045	−2	6
0.026	−3	5
0.026	0	0
0.025	−4	4
0.023	−5	4
0.022	−3	4
0.021	−6	4
0.021	−4	5

Table 5: Occurrence rate for all the maneuvers included in the selected subset M_r, obtained with a threshold $\xi^* = 0.02$.

6.4 Parameters k_d and k_w

Given the adjusted values for the time step, the reference curve and the set of maneuvers, we can tune the parameters k_d and k_w for the pruning operations. We studied the performance of our algorithm varying the values of these two key parameters. For that, we ran OSPA in the 80 scenarios, using the maneuvers in the set M_r, the reference curve in Equation 12 and a time step $t_s = 12$ s. Then, we analyzed the average values for the metrics: accuracy Δ, energy consumption and computation time.

Table 6 depicts the average results of our experiment for some representative values of the parameters. We tested more values within the intervals in Table 1 but we got the same trend, so they are not included here for the sake of brevity. As expected, the larger the values of k_d and k_w, the more node states are explored, and the better the quality of the solutions is, both in terms of energy and accuracy. The computation time also increases though. Depending on the acceptable level of accuracy and the available time budget for OSPA for a given application, different values of k_d and k_w could be selected.

6.5 Multi-resolution approach

Finally, we performed another computational experiment of interest to test a multi-resolution approach. OSPA builds a tree generating new states by integrating the system with different maneuvers during a fixed time step t_s. However, as we have already discussed, this parameter can be key in different aspects. Thus, we also tested a multi-resolution approach, consisting of using a set of several possible values for the time step instead of a fixed one, i.e., $t_s \in \mathcal{T} = \{t_1, \ldots, t_n\}$. At each leaf node, new nodes are generated for each maneuver using all the time steps in $\mathcal{T}$, hence increasing the branching factor and the computation time, but also the searching space granularity.

k_d, k_w	Δ	E (W)	$Time$ (s)
$10, 15$	0.95 ± 0.06	4400 ± 495	137 ± 8
$15, 25$	1.53 ± 0.07	4092 ± 424	239 ± 14
$20, 20$	0.56 ± 0.08	4270 ± 436	295 ± 18
$25, 35$	0.52 ± 0.08	4590 ± 447	443 ± 16

Table 6: Average results and deviations for some representative values of k_d (in meters) and k_w. Accuracy, energy and computation time are included.

$Steps$	Δ	E (W)	$Time$ (s)
$\mathcal{T}_1$	1.53 ± 0.07	4092 ± 424	239 ± 14
$\mathcal{T}_2$	2.01 ± 0.15	3962 ± 439	3150 ± 86
$\mathcal{T}_3$	2.22 ± 0.16	3927 ± 434	7427 ± 243

Table 7: Average results and deviations over 80 simulations for the multi-resolution approach with different sets of time steps.

We ran an experiment with the 80 scenarios, using the reference curve in Equation 12 and the reduced set of maneuvers M_r. We set the pruning parameters to $k_d = 15$ m, $k_w = 25$, since these values offer a good trade-off in terms of solution quality and computation time, according to Table 6. The multi-resolution approach was tested with three sets of time step values: $\mathcal{T}_1 = \{12s\}$, $\mathcal{T}_2 = \{11s,\ 13s\}$, and $\mathcal{T}_3 = \{10s,\ 12s,\ 14s\}$. Table 7 shows the average results for the accuracy, the energy and the computation time. Based on these results, we conclude that multiple time step values can improve the energy value with a slight degradation in the accuracy Δ. However, the improvement in the solution quality is not that significant in comparison with the outstanding increase in the computation time, which made us discard this multi-resolution approach.

7 Experimental Evaluation

This section shows some experimental results to assess the performance of OSPA. The problem at hand

is very specific from an engineering point of view, and we need to integrate complex and non-linear dynamics, so it is difficult to find a general metaheuristic that works. In contrast, we have compared our algorithm with methods focused on trajectory optimization problems. First, we compare OSPA with a competitive probabilistic planner from the literature. Then, we depict the results of a special case study to illustrate how OSPA can also be applied to plan in real time ornithopter trajectories for perching.

7.1 Comparison with a sampling approach

We believe that the probabilistic planning approaches are the most suitable alternative in the state of the art to tackle online trajectory planning for ornithopters. In particular, we have selected the recently published method AO-RRT Hauser & Zhou (2016), which is, to the best of our knowledge, one of the most competitive in the literature. They propose a meta-algorithm that can integrate any feasible kinodynamic planner as a subroutine, and they outperform other sampling-based planners of the state of the art. Another interesting comparison would have been against purely numerical methods (direct and indirect) providing optimal solutions for trajectory optimization (as explained in Section 2). However, we have tried a couple of these methods [8] for non-linear control optimization in our problem, and the computation times were prohibitive to obtain practical results, due to the highly nonlinear dynamic model of our ornithopter.

AO-RRT is an approach to adapt the RRT* probabilistic planner Li et al. (2016) to cope with non-linear dynamics constraints. As the classical RRT*, AO-RRT receives as input an initial state and it samples the state space randomly, in order to generate a dynamically feasible tree that eventually will reach the vicinity of the target state. The strategy consists of generating multiple states by integrating numerically the nonlinear system dynamics, using control inputs that are randomly selected. The algorithm stops when a computation time limit is reached and it takes as solution the path toward the target state that minimizes a given objective function. In particular, we adapted the original AO-RRT [9] to use our ornithopter model and the energy consumption as cost function. Moreover, we included a modification to sample the control inputs from a discrete set. Otherwise, if we let the algorithm select the frequency values uniformly within the interval $[0, 6]$ Hz, the theoretical probability to pick $f = 0$ is zero. This would preclude us from performing gliding maneuvers, significantly increasing the energy consumption. Therefore, we made the algorithm select the control frequencies and tail angles randomly from the discrete set M.

We created a set of 324 simulated scenarios to compare the two approaches. For each simulation, we took $(x_0, z_0) = (0, 0)$ as the position of the initial state and we selected a target state uniformly within the intervals $x \in [130, 250]$ m and $z \in [-20, 100]$ m. Regarding the target tolerance, we defined a square of 6-meters side centered at the chosen target point as the region to consider the goal as reached. The selected parameters for OSPA were $t_s = 12$ s, $k_d = 10$ m, $k_w = 15$ (to achieve computation times below 200 s) and M_r as the set of maneuvers. For AO-RRT, we fixed the maximum computation time to 200 s, and we checked that OSPA was able to run each of the simulated scenarios within a time lower than 200 s. Another relevant parameter for AO-RRT was the time step between two consecutive control actions, because we realized that it affected the results significantly. We tested different values in our comparison to make it fairer.

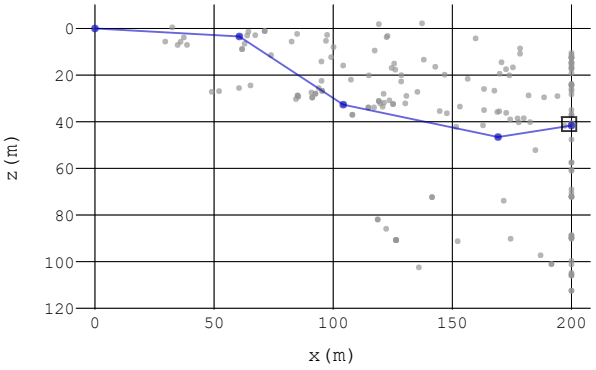
We use two metrics to evaluate the methods: the average energy consumption of the solution trajectory over all the scenarios; and the *precision rate*, which is the percentage of cases where a feasible solution was found within the 6-squared-meter tolerance region. Figure 8 shows the output of one of the simulated experiments for AO-RRT and OSPA, including the obtained solutions and the sampled waypoints in the trees. The figure depicts the advantages of the search strategy in OSPA: AO-RRT samples random states that are more uniformly distributed in the space, while in OSPA, thanks to the reference curve, the states are more concentrated around the optimal solution. Figure 9 depicts the resulting metrics for the complete comparison. As expected, AO-RRT presents results that are degrading in terms of energy consumption and precision as the time step value increases. It can also be seen that the precision rate of OSPA is always greater, and that OSPA achieves more efficient trajectories in terms of energy. Thus, OSPA outperforms the state-of-the-art AO-RRT algorithm for ornithopter trajectory optimization.

7.2 A case study: landing for perching

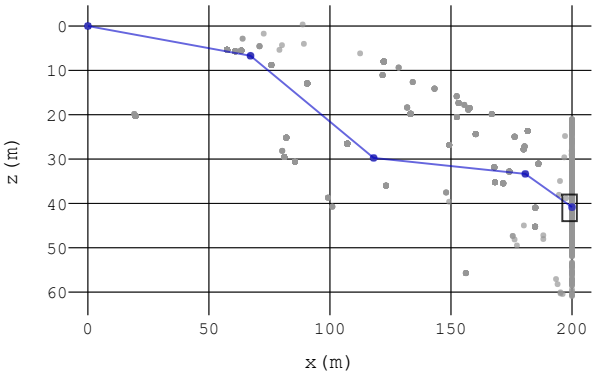
OSPA is thought for ornithopter navigation in short and medium distance flights, and it aims to optimize the energy consumption during the operation. For

[8] We have used the Forces Pro Zanelli et al. (2017) and ACADO toolkits Houska et al. (2011).

[9] We have used the open-source Python implementation provided by the authors.



(a) AO-RRT



(b) OSPA

Figure 8: Sampled waypoints in the tree (grey) and solution (blue) for an example experiment. The square denotes the tolerance region around the target state.

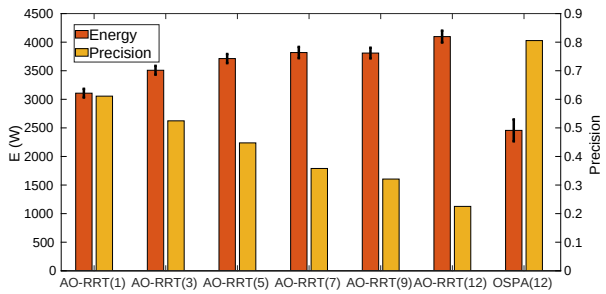


Figure 9: Comparison of the energy and the precision between OSPA and AO-RRT for different time step values (in parentheses).

those scenarios, we can compute a trajectory online and then use nonlinear controllers at a high rate for trajectory following. Now, we show a particular case study to demonstrate how OSPA can also be used for landing maneuvers when the ornithopter is going to perch.

A perching maneuver was studied analytically in Crowther (2000), consisting of two phases: a gliding phase (almost horizontal in this case) and a rapid pitch up to a high angle of attack. The example in Figure 3 shows various stages of this type of maneuver. Perching maneuvers require computing trajectories quite fast, as the ornithopter has no much time for reaction. Also, it is more relevant to land closer to the target rather than to reduce the energy cost, as perching requires high accuracy, particularly if the available area for landing is small. Therefore, in this experiment, we measure the Euclidean distance to the target state considering only the ornithopter position, as the velocity and the attitude may be more sensitive and they could be adjusted by lower-level attitude controllers at a high rate.

We created a set of experiments to test the trajectories computed by OSPA to land at a certain spot. Typically, the initial and target positions should be relatively close and the altitude be descendant. Therefore, we set the ornithopter initially at the origin of coordinates with a zero pitch, and the target point at a 10-meter longitudinal distance with a z coordinate ranging between 2 and 5 m. Recall that positive z values mean descending.

Since we pursue lower computation times when planning perching trajectories, we refined the set of maneuvers to be considered by OSPA. We experimentally selected a new reduced set of maneuvers for perching M_p, specially focusing on those with particular interest for that particular operation. To do this, we ran our set of experiments and we selected M_p following the same procedure as in Section 6.3. In particular, we used a probability threshold $\xi^* = 0.03$, and we obtained the reduced set M_p shown in Table 8. It can be seen that most of these maneuvers involve gliding, as it was demonstrated in Cory & Tedrake (2008) for optimized perching maneuvers.

We set OSPA parameters to $t_s = 1$ s, $k_d = 2$ m and $k_w = 4$. Table 9 shows the results for the performed experiments. It is important to remark that, in all the cases, the planning time is around 1 s, and the error distance with respect to the target, below 0.05 m. Interestingly, the optimal strategy yielded by OSPA was similar to the profile showed in Figure 3, i.e., a gliding phase followed by a pitch up with maximum upward elevation deflection. This indicates that OSPA can be a valid approach in practice to compute

ξ	δ (°)	f (Hz)
0.079	−1	0
0.107	−2	0
0.122	−3	0
0.081	−4	0
0.090	−5	0
0.093	−6	0
0.115	0	4
0.057	0	5
0.033	0	6

Table 8: Used maneuvers for perching planning. This set M_p was obtained with a threshold $\xi^* = 0.03$.

z_f (m)	Error (m)	Time (s)
2	0.049	1.09
2.5	0.005	1.08
3	0.021	0.98
3.5	0.008	1.10
4	0.042	1.08
4.5	0.017	1.11
5	0.033	0.95

Table 9: Results for perching experiments using OSPA. The final point is located 10 meters away in longitudinal distance, and at different altitudes z_f. The error distance to the target position and the computational time are shown.

trajectories to approach a landing area where the ornithopter wants to perch.

8 Discussion and Future Work

This paper proposed OSPA, a new algorithm for kinodynamic planning of trajectories for autonomous ornithopters. The method is able to compute energy-efficient trajectories in an online fashion, combining gliding and flapping maneuvers. OSPA builds trees dynamically feasible and it runs pruning operations to efficiently plan trajectories. This paradigm can be applied to any dynamic model (we used a nonlinear aerodynamic model for ornithopters) and different flight types. We have demonstrated a proper performance of the algorithm for medium distance flights of up to 250 m, but also for planning short landing trajectories of up to 10 m. The computation time is suitable for online trajectory planning, achieving solutions for short flights in less than 1 s. Moreover, our experimental results have showed that OSPA outperforms alternative probabilistic kinodynamic planners both in cost (total energy) and accuracy (distance to the target). An open-source implementation of OSPA and our benchmarks are available online [10]. Notice that our current implementation is written in Python, so there is still room for improvement with more efficient languages like C.

As future work, we foresee some potential extensions for OSPA that we discuss in the following.

Improving the reference curve: One of the main aspects that affects the performance of the heuristic search in OSPA is the reference curve, as the algorithm relies on having a good approximation of optimal curves to guide the search. We have proposed curves computed empirically, but an open mathematical problem is to calculate the theoretical reference curve that minimizes the energy consumption for a given model. With a better estimation of that optimal energy curve, the parameters k_d and k_w could be reduced to compute pseudo-optimal trajectories more efficiently.

Planning in 3D: In this paper, we have considered 2D trajectories for the ornithopter. However, OSPA is not limited to that. As long as there is available a dynamic model for the 3D motion of the ornithopter, OSPA could be used to plan 3D trajectories. As it is done in 2D, that 3D model would be used by OSPA to generate the tree nodes. The reference curves to prune the tree should also be adapted to 3D using a similar heuristic procedure as the one used in the 2D version. Planning 3D trajectories is particularly useful for settings where the wind effects on lateral displacement cannot be neglected, and it could be tackled by considering curvature-constrained trajectories as in Al-Sabban et al. (2013); Zhang et al. (2014).

Machine learning methods: These days methods for trajectory planning based on machine learning are spreading fast. Even though there are some works that present data from real bird flights Nagy et al. (2010), these data are used to investigate the influence that a given bird has on its fellow flock members. However, high-resolution spatio-temporal data from individuals moving between two known locations are scarce, which makes it difficult to imitate bird trajectories with ornithopters. In this sense, OSPA may be helpful to generate artificial bio-inspired datasets with pseudo-optimal trajectories, which could be used for training alternative machine learning methods.

Experiments with a real ornithopter: Finally, we plan to use OSPA to compute trajectories on board our real ornithopter prototype within the framework of the GRIFFIN project. For that, we will combine the planner with nonlinear controllers for flight sta-

[10] https://github.com/fragnarxx/kinodynamic-planning

bilization and trajectory following.

Acknowledgement. This project has received funding from the European Research Council Advanced Grant GRIFFIN (Action 788247), the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 734922 (CONNECT) and the Spanish Ministry of Economy and Competitiveness (GALGO, MTM2016-76272-R AEI/FEDER,UE).

References

Al-Sabban, W. H., Gonzalez, L. F., & Smith, R. N. (2013). Wind-energy based path planning for unmanned aerial vehicles using markov decision processes. In *IEEE International Conference on Robotics and Automation* (pp. 784–789).

Arabagi, V., Hines, L., & Sitti, M. (2012). Design and manufacturing of a controllable miniature flapping wing robotic platform. *The International Journal of Robotics Research*, *31*(6), 785-800. doi: 10.1177/0278364911434368

Ayancik, F., Zhong, Q., Quinn, D. B., Brandes, A., Bart-Smith, H., & Moored, K. W. (2019). Scaling laws for the propulsive performance of three-dimensional pitching propulsors. *Journal of Fluid Mechanics*, *871*, 1117–1138. doi: 10.1017/jfm.2019.334

Betts, J. T. (2010). *Practical methods for optimal control and estimation using nonlinear programming*. SIAM.

Boussaid, I., Lepagnot, J., & Siarry, P. (2013). A survey on optimization metaheuristics. *Information Sciences*, *237*, 82-117. doi: https://doi.org/10.1016/j.ins.2013.02.041

Brescianini, D., & D'Andrea, R. (2018, June). Computationally efficient trajectory generation for fully actuated multirotor vehicles. *IEEE Transactions on Robotics*, *34*(3), 555-571. doi: 10.1109/TRO.2018.2813373

Chirarattananon, P., Ma, K. Y., & Wood, R. J. (2014). Single-loop control and trajectory following of a flapping-wing microrobot. In *IEEE International Conference on Robotics and Automation (ICRA)* (p. 37-44). doi: 10.1109/ICRA.2014.6906587

Cory, R., & Tedrake, R. (2008). Experiments in fixed-wing UAV perching. In *AIAA Guidance, Navigation and Control Conference and Exhibit* (p. 7256).

Coutinho, W. P., Battarra, M., & Fliege, J. (2018). The unmanned aerial vehicle routing and trajectory optimisation problem, a taxonomic review. *Computers & Industrial Engineering*, *120*, 116–128.

Crowther, W. (2000). Perched landing and take-off for fixed wing UAVs. In *NATO symposium on unmanned vehicles for aerial, ground, and naval military operations* (pp. 9–13).

de Croon, G., de Clercq, K., Ruijsink, R., Remes, B., & de Wagter, C. (2009). Design, aerodynamics, and vision-based control of the delfly. *International Journal of Micro Air Vehicles*, *1*(2), 71-97. doi: 10.1260/175682909789498288

Dietl, J. M., & Garcia, E. (2013). Ornithopter optimal trajectory control. *Aerospace Science and Technology*, *26*(1), 192 - 199. doi: https://doi.org/10.1016/j.ast.2012.04.003

Dokeroglu, T., Sevinc, E., Kucukyilmaz, T., & Cosar, A. (2019). A survey on new generation metaheuristic algorithms. *Computers & Industrial Engineering*, *137*, 106040. doi: https://doi.org/10.1016/j.cie.2019.106040

Fernandez-Feria, R. (2016). Linearized propulsion theory of flapping airfoils revisited. *Physical Review Fluids*, *1*(8), 084502.

Fernandez-Feria, R. (2017). Note on optimum propulsion of heaving and pitching airfoils from linear potential theory. *Journal of Fluid Mechanics*, *826*, 781-796.

Garrick, I. E. (1936). Propulsion of a flapping and oscillating airfoil. *Technical Report TR 567, NACA*.

Grauer, J., & Hubbard, J. (2010). Modeling of ornithopter flight dynamics for state estimation and control. In *Proceedings of the American Control Conference* (p. 524-529). doi: 10.1109/ACC.2010.5530874

Hauser, K., & Zhou, Y. (2016, December). Asymptotically optimal planning by feasible kinodynamic planning in a state-cost space. *IEEE Transactions on Robotics*, *32*(6), 1431–1443. doi: 10.1109/TRO.2016.2602363

Hoff, J., Syed, J., Ramezani, A., & Hutchinson, S. (2019). Trajectory planning for a bat-like flapping wing robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (p. 6800-6805). doi: 10.1109/IROS40897.2019.8968450

Houska, B., Ferreau, H., & Diehl, M. (2011). ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization. *Optimal Control Applications and Methods*, *32*(3), 298–312. doi: https://doi.org/10.1002/oca.939

Hussain, K., Salleh, M. M., Cheng, S., & Shi, Y. (2019). Metaheuristic research: a comprehensive survey. *Artificial Intelligence Review*, *52*(4), 2191-2233. doi: https://doi.org/10.1007/s10462-017-9605-z

Karaman, S., & Frazzoli, E. (2010). Optimal kinodynamic motion planning using incremental sampling-based methods. In *IEEE Conference on Decision and Control (CDC)* (p. 7681-7687). doi: 10.1109/CDC.2010.5717430

Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, *30*(7), 846-894. doi: 10.1177/0278364911406761

Li, Y., Littlefield, Z., & Bekris, K. E. (2016). Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research*, *35*(5), 528-564. doi: 10.1177/0278364915614386

Maldonado, F. J., Acosta, J. A., Tormo-Barbero, J., Grau, P., Guzman, M. M., & Ollero, A. (2020). Adaptive nonlinear control for perching of a bioinspired ornithopter. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (p. 1385-1390). doi: 10.1109/IROS45743.2020.9341793

Martín-Alcántara, A., Grau, P., Fernandez-Feria, R., & Ollero, A. (2019). A simple model for gliding and low-amplitude flapping flight of a bio-inspired UAV. In *International Conference on Unmanned Aircraft Systems (ICUAS)* (p. 729-737). doi: 10.1109/ICUAS.2019.8798233

Mellinger, D., & Kumar, V. (2011). Minimum snap trajectory generation and control for quadrotors. In *IEEE International Conference on Robotics and Automation (ICRA)* (p. 2520-2525). doi: 10.1109/ICRA.2011.5980409

Menezes, A. A., & Kabamba, P. T. (2016). Efficient and resilient micro air vehicle flapping wing gait evolution for hover and trajectory control. *Engineering Applications of Artificial Intelligence*, *54*, 1 - 16. doi: https://doi.org/10.1016/j.engappai.2016.05.001

Mueller, M. W., Hehn, M., & D'Andrea, R. (2015, Dec). A computationally efficient motion primitive for quadrocopter trajectory generation. *IEEE Transactions on Robotics*, *31*(6), 1294-1310. doi: 10.1109/TRO.2015.2479878

Nagy, M., Akos, Z., Biro, D., & Vicsek, T. (2010). Hierarchical group dynamics in pigeon flocks. *Nature*, *464*, 890-893. doi: 10.1038/nature08891

Nguyen, Q.-V., & Chan, W. L. (2018). Development and flight performance of a biologically-inspired tailless flapping-wing micro air vehicle with wing stroke plane modulation. *Bioinspiration & Biomimetics*, *14*(1), 016015. doi: 10.1088/1748-3190/aaefa0

Nguyen, T. A., Phan, H. V., Au, T. K. L., & Park, H. C. (2016). Experimental study on thrust and power of flapping-wing system based on rack-pinion mechanism. *Bioinspiration & Biomimetics*, *11*(4), 046001. doi: 10.1088/1748-3190/11/4/046001

Oleynikova, H., Burri, M., Taylor, Z., Nieto, J., Siegwart, R., & Galceran, E. (2016). Continuous-time trajectory optimization for online UAV replanning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (p. 5332-5339). doi: 10.1109/IROS.2016.7759784

Paranjape, A. A., Chung, S., & Kim, J. (2013). Novel dihedral-based control of flapping-wing aircraft with application to perching. *IEEE Transactions on Robotics*, *29*(5), 1071-1084. doi: 10.1109/TRO.2013.2268947

Paranjape, A. A., Dorothy, M. R., Chung, S.-J., & Lee, K.-D. (2012). A flight mechanics-centric review of bird-scale flapping flight. *International Journal of Aeronautical and Space Sciences*, *13*(3), 267–281.

Paranjape, A. A., Meier, K. C., Shi, X., Chung, S.-J., & Hutchinson, S. (2015). Motion primitives and 3D path planning for fast flight through a forest. *The International Journal of Robotics Research*, *34*(3), 357-377. doi: 10.1177/0278364914558017

Posa, M., Cantu, C., & Tedrake, R. (2014). A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, *33*(1), 69–81.

Qin, Y., Cheng, B., & Deng, X. (2014). Trajectory optimization of flapping wings modeled as a three degree-of-freedoms oscillation system. In

IEEE/RSJ International Conference on Intelligent Robots and Systems (p. 3193-3200). doi: 10.1109/IROS.2014.6943005

Rose, C. J., Mahmoudieh, P., & Fearing, R. S. (2016). Modeling and control of an ornithopter for diving. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (p. 957-964). doi: 10.1109/IROS.2016.7759165

Sihite, E., & Ramezani, A. (2020). Enforcing nonholonomic constraints in aerobat, a roosting flapping wing model. In *IEEE Conference on Decision and Control (CDC)* (p. 5321-5327). doi: 10.1109/CDC42340.2020.9304158

Stoneman, S., & Lampariello, R. (2014). Embedding nonlinear optimization in RRT* for optimal kinodynamic planning. In *IEEE Conference on Decision and Control* (p. 3737-3744). doi: 10.1109/CDC.2014.7039971

Theodorsen, T. (1935). General theory of aerodynamic instability and the mechanism of flutter. *Technical Report TR 496, NACA*.

Thomas, A. L. (1993). On the aerodynamics of birds' tails. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, *340*(1294), 361-380.

Wang, J., Zahr, M. J., & Persson, P.-O. (2017). Energetically optimal flapping flight via a fully discrete adjoint method with explicit treatment of flapping frequency. In *AIAA Computational Fluid Dynamics Conference*. doi: 10.2514/6.2017-4412

Webb, D. J., & van den Berg, J. (2013). Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics. In *IEEE International Conference on Robotics and Automation (ICRA)* (p. 5054-5061). doi: 10.1109/ICRA.2013.6631299

Zanelli, A., Domahidi, A., Jerez, J., & Morari, M. (2017). FORCES NLP: an efficient implementation of interior-point methods for multistage nonlinear nonconvex programs. *International Journal of Control*, 13-29. doi: 10.1080/00207179.2017.1316017

Zhang, X., Chen, J., Xin, B., & Peng, Z. (2014). A memetic algorithm for path planning of curvature-constrained UAVs performing surveillance of multiple ground targets. *Chinese Journal of Aeronautics*, *27*(3), 622–633.