Routing a fleet of unmanned aerial vehicles: a trajectory optimisation-based framework

Walton P. Coutinho*[a], Jörg Fliege[b], Maria Battarra[c], Anand Subramanian[d]

[a] *Department of Technology, Federal University of Pernambuco, Caruaru, 55016-400, Brazil*
walton.coutinho@ufpe.br
[b] *Mathematical Sciences, University of Southampton, Southampton, SO17 1BJ, United Kingdom*
j.fliege@soton.ac.uk
[c] *School of Management, University of Bath, Claverton Down, Bath, BA2 7AY, United Kingdom*
m.battarra@bath.ac.uk
[d] *Departamento de Sistemas de Computação, Centro de Informática, Universidade Federal da Paraíba, João Pessoa, 58058-600, Brazil*
anand@ci.ufpb.br

Abstract

We consider an aerial survey operation in which a fleet of unmanned aerial vehicles (UAVs) is required to visit several locations and then land in one of the available landing sites while optimising some performance criteria, subject to operational constraints and flight dynamics. We aim to minimise the maximum flight time of the UAVs. To efficiently solve this problem, we propose an algorithmic framework consisting of: (i) a nonlinear programming formulation of trajectory optimisation that accurately reflects the underlying flight dynamics and operational constraints; (ii) two sequential trajectory optimisation heuristics, designed to cope with the challenging task of finding feasible flight trajectories for a given route; and (iii) a routing metaheuristic combining iterated local search and a set-partitioning-based integer programming formulation. The proposed framework is tested on randomly generated instances with up to 50 waypoints, showing its efficacy.

Keywords: Unmanned gliders, routing, trajectory optimisation

1. Introduction

The traditional way of performing aerial survey operations involves the use of manned aircraft. While this solution has been widely used, it presents several drawbacks such as high operational costs, the need for nearby infrastructure (e.g., helipads and runways), relatively high response times and the life risk imposed on the aircraft crew. An alternative to this approach consists of using Unmanned Aerial Vehicles (UAVs), a.k.a. drones (Xia et al., 2017; Öztürk & Köksalan, 2023). UAVs are aircraft that do not need a human pilot on board. These vehicles can be controlled either by autonomous embedded computers or by a remote pilot. Several applications of this type can be found in the literature, such as for forest fire detection (Yuan et al., 2015), target observation (Rysdyk, 2006), traffic monitoring and management (Kanistras et al., 2015), military operations (Xia et al., 2017), three-dimensional mapping (Nex & Remondino, 2013) and disaster assessment (Nedjati et al., 2016; Aretoulaki et al., 2023).

Gliders are UAVs without an onboard propulsion system (e.g., an electrical or combustion engine). In the past few years, UAVs have become very popular for logistics and surveillance applications. The main advantage of gliders over other powered UAVs is their unit cost. As examples, the SULSA UAVs can be easily 3D printed (Keane et al., 2017), while the so-called MAVIS gliders provide a low-cost platform that can be launched by atmospheric balloons (Crispin, 2016).

Providing rapid response to unpredictable and large-scale disasters is a key challenge for search and rescue organisations around the world. In the aftermath of such events, collecting as much information as possible about its effects is a crucial activity that must be performed in a timely and cost-efficient way. Aerial survey operations play an important role when rapid and accurate information of affected

*Corresponding author.

areas is necessary (Mersheeva, 2015). High-resolution imaging of entire affected areas can provide search and rescue teams with useful reports about the location of victims, damaged buildings and potential environmental hazards, among others. Moreover, better response and evacuation plans can be designed with the support of aerial imaging (Aretoulaki et al., 2023).

In this work, we consider the problem in which a fleet of aerial gliders launched from an atmospheric balloon or some other air platform is required to visit several *waypoints*, representing points of interest. The gliders must land at one of the available landing sites while optimising some performance criteria, e.g., the mission time, subject to operational constraints and flight dynamics. Minimising such mission time is obviously one of the relevant aspects of fast disaster response. We refer to this problem as the Glider Routing and Trajectory Optimisation Problem (GRTOP). Most of the literature on UAV routing problems overlooks the influence of flight dynamics when formulating aerial route designs (Agatz et al., 2018; Khoufi et al., 2019; Morandi et al., 2023). In the case of gliders, such considerations are not only relevant but necessary to ensure the feasibility of routes.

Integrating flight dynamics into the design of routes is a very challenging task that, in the O.R. context, can be seen as a combination of the Vehicle Routing Problem (VRP) and the Trajectory Optimisation Problem (TOP) Coutinho et al. (2018). In the current paper, we integrate for the first time non-linear flight dynamics and routing decisions for a fleet of gliders in a computationally efficient way. By this, we provide important contributions to the UAV routing and Trajectory Optimisation (TO) literature and provide substantial methodological innovation compared to previous work, see, e. g. Coutinho et al. (2019).

Our main contributions can be summarised as follows:

- We propose a novel multi-phase Mixed-Integer Non-linear Programming (MINLP) formulation for the GRTOP that allows for the use of sub-models of varying fidelity for TO. For example, along the arcs of a given route, we allow for different flight modes, flight dynamics, wind conditions, discretisation methods and discretisation step sizes;
- We provide theoretical bounds on the discretisation errors for the linearised reformulation of the gliders' Equations of Motion (EOMs) and demonstrate how to reformulate the proposed GRTOP model to incorporate the error-bounding constraints. Our computational experiments show that providing a modelling framework that incorporates such errors leads to more accurate trajectories;
- We develop two heuristics based on the so-called Sequential Trajectory Optimisation (STO) approach, designed to find feasible (flyable) trajectories for a given route with low computational effort. The first heuristic is based on nonconvex trajectory optimisation subproblems, while the second one is based on an iterative flight time minimisation procedure that solves Second-Order Cone Programmings (SOCPs) subproblems;
- By integrating the proposed STO heuristics with a state-of-the-art routing algorithm, we develop a new matheuristic framework for the GRTOP in which we decouple the continuous dynamics of flight from the combinatorial waypoint routing problem. We highlight that such integration is non-trivial since one has to find a good compromise between local search and trajectory computations to develop a scalable algorithm. This computational framework allows us to solve large-sized problem instances.

In total, we present an optimisation framework that takes as input a set of waypoints, environmental conditions and flight dynamics of the UAVs under question and computes flyable trajectories and control commands that can be promptly embedded into the UAVs' microcontrollers. Our framework can be easily adapted for problems involving other autonomous vehicles such as powered UAVs and unmanned marine vehicles.

Some further contributions are as follows. (i) We employ an objective function based on the concept of makespan that minimises the mission duration. This formulation better suits the needs of disaster response organisations when a large number of waypoints must be surveyed. In such cases, it is a strategic goal to collect information about all waypoints as quickly as possible before designing an adequate intervention plan. (ii) We compare two different strategies for calculating steady-flight conditions for level and descent flight modes. Such conditions are necessary to model different flight modes and can be used in the linearisation of non-linear flight dynamics. (iii) We address the lack of data by providing a problem generator and a rich set of large-sized benchmark instances. (iv) We test our GRTOP solution framework on existing instances as well as on novel randomly generated instances with up to 50 waypoints. We show that our framework is capable of finding feasible solutions within 200 seconds of computing time on average.

The remainder of this paper is organised as follows. In Section 2, we present different equilibrium conditions of gliding flight. Section 3 presents a multi-phase MINLP formulation for the GRTOP. A linearisation of the gliders' EOM and theoretical contributions are presented in Section 4. In Sections 5 and 6, a matheuristic approach is proposed for the GRTOP. Section 7 describes an extensive number of computational experiments that allow us to assess the performance of the proposed STO algorithms, showing the efficacy of our approach. Finally, in Section 8, we provide conclusions and recommendations for future research.

2. Flight dynamics

This section contains the technical background on glider flight dynamics and provides innovative means to calculate steady-state flight conditions for two different flight modes.

2.1. Preliminaries

We define the *state* of a glider at time $\tau \in \mathbb{R}_{\geq 0}$ as $\mathbf{y}(\tau) = (x(\tau), y(\tau), h(\tau), v(\tau), \gamma(\tau), \varphi(\tau))^\top$, where $x(\tau)$, $y(\tau) \in \mathbb{R}$ and $h(t) \in \mathbb{R}_{\geq 0}$ denote the position and height of the glider, while $v(\tau) \in \mathbb{R}_{\geq 0}$ is it's *airspeed* (flight velocity). The airspeed can be defined as the glider's rate of movement relative to the wind velocity. Variables $\gamma(\tau)$ and $\varphi(\tau) \in \mathbb{R}$, denote the flight path and heading angles, respectively. Let us define the *control* variables (or input) as $\mathbf{u}(\tau) = (Cl(\tau), \mu(\tau))^\top$. Here, $Cl(\tau) \in \mathbb{R}$ represents the lift coefficient and variable $\mu(\tau) \in \mathbb{R}$ the bank angle. The lift coefficient accounts for the amount of lift generated by the wings of an aircraft. The angles $\gamma(\tau)$, $\varphi(\tau)$ and $\mu(\tau)$, depicted in Figure 1, are defined over an *aerodynamic* frame (a.k.a., *relative* frame). In Figure 1, the North-East-Down frame, represented by vectors x_o, y_o and $-h_o$, is rotated to obtain the aerodynamic frame denoted by x_A, y_A and $-h_A$ in the sequence of rotation of the angles $\varphi \to \gamma \to \mu$. This is a common representation used in the aviation literature (Fisch, 2011). For simplicity, we will conveniently omit τ when referring to state and control variables. For a more detailed understanding of aircraft flight dynamics, we refer the interested reader to the books by Blanchard (1967), Russell (1996), Stengel (2004) and Fisch (2011).

In this paper, we employ the EOMs presented by Zhao (2004) to model the flight of unmanned gliders. A compact representation of the system dynamics can be written as the system of Ordinary Differential Equations (ODEs) in Equation (1), where $f(\mathbf{y}(\tau), \mathbf{u}(\tau), \tau) \in \mathbb{R}^6$ corresponds to the function describing the evolution of the system dynamics over time. Here, we use the dot notation "$\dot{}$" to represent time derivatives.

$$\dot{\mathbf{y}} = f(\mathbf{y}(\tau), \mathbf{u}(\tau), \tau) \tag{1}$$

We assume that state and control variables are limited by lower and upper bounds. Here we denote $\mathbf{y}_{lb} = (x_{lb}, y_{lb}, h_{lb}, v_{lb}, \gamma_{lb}, \varphi_{lb})^\top$ and $\mathbf{u}_{lb} = (Cl_{lb}, \mu_{lb})^\top$ as the lower bounds on state and control variables,

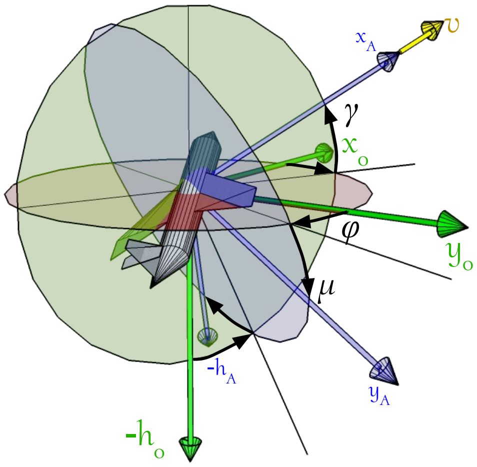


Figure 1: Coordinate frames used to define the glider's flight dynamics.

respectively. Similarly, $\mathbf{y}_{ub} = (x_{ub}, y_{ub}, h_{ub}, \upsilon_{ub}, \gamma_{ub}, \varphi_{ub})^\top$ and $\mathbf{u}_{ub} = (Cl_{ub}, \mu_{ub})^\top$ denote the upper bounds on states and controls. For the sake of clarity, we have listed all design parameters and constant values in this paper's online supplement.

2.2. *Equilibrium flight modes*

Two types of stability can be defined for an aircraft. A body is said to be in *static stability* (or in a *static steady-state*) if its state is to some extent resistant to disturbances (being stationary or at rest). *Dynamic stability* requires an investigation using the full dynamic equations of an aircraft. In a steady-state flight, the aerodynamic basic forces are balanced. A powered UAV, for example, can achieve a steady flight when lift equals weight and thrust equals drag. Similarly, a glider is in steady flight when its airspeed and *angle of attack* (Stengel, 2004, p. 53) are such that the lift force equals its weight.

Let us denote by $\mathbf{y}_{eq} = (x_{eq}, y_{eq}, z_{eq}, \upsilon_{eq}, \gamma_{eq}, \varphi_{eq})^\top$ and $\mathbf{u}_{eq} = (Cl_{eq}, \mu_{eq})^\top$ some steady-states and controls, respectively, of the dynamics defined by Equation (1). Then, in continuous time, the derivatives of the state variable with respect to time are zero ($\mathbf{0} \in \mathbb{R}^6$), that is:

$$\dot{\mathbf{y}} = f(\mathbf{y}_{eq}, \mathbf{u}_{eq}, \tau) = \mathbf{0}. \tag{2}$$

In this paper, we are concerned with finding equilibrium flight conditions for two distinct practical situations, namely, *steady-level* flight, in which the origin and destination points are nearly at the same altitude, and *steady-descent* flight, in which the origin is at a higher altitude and the glider must descend to reach the desired destination.

2.2.1. *Steady-level flight conditions*

In previous work, Coutinho et al. (2016, 2019) applied a set of analytical steady-state conditions for a gliding level-flight as described, e.g., in Russell (1996). Such analytical steady-states were computed under simplifying assumptions about the glider's flight dynamics. In this paper, we instead use a numerical approach for computing more accurate and realistic steady-states without additional assumptions. Our formulation extends the one presented by Stengel (2004) by the addition of box constraints (4) and constraints (5) on state and control variables. These additional constraints are important to obtain realistic, that is, flyable trajectories. In full, we consider the following optimisation problem for an arbitrary fixed time τ to compute a steady-state solution:

$$\min_{\mathbf{y},\mathbf{u}} \quad ||f(\mathbf{y}, \mathbf{u})||_2 \tag{3}$$

$$s.t. \quad \mathbf{y}_{lb} \leq \mathbf{y} \leq \mathbf{y}_{ub} \tag{4}$$

$$\mathbf{u}_{lb} \leq \mathbf{u} \leq \mathbf{u}_{ub}. \tag{5}$$

In this problem, the objective function (3) minimises the Euclidean norm of the right-hand side of Equation (1) for an arbitrary fixed τ. By minimising such norm, we expect to find a solution that fulfils the condition in Equation (2), if such a solution exists. Otherwise, only an approximation is returned. Constraints (4) and (5) ensure that the optimal steady-flight conditions lie within the bounds of state and control variables. As we will see below, this problem is equivalent to a quadratic programming problem which can be reliably solved by available optimisation software. Let $\mathbf{y}^*$ and $\mathbf{u}^*$ be the optimal solution of the optimisation problem defined by Equations (3–5) for an arbitrary fixed time τ. With these we will approximate the steady-states $\mathbf{y}_{eq}$, $\mathbf{u}_{eq}$ of Equation (2).

In Table 1, we compare the steady-states found analytically, denoted by s_1, with the ones found by using the proposed formulation, denoted by s_2. We fixed the reference altitude to 500 metres to match the parameter for the wind strength as defined in this paper's online supplement. Table 1 shows that the proposed numerical approach provides better results than the analytical solution, i.e., s_2 is closer to the condition defined in Equation (2) than s_1. Hence, hereafter we will use s_2 as level flight steady-state conditions.

Table 1: Comparison of steady-level flight states.

	h_{eq}	v_{eq}	γ_{eq}	φ_{eq}	Cl_{eq}	μ_{eq}	$\|\|f(\mathbf{y},\mathbf{u})\|\|_2$
s_1	500.00	9.45	-0.04	0.0	0.74	0.0	15.6705
s_2	500.00	12.48	-0.02	-1.57	0.37	0.0	0.4133

2.2.2. Steady-descent flight

Level-flight steady-states are easy and fast to compute, but they may not be accurate for descent or climb manoeuvres. The forces acting on a UAV in steady-descent gliding flight are *lift* (L), drag (D) and weight (defined as the mass of the glider m_g times gravity g_e). In a descent flight, these forces are in equilibrium and can be depicted as in Figure 2, where a UAV is assumed to move from (x_1, y_1, h_1) to (x_2, y_2, h_2). Here, the equilibrium bank and heading angles are considered to be close to zero, i.e., $\mu_{eq} \approx \varphi_{eq} \approx 0$. If we approximate the equilibrium pitch angle γ_{eq} by the angle between points (x_1, y_1, h_1) and (x_2, y_2, h_2) compared to a horizontal plane, then γ_{eq} can be written as in Equation (6), where R is the flight range and $\Delta h = h_2 - h_1$:

$$\gamma_{eq} \approx -\tan^{-1}(\Delta h/R). \tag{6}$$

From the triangle of forces in Figure 2, one can write an expression relating γ_{eq} to the lift and drag forces as in Equation (7). The rightmost expression in Equation (7) follows from the definition of the lift and drag coefficients, respectively, Cl and C_D (Russell, 1996, p. 42):

$$\cos\gamma_{eq} = \frac{L}{\sqrt{L^2 + D^2}} = \frac{Cl}{\sqrt{Cl^2 + C_D^2}}. \tag{7}$$

Two important parameters that are fixed when designing the glider are the coefficient of drag at zero-lift, C_{D0}, and the aerodynamic coefficient of the glider, k_A. The values of these that we will use are provided in this paper's online supplement. Let us then define the auxiliary variables $b = 1 + \sqrt{2C_{D0}k_A} - \cos\gamma_{eq}^{-1}$ and $\Delta = b^2 - 4k_AC_{D0}$. Now, the equilibrium lift coefficient can be computed as

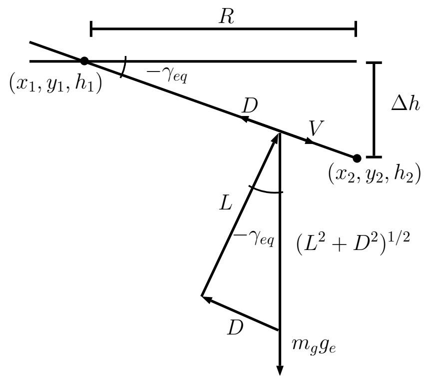


Figure 2: Steady descent flight conditions. The glider is assumed to be in a descent flight with airspeed V going from point (x_1, y_1, h_1) to (x_2, y_2, h_2).

in Equation (8):

$$Cl_{eq} = \min\left\{\frac{-b \pm \Re(\sqrt{\Delta})}{2k_A}\right\}, \tag{8}$$

where we denote by $\Re(.)$ the real part of a complex number. Solving Equation (7) for Cl leads to a quadratic equation with two possible solutions. The two roots of this equation define a minimum and a maximum *angle of attack*. By choosing the smallest root, one indirectly chooses the smallest angle of attack and therefore the minimum amount of generated lift, thus allowing the glider to perform a steady-descent flight.

Finally, from the lift equation, one can derive an expression for the equilibrium airspeed as in Equation (9), involving the additional glider parameters m_g (the mass of the glider) and S (wing area), as well as the density of the air ρ:

$$v_{eq} = \begin{cases} \sqrt{\frac{2m_g g_e \cos\gamma_{eq}}{\rho S Cl_{eq}}}, & \text{if } Cl_{eq} > 0, \\ v_{lb}, & \text{otherwise.} \end{cases} \tag{9}$$

Again, the values used here are provided in this paper's online supplement.

By approximating the steady-descent pitch (γ_{eq}), lift coefficient (Cl_{eq}) and airspeed (v_{eq}) using Equations (6), (8) and (9), respectively, one might find equilibrium values that are out of the bounds given for the state and control variables. In this case, we simply project the respective equilibrium values to the nearest bound.

3. Problem definition

We will consider the following problem. A fleet G of gliders is available at the launch site $0 \in \mathbb{R}^3$. We consider a finite graph with vertex set $V' = \{0\} \cup V \cup L$ and a set A of arcs connecting these vertices. The set V represents all waypoints that need to be visited by the fleet of gliders, while L is the set of all possible landing sites. We assume $V \cap L = \emptyset$ and $0 \notin V \cup L$. In general, the graph formed by the sets V' and A is not complete since we do not allow for arcs connecting the launch point directly to any landing site, nor do we consider arcs going from waypoints to the launch point. Moreover, problem-specific characteristics might eliminate further arcs from consideration. The objective is to find optimal routes and trajectories in such a way that: (i) all waypoints in V are visited at least once; (ii) each route finishes at some landing site in L; (iii) all routes depart from the launch site, and; (iv) the maximum flight duration among all gliders is minimised.

Each waypoint $i \in V$ is represented by $(\bar{x}_i, \bar{y}_i, \bar{r}_i, \underline{h}_i, \overline{h}_i)$, where $(\bar{x}_i, \bar{y}_i)$ is the position of the object

i in the xy plane and $\bar{r}_i > 0$ is the radius of the base of an inverted truncated cone with the waypoint at its centre. Gliders will visit waypoints in order to photograph them. To this end, we are given $\underline{h}_i$ and $\overline{h}_i$, the minimum and maximum allowed photographing heights of waypoint i. Assuming w.l.o.g. that the opening angle of the cameras is 45° and that all waypoints lie in the same xy plane, a glider at position (x, y) and flying at altitude h is said to *visit* waypoint i if the glider passes through the inverted truncated cone covering i and respecting the minimum and maximum photographing altitudes, i.e. if $(x - \bar{x}_i)^2 + (y - \bar{y}_i)^2 \leq (h + \bar{r}_i)^2$ and $\underline{h}_i \leq h \leq \overline{h}_i$ holds. A landing site $i \in L$ is determined by a half-sphere of radius $\tilde{r}_i$ centred at $(\tilde{x}_i, \tilde{y}_i)$. Assuming that landing sites lie in the same xy plane as the waypoints, a glider will be considered landed at site i if it touches or enters the half-sphere containing i, i.e. if $(x - \tilde{x}_i)^2 + (y - \tilde{y}_i)^2 + h^2 \leq \tilde{r}_i^2$. Figure 3 depicts a feasible solution for an instance with two waypoints and one landing site.

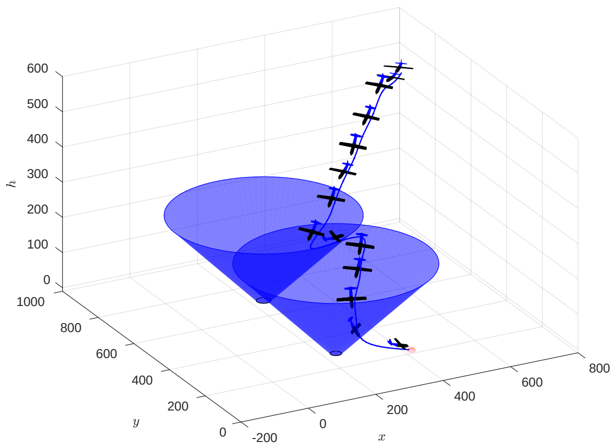


Figure 3: A feasible solution for the GRTOP showing the geometry of waypoints and landing sites. If the gliders' cameras are the same, one can transfer their field of view to the waypoints forming truncated inverted cones.

The problem defined here shares some similarities with the one proposed by Coutinho et al. (2019), but there are also important differences. In the present work the TO is modelled as a multi-phase problem, in which changes to the system dynamics during the flight are modelled one arc at a time, leading to a more accurate representation of the overall dynamics. The exact approach of Coutinho et al. (2019), based on a single-phase formulation, requires a fixed and rather coarse discretisation step size and flight times for all gliders' trajectories. Given that gliders might have very different flight times (i.e., some fly for minutes, others for hours), such a global time discretisation will often be too coarse for long flights, and possibly lead to trajectories that are not sufficiently accurate. In the following, we alleviate this problem by providing a mathematical formulation for TO that allows for adaptive discretisation schemes.

3.1. A Multi-phase mixed-Integer trajectory optimisation formulation

We associate each arc $(i, j) \in A$ as a phase of a glider's trajectory. A *phase* is a section of the trajectory in which the flight dynamics and parameters (e.g., flight mode, target definition and flight environment) remain unchanged. In this paper, we use the words phase and arc interchangeably. Let us denote by $\mathbf{y}_{ijg}(\tau_{ijg}) : [0, \infty) \to \mathbb{R}^6$ and $\mathbf{u}_{ijg}(\tau_{ijg}) : [0, \infty) \to \mathbb{R}^2$ the state and control variables, respectively, of glider $g \in G$ flying along arc $(i, j) \in A$, and denote by $\tau_{ijg} \in \mathbb{R}$ the time variable. Let variables τ^o_{ijg} and τ^f_{ijg} represent the initial and final flight times of the glider g flying along arc (i, j) such that $\tau_{ijg} \in [\tau^o_{ijg}, \tau^f_{ijg}]$. We recall that the unknown states and controls are interpreted as the evolution of the dynamical system (Equation 1), where τ is the independent variable.

In what follows, we express the EOMs of the glider g flying on arc (i, j) by

$$\dot{\mathbf{y}}_{ijg} = f_{ijg}(\mathbf{y}_{ijg}(\tau_{ijg}), \mathbf{u}_{ijg}(\tau_{ijg}), \tau_{ijg}). \tag{10}$$

Both state and control variables are limited by lower and upper bounds, as explained in Section 2.1. Initial conditions $\mathbf{y}^o$ and $\mathbf{u}^o$ must be provided at time 0, i.e., $\mathbf{y}_{ijg}(0) = \mathbf{y}^o$ and $\mathbf{u}_{ijg}(0) = \mathbf{u}^o$, for all $(i,j) \in A$ and for all $g \in G$.

Figure 4 illustrates our conceptual model with a small example and its respective feasible solution. Let the launching point be 0, $V = \{1,2\}$, $L = \{3\}$, and $G = \{1,2\}$. For each arc in the set $A = \{(0,1),(0,2),(1,2),(2,1),(1,3),(2,3)\}$ there are associated times τ_{ijg}, states $\mathbf{y}_{ijg}(\tau_{ijg})$ and controls $\mathbf{u}_{ijg}(\tau_{ijg})$, variable time limits τ^o_{ijg} and τ^f_{ijg}, and a dynamical system as in Equation (10), where $(i,j) \in A$ and $g \in G$. In Figure 4, the flight duration of gliders 1 and 2 can be computed as $\Delta t_1 = (\tau^f_{011} - \tau^o_{011}) + (\tau^f_{131} - \tau^o_{131}) = \tau^f_{131}$ and $\Delta t_2 = (\tau^f_{022} - \tau^o_{022}) + (\tau^f_{232} - \tau^o_{232}) = \tau^f_{232}$. Note that the continuity of the dynamics of each glider's route is guaranteed by setting the values of states and controls at the initial time of an arc equal to the states and controls at the final time of its preceding arc, assuming w.l.o.g. that $\tau^o_{0jg} = 0, \forall j \in V, g \in G$.

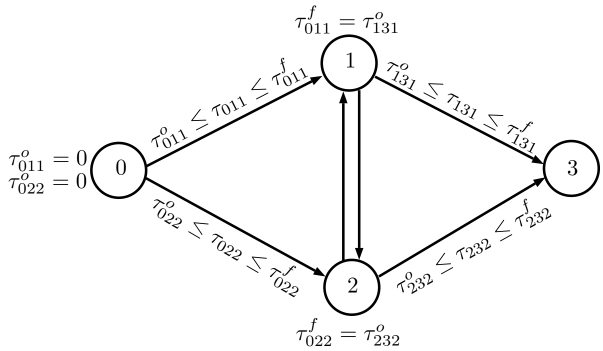


Figure 4: A graph representing routes $(0,1,3)$ and $(0,2,3)$ and their respective time, state and control continuity constraints.

Now let us define variables $a_{ijg} \in \{0,1\}$ such that that $a_{ijg} = 1$ if glider g traverses arc $(i,j) \in A$ and taking value 0 otherwise. For simplicity, we define the set A' as the set of arcs not leaving from the launching point, i.e., $A' = A \setminus \{(0,j), j \in V\}$. Thus, the objective

$$\min \max_{g \in G} \left\{ \sum_{(i,j) \in A} (\tau^f_{ijg} - \tau^o_{ijg}) a_{ijg} \right\}. \tag{11}$$

minimises the total flight duration of the longest route (where the length of a route is measured in flight time). We highlight that this objective formulation is nonlinear since the initial τ^o_{ijg} and final τ^f_{ijg} flight times depend nonlinearly on other decision variables, to be introduced below.

Constraints (12–14) account for the assignment of routes to gliders. Constraint (12) ensures that every launched glider lands in one of the predetermined landing sites. Constraints (13) and (14) make sure that every waypoint is visited at least once and that the continuity of routes is preserved:

$$\sum_{i \in V} a_{0ig} = \sum_{i \in V} a_{ilg} \leq 1, \forall l \in L, \forall g \in G \tag{12}$$

$$\sum_{g \in G} \sum_{i \in V} a_{ijg} = 1, \forall j \in V, j \neq i \tag{13}$$

$$\sum_{i \in V} a_{ijg} - \sum_{i \in V} a_{jig} = 0, \forall j \in V, j \neq i, \forall g \in G. \tag{14}$$

Constraints (15–17) below ensure that the gliders fly through the respective covering regions of the waypoints at the end times of each phase, and finally arrive at a landing site. A phase at an arc (i,j) is deemed to start and end at its respective initial and final flight times τ^o_{ijg} and τ^f_{ijg}. Constraints (15) and (16) ensure that a glider g flies within the boundaries of waypoint i at time τ^o_{ijg}, if arc (i,j) is used, while Constraints (17) state that a glider g must be within the boundaries of a landing site at the end of its mission.

$$a_{ijg}((x_{ijg}(\tau^o_{ijg}) - \bar{x}_i)^2 + (y_{ijg}(\tau^o_{ijg}) - \bar{y}_i)^2) \leq (h_{ijg}(\tau^o_{ijg}) + \bar{r}_i)^2, \forall (i,j) \in A', \forall g \in G \tag{15}$$

$$a_{ijg}\underline{h}_i \leq a_{ijg}h_{ijg}(\tau^o_{ijg}) \leq \overline{h}_i, \forall (i,j) \in A', \forall g \in G \tag{16}$$

$$a_{ijg}((x_{ijg}(\tau^f_{ijg}) - \tilde{x}_i)^2 + (y_{ijg}(\tau^f_{ijg}) - \tilde{y}_i)^2 + h^2_{ijg}(\tau^f_{ijg})) \leq \tilde{r}^2_j, \forall i \in V, \forall j \in L, \forall g \in G. \tag{17}$$

Taking photographs at an unfavourable angular orientation (a.k.a. flight attitude) must be avoided. For this, we consider given parameters $\hat{\gamma}$ for the maximum pitch and $\hat{\mu}$ for the maximum roll angle, and add constraints (18) and (19) that ensure that glider g is nearly in level flight at the moment it takes a photograph of waypoint i:

$$-\hat{\gamma} \leq a_{ijg}\gamma_{ijg}(\tau^o_{ijg}) \leq \hat{\gamma}, \forall (i,j) \in A', \forall g \in G \tag{18}$$

$$-\hat{\mu} \leq a_{ijg}\mu_{ijg}(\tau^o_{ijg}) \leq \hat{\mu}, \forall (i,j) \in A', \forall g \in G. \tag{19}$$

Next, we provide the constraints that describe the flight dynamics of the gliders. Constraints (20) enforce the flight dynamics of glider g to be applied if this glider flies through arc (i,j). Constraints (21–23) ensure the continuity of state, control and time variables is maintained if arc (i,j) precedes arc (j,k) in a solution. Constraints (24) preserve the time variable within its bounds on arc (i,j).

$$\dot{\mathbf{y}}_{ijg} = f_{ijg}(\mathbf{y}_{ijg}(\tau_{ijg}), \mathbf{u}_{ijg}(\tau_{ijg}), \tau_{ijg})a_{ijg}, \forall (i,j) \in A, \forall g \in G \tag{20}$$

$$\mathbf{y}_{jkg}(\tau^o_{jkg})a_{ijg} = \mathbf{y}_{ijg}(\tau^f_{ijg})a_{ijg}a_{jkg}, \forall (i,j),(j,k) \in A, \forall g \in G \tag{21}$$

$$\mathbf{u}_{jkg}(\tau^o_{jkg})a_{ijg} = \mathbf{u}_{ijg}(\tau^f_{ijg})a_{ijg}a_{jkg}, \forall (i,j),(j,k) \in A, \forall g \in G \tag{22}$$

$$\tau^o_{jkg}a_{ijg} = \tau^f_{ijg}a_{ijg}a_{jkg}, \forall (i,j),(j,k) \in A, \forall g \in G \tag{23}$$

$$\tau^o_{ijg} \leq \tau_{ijg} \leq \tau^f_{ijg}, \forall (i,j) \in A, \forall g \in G \tag{24}$$

Finally, Constraints (25–28) define the domain of the optimisation variables:

$$a_{ijg} \in \{0,1\}, \forall (i,j) \in A, \forall g \in G \tag{25}$$

$$\mathbf{y}_{ijg}(\tau_{ijg}) \in \mathbb{R}^6, \forall (i,j) \in A, \forall g \in G \tag{26}$$

$$\mathbf{u}_{ijg}(\tau_{ijg}) \in \mathbb{R}^2, \forall (i,j) \in A, \forall g \in G \tag{27}$$

$$\tau^o_{ijg}, \tau^f_{ijg} \in \mathbb{R}, \forall (i,j) \in A, \forall g \in G. \tag{28}$$

The formulation defined by Expressions (11–28) is a non-convex MINLP TO problem. In particular, due to constraints (20), solving this formulation directly using off-the-shelf optimisation software is very challenging, if not impossible, even for small instances. Such conclusions are supported by our preliminary computational experiments. Therefore, we aim to solve this problem via heuristic algorithms. In the next sessions, we first show how to linearise the gliders' EOMs subject to error bounding constraints, followed by the presentation of the proposed STO algorithms. Next, we illustrate how our multi-phase method can be integrated into a routing matheuristic.

4. Linearisation of the equations of motion

The glider's EOMs (1) completely describes the aerodynamics of the glider under the influence of wind in a 3D environment. However, in Section 3.1 we imply (based on preliminary computational experiments) that embedding these dynamics into a routing mathematical formulation leads to a computationally intractable model, probably due to highly non-convex constraints.

In order to simplify Equation (1) into a more numerically tractable form, we employ a linearisation technique based on the steady-state conditions computed in Section 2.2. Several results in the literature show how linear dynamic equations can be used to solve TO problems (e.g., Richards et al., 2002; Keviczky et al., 2008; How et al., 2015).

For the sake of simplicity, in this section indices i, j and g will be omitted on state, control and time variables. Let $\mathbf{y}_{eq}$ and $\mathbf{u}_{eq}$ be some equilibrium state and controls of arc $(i, j) \in A$. By defining auxiliary variables $\delta\mathbf{y}(\tau) = \mathbf{y}(\tau) - \mathbf{y}_{eq}$ and $\delta\mathbf{u}(\tau) = \mathbf{u}(\tau) - \mathbf{u}_{eq}$ as perturbations of equilibriums of state and control variables, one can apply the first-order Taylor expansion, here denoted by $T(.)$, to the system dynamics described in Equation (1). This leads to the following expression:

$$T(\mathbf{y}_{eq}, \mathbf{u}_{eq}, \delta\mathbf{y}, \delta\mathbf{u}, \tau) = f(\mathbf{y}_{eq}, \mathbf{u}_{eq}, \tau) + \frac{\partial f(\mathbf{y}_{eq}, \mathbf{u}_{eq}, \tau)}{\partial \mathbf{y}} \delta\mathbf{y}(\tau) + \frac{\partial f(\mathbf{y}_{eq}, \mathbf{u}_{eq}, \tau)}{\partial \mathbf{u}} \delta\mathbf{u}(\tau). \tag{29}$$

By considering the characterisation of steady-states in Equation (2), the first term of Equation (29) equals zero by definition. We disregard the higher-order terms of Taylor's expansion for convenience.

Matrices $J^y = \frac{\partial f(\mathbf{y}_{eq}, \mathbf{u}_{eq}, \tau)}{\partial \mathbf{y}}$ and $J^u = \frac{\partial f(\mathbf{y}_{eq}, \mathbf{u}_{eq}, \tau)}{\partial \mathbf{u}}$ represent the Jacobians of the EOMs (1) with respect to state and control variables. Hence, we can approximate the EOMs (10) of the glider g flying through arc (i, j) by the linear system dynamics in state-space form as shown in Equation (30).

$$\dot{\mathbf{y}} = J^y \delta\mathbf{y}(\tau) + J^u \delta\mathbf{u}(\tau), \qquad \mathbf{y}(\tau^o) = \mathbf{y}^o, \mathbf{u}(\tau^o) = \mathbf{u}^o. \tag{30}$$

The linear EOMs (30) are expected to be a good approximation of Equation (1), provided that the glider is restricted to small variations around the equilibrium conditions $\mathbf{y}_{eq}$ and $\mathbf{u}_{eq}$. A simple numerical integration experiment can be carried out to show that constraining the control variables $\mathbf{u}$ to small perturbations around $\mathbf{u}_{eq}$ leads to a satisfactory approximation of the actual system dynamics. In fact, our preliminary computational experiments showed that adding small perturbation constraints for $\mathbf{u}$ into our optimisation framework is sufficient to decrease approximation errors to such an extent that they can be disregarded. The following theorem provides a justification for our approach.

Theorem 1 (Bounding the linearisation error)**.** *Consider the initial value problem consisting of the ordinary differential equation (1) and the initial value $\mathbf{y}(\tau_0) = \mathbf{y}_0$, for a given continuous control function $\mathbf{u}$ and value $\mathbf{y}_0$. Suppose that f is continuous and that $\mathbf{y}$ is a solution to this problem in some interval $I := [\tau_0 - \alpha, \tau_0 + \alpha]$ with some $\alpha > 0$. Likewise, consider the initial value problem*

$$\dot{\tilde{\mathbf{y}}} = J^y(\tilde{\mathbf{y}}(\tau) - \mathbf{y}_{eq}) + J^u(\tilde{\mathbf{u}}(\tau) - \mathbf{u}_{eq}), \qquad \tilde{\mathbf{y}}(\tau_0) = \tilde{\mathbf{y}}_0 \tag{31}$$

with some continuous control function $\tilde{\mathbf{u}}$ and value $\tilde{\mathbf{y}}_0$. Then, (31) has a unique solution $\tilde{\mathbf{y}}$. Suppose further that there exists some $\omega \geq 0$ with

$$\|f(\mathbf{y}, \mathbf{u}(\tau), \tau) - J^y(\mathbf{y} - \mathbf{y}_{eq}) - J^u(\tilde{\mathbf{u}}(\tau) - \mathbf{u}_{eq})\| \leq \omega \tag{32}$$

for all $(\mathbf{y}, \tau)$ from some compact set $\{\mathbf{y} : \|\mathbf{y} - \mathbf{y}_{eq}\| \leq \beta\} \times I$. Let $L := \|J^y\|$. We then have the error estimate

$$\|\mathbf{y}(\tau) - \tilde{\mathbf{y}}(\tau)\| \leq \|\mathbf{y}_0 - \tilde{\mathbf{y}}_0\| e^{L|\tau - \tau_0|} + \frac{\omega}{L}(e^{L|\tau - \tau_0|} - 1) \tag{33}$$

for all τ from some subinterval of I.

We omit the proof of Theorem 1 since it follows directly from the Picard-Lindelöf theorem (Teschl, 2012).

4.1. Numerical integration and bounding approximation errors

Traditional TO methods can be classified as *indirect* or *direct*. Indirect methods usually provide solutions with higher accuracy but need good starting guesses. On the other hand, direct methods (e.g., *direct collocation* or *direct transcription* methods), do in general not need good starting guesses and are thus more popular for complex problems with path constraints. They work by discretising the equations of motion by means of some numerical integration procedure and embedding the discretised differential equations into the nonlinear optimisation problem. More information about numerical algorithms for solving TO problems can be found, e.g., in Betts (2001).

In this paper, we employ a direct collocation method for optimising the trajectories of gliders. Let $\mathbf{y}(\tau)$ and $\mathbf{u}(\tau)$ be the state and control variables of a glider g flying through arc (i,j), where $\tau \in [\tau^o, \tau^f]$, and let $T = \{0, \ldots, N-1\}$ be the set of N *collocation points* (or time indices). By discretising the time interval $[\tau^o, \tau^f]$ over T, we can define a uniform time grid $\tau_t = \tau_o + \eta t$, in which the index $t \in T$ represents a time instant τ_t within the interval $[\tau^o, \tau^f]$ and $\eta = \frac{\tau_f - \tau_o}{N}$ is a uniform step size. Let $\mathbf{y}_t$ and $\mathbf{u}_t$ represent the approximations of the state and control, respectively, at time τ_t.

Preliminary experiments suggest that Euler's discretisation usually leads to simpler optimisation problems that can be solved efficiently by existing solvers. However, we highlight that the proposed approach can be easily adapted to employ any other integration methods, such as the trapezoidal or Runge-Kutta methods. The Euler method applied to the linear dynamical system defined by Equation (30) leads to the following discretised EOMs, with predefined initial conditions $\mathbf{y}^o$ and $\mathbf{u}^o$:

$$\mathbf{y}_{t+1} = \mathbf{y}_t + \eta(J^y \delta \mathbf{y}_t + J^u \delta \mathbf{u}_t) + \boldsymbol{\varepsilon}_t, \forall t \in T, \tag{34}$$

$$\mathbf{y}_0 = \mathbf{y}^o, \mathbf{u}_0 = \mathbf{u}^o. \tag{35}$$

Theorem 2 (Bounding the local integration error)**.** *Consider the initial value problem defined by Equations* (30)*. Assume that* $\dot{\mathbf{y}}$ *is continuous for all* $\mathbf{y}(\tau)$, $\mathbf{u}(\tau)$ *and* $\tau \in [\tau^o, \tau^f]$*. We denote the* ∞*-norm by* $\|.\|$*. The* local truncation error $\boldsymbol{\varepsilon}_t$ *at the* t*-th Euler's step can then be bounded by*

$$\|\boldsymbol{\varepsilon}_t\| \leq \frac{1}{2}\eta^2 \left(\|J^y\| \|\dot{\mathbf{y}}_{ub}\| + \|J^u\| \|\dot{\mathbf{u}}_{ub}\|\right), \forall t \in T. \tag{36}$$

Here, $\dot{\mathbf{y}}_{ub} = (\dot{x}_{ub}, \dot{y}_{ub}, \dot{h}_{ub}, \dot{v}_{ub}, \dot{\gamma}_{ub}, \dot{\phi}_{ub})^\top$ *and* $\dot{\mathbf{u}}_{ub} = (\dot{Cl}_{ub}, \dot{\mu}_{ub})^\top$ *define upper bounds on the values of the derivatives of state and control variables, respectively.*

Proof. By differentiating the continuous EOMs (30), we obtain

$$\ddot{\mathbf{y}} = \frac{\partial \dot{\mathbf{y}}}{\partial \tau} = J^y \dot{\mathbf{y}} + J^u \dot{\mathbf{u}}.$$

By definition, J^y and J^u are bounded. This means that there exists an $M \in \mathbb{R}^+$ such that

$$\|J^y \dot{\mathbf{y}} + J^u \dot{\mathbf{u}}\| = \|\ddot{\mathbf{y}}\| \leq M, \quad \tau \in [\tau^o, \tau^f]. \tag{37}$$

Since $\tau_{t+1} = \tau_t + \eta$, Taylor's theorem implies that

$$\mathbf{y}(\tau_{t+1}) = \mathbf{y}(\tau_t) + \eta f(\mathbf{y}(\tau_t), \mathbf{u}(\tau_t), \tau) + \frac{1}{2}\eta^2 \ddot{\mathbf{y}}(\tilde{\tau}), \tau_t < \tilde{\tau} < \tau_{t+1}. \tag{38}$$

Comparing Equation (38) with Equation (34) shows that:

$$\boldsymbol{\varepsilon}_t = \frac{1}{2}\eta^2 \ddot{\mathbf{y}}(\tilde{\tau}), \tau_t < \tilde{\tau} < \tau_{t+1}.$$

Recalling Expression (37), the following inequality is thus valid:

$$\|\boldsymbol{\varepsilon}_t\| \leq \frac{1}{2}\eta^2 \left\|(J^y\dot{\mathbf{y}}_{ub} + J^u\dot{\mathbf{u}}_{ub})\right\| \leq \frac{1}{2}\eta^2 \left(\|J^y\|\|\dot{\mathbf{y}}_{ub}\| + \|J^u\|\|\dot{\mathbf{u}}_{ub}\|\right) \leq \frac{1}{2}\eta^2 M, \forall t \in T. \tag{39}$$

□

Within the algorithms proposed in this paper, estimate values for $\dot{\mathbf{y}}_{ub}$ have been empirically computed by using Matlab to simulate the flight of a glider under the steady-state conditions computed in Section 2. In addition, estimate values for $\|\dot{\mathbf{u}}_{ub}\|$ can usually be found in the flight dynamics literature, for example, as in Mustapa & Saat (2016).

4.2. Reformulation of the infinite-dimensional problem

Following the discretisation of the glider's EOMs it is necessary to reformulate the infinite-dimensional TO problem presented in Section 3.1 as a discrete-time Non-linear Programming (NLP) problem. With the introduction of new variables representing the error term $\boldsymbol{\varepsilon}_{ijgt}, i, j \in V, g \in G, t \in T$, the objective function (11) is reformulated to penalise the solution error in the new objective function:

$$\min \left\{ \max_{g\in G} \left\{ \sum_{(i,j)\in A} (\tau^f_{ijg} - \tau^o_{ijg}) a_{ijg} \right\} + p \sum_{g\in G} \sum_{(i,j)\in A} \sum_{t\in T} \|\boldsymbol{\varepsilon}_{ijgt}\| a_{ijg} \right\}. \tag{40}$$

Here, $p > 0$ is a fixed penalty parameter that also serves as a constant conversion factor.

Theorem 2 allows us to bound the error terms as in Constraint (41) below.

$$\|\boldsymbol{\varepsilon}_{ijgt}\| \leq \frac{1}{2}\eta^2 \left(\|J^{y_{ijg}}\|\|\dot{\mathbf{y}}_{ub}\| + \|J^{u_{ijg}}\|\|\dot{\mathbf{u}}_{ub}\|\right), \forall i, j \in V, g \in G, t \in T \tag{41}$$

Here, we denote by $J^{y_{ijg}}$ and $J^{u_{ijg}}$ the Jacobians, with respect to state and control variables, of the EOMs of glider g flying from waypoint i to waypoint j. For the sake of being succinct, in this section, we will omit the reformulation of the remaining constraints. These will be fully stated in Section 5 where the TO subproblems that we solve will be defined.

4.3. Interpolation of the discretised solutions

Euler integration steps replace the controls and system dynamics by piece-wise linear approximations. Different integration methods might employ different approximations though. Therefore, after solving the optimisation problem NLP one needs to reconstruct the controls' and system dynamics' trajectories. In this paper, this is done by using a linear interpolation of the control variables, which is appropriate for Euler's method. Let us define the independent variable τ in terms of subsequent time steps t and $t+1$, i.e., $\tau \in [\tau_t, \tau_{t+1}]$. An approximation for the control function can be written as in

$$\mathbf{u}(\tau) = \mathbf{u}_t + \frac{\tau - \tau_t}{\eta}(\mathbf{u}_{t+1} - \mathbf{u}_t). \tag{42}$$

The approximation for the system dynamics can likewise be defined as a linear function.

5. Heuristic algorithms for trajectory optimisation

In this section, we propose efficient TO heuristics for finding a feasible glider's trajectory passing through the sequence of waypoints defined by a given route. Even for a fixed route, solving a TO problem involving a set of waypoints is often a challenging and computationally expensive task (Fisch, 2011). For this reason, the heuristics developed in this paper are based on the decomposition of trajectories into arcs that can be solved independently, thus reducing the overall difficulty of the problem.

With this concept in mind, our so-called STO heuristic can be summarised as follows. For a given glider and its assigned route, we decompose the glider's trajectory into several phases, each one corresponding to an arc of the provided route. Next, each arc is modelled as a single-phase TO problem and then sequentially solved from the beginning to the end of the glider's route. Feasibility is maintained by linking the final conditions of each solved arc to the initial conditions of its subsequent one.

Two different methods are proposed for solving the single-phase TO subproblems. In the first approach, the subproblems are reformulated as NLP problems and directly solved by available NLP software. In the second approach, for a fixed flight duration, the corresponding NLP can be reformulated as a SOCP model. Next, arc flight times are heuristically minimised by a sequence of SOCPs. In the next sections, we discuss each component of our algorithms in detail.

5.1. General approach to the sequential trajectory optimisation heuristics

This section presents the STO heuristic developed to find feasible trajectories, i.e., feasible states and controls for a glider flying through several waypoints. A feasible trajectory is basically a solution to an infinite-dimensional problem defining the state and control variables of a dynamical system. Constructing a feasible trajectory under nonlinear constraints is, in general, a challenging problem (Zhou et al., 2017).

Our STO heuristic can be formally defined as follows. Let S be the set of all subsequences (or routes) of V starting at 0 and ending at a landing point in L, such that every waypoint is visited at most once. Without loss of generality, we can write an element r of S as $r = (0, i_1, \ldots, i_k, i_l), i_1, \ldots, i_k \in V$ and $i_l \in L$. The trajectory of a glider flying through route r is then divided into $|r| - 1$ phases according to each arc of the route. Next, each phase (arc) of the route is solved by means of a TO subproblem based on the linearised EOMs presented in Section 4. The arcs of route r are connected in such a way that the initial conditions of arc (j, k) equals the final conditions of arc (i, j) if waypoints i, j and k are present in route r in this specific order.

Figure 5 illustrates the procedure described above. In this example, we are asked to find a feasible trajectory for the route $r = (0, 1, 2, l)$, where $1, 2 \in V$ and $l \in L$. The initial conditions on arc $(0, 1)$ are set such that $\mathbf{y}^o_{01} = \mathbf{y}^o$, $\mathbf{u}^o_{01} = \mathbf{u}^o$ and $\tau^o_{01} = 0$, where $\mathbf{y}^o$ and $\mathbf{u}^o$ are known beforehand, see Figure 5(a). The shortest flight duration $(\tau^f_{01} - \tau^o_{01})$ on arc $(0, 1)$ and its respective trajectory $(\mathbf{y}_{01}(\tau_{01}), \mathbf{u}_{01}(\tau_{01}))$ are computed by solving a TO subproblem as explained in Sections 5.2 and 5.3. If no optimal trajectory can be found for arc $(0, 1)$, the algorithm stops and the route is considered infeasible. Otherwise, the solution corresponding to the arc $(0, 1)$ is stored and the algorithm proceeds to the next arc, i.e., $(1, 2)$. The continuity of the trajectory along route r is maintained by setting the initial conditions of arc $(1, 2)$ equal to the final conditions of arc $(0, 1)$ as in Figure 5(b). After the last arc $(2, l)$ is processed as in Figure 5(c), the route total flight time corresponding to route r can be computed as $\tau^f_{2l} - \tau^o_{01}$.

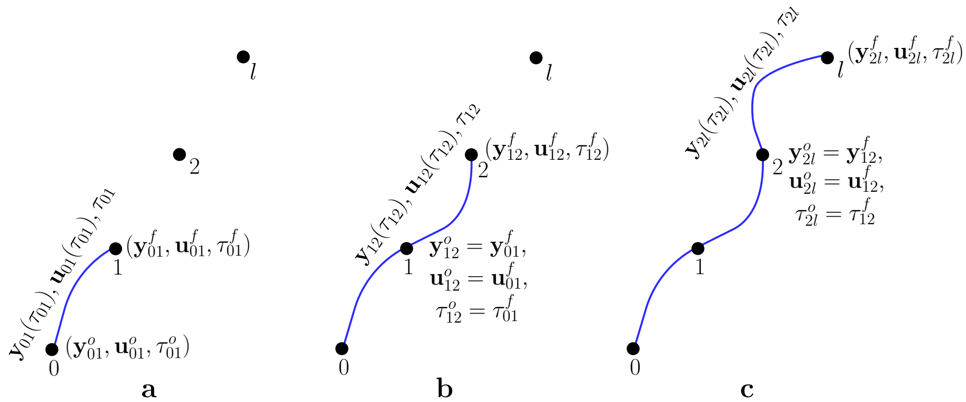


Figure 5: Illustration of the proposed STO approach for a small example with two waypoints.

Algorithm 1 provides a pseudo-code of the procedure to find feasible trajectories for a glider's route. The algorithm starts with the initialisation of the auxiliary variables $\bar{\mathbf{y}}$, $\bar{\mathbf{u}}$ and $\bar{\tau}$, and the initialisation of the route flight duration $f(r)$ (line 1). The main loop of our heuristic iterates over the arcs of route r (lines 3–13). Next, the status of the current optimisation is initialised (line 4). The state and control variables associated with arc $(i,j) \in r$ are initialised with an empty value (line 5). Next, the starting conditions $\mathbf{y}^o_{ij}$ and $\mathbf{u}^o_{ij}$, and initial flight time τ^o_{ij} associated with arc (i,j) are initialised (lines 6–4). The TO subproblem associated with arc $(i,j) \in r$ is solved by means of the `TrajectoryOptimisation()` routine (line 7) either by solving a NLP single-phase subproblem directly or by means of an iterative flight time minimisation algorithm based on a SOCP reformulation of the NLP subproblem. This step will be further explained in the next sections. If an optimal solution is obtained, the route flight time is incremented accordingly and the auxiliary coupling variables $\bar{\mathbf{y}}$, $\bar{\mathbf{u}}$ and $\bar{\tau}$ are updated (lines 9–10) thus maintaining the continuity of the trajectory associated with route r. Otherwise, the algorithm terminates and a very large route flight duration is returned (line 12). If an optimal trajectory is found for every arc of route r, the algorithm returns the route's flight time (line 13).

Algorithm 1 STO

1: Procedure STO(r)
2: $\bar{\mathbf{y}} \leftarrow \mathbf{y}^o$; $\bar{\mathbf{u}} \leftarrow \mathbf{u}^o$; $\bar{\tau} \leftarrow 0$ $f(r) \leftarrow 0$
3: **for** each arc (i,j) in route r **do**
4:     *status* $\leftarrow$ not optimal
5:     $\mathbf{y}_{ij}(\tau_{ij}) \leftarrow \texttt{NULL}$; $\mathbf{u}_{ij}(\tau_{ij}) \leftarrow \texttt{NULL}$
6:     $\mathbf{y}^o_{ij} \leftarrow \bar{\mathbf{y}}$; $\mathbf{u}^o_{ij} \leftarrow \bar{\mathbf{u}}$; $\tau^o_{ij} \leftarrow \bar{\tau}$
7:     $[status, \tau^f_{ij}, \mathbf{y}_{ij}(\tau_{ij}), \mathbf{u}_{ij}(\tau_{ij}), \boldsymbol{\varepsilon}_{ij}(\tau_{ij})] \leftarrow \texttt{TrajectoryOptimisation}(\mathbf{y}^o_{ij}, \mathbf{u}^o_{ij}, \tau^o_{ij})$
8:     **if** *status* = optimal **then**
9:         $f(r) \leftarrow f(r) + \tau^f_{ij}$
10:         $\bar{\mathbf{y}} \leftarrow \mathbf{y}^f_{ij}, \bar{\mathbf{u}} \leftarrow \mathbf{u}^f_{ij}, \bar{\tau} \leftarrow \tau^f_{ij}$
11:     **else**
12:         **return** ∞
13: **return** $f(r)$
14: **end** STO.

5.2. Single-phase nonlinear trajectory optimisation subproblem

In this section, we are interested in finding an optimal trajectory for a glider g flying through a single arc $(i,j) \in A$. In our first method, the `TrajectoryOptimisation()` routine (line 7 of STO) attempts to find such optimal trajectory by directly solving its corresponding NLP formulation. Hereafter, we will refer to this version of our algorithm as STO-NLP. For the sake of simplicity, the i, j and g subscripts on state, control and time variables will be omitted for the rest of this section. The glider's flight is governed by the linearised EOMs (34), subject to the initial conditions of arc (i,j), represented by (35), and the upper bounds on the norms of the errors provided by Inequalities (36). Note that, since the arc's flight time τ^f (and therefore η) is unknown, EOMs (34) are still non-linear.

By assuming w.l.o.g. that waypoints are always visited in the last time step (denoted as t^f), one can define the visiting constraints for the glider flying to any waypoint in set V through Inequalities (43–46). More precisely, Constraints (43) and (44) ensure that the glider will lie within the waypoint's boundaries at the last time step corresponding to the arc (i,j), whereas Constraints (45) and (46) guarantee that the glider will be in an appropriate attitude to photograph the waypoint, i.e. the final yaw and roll angles will be within a small interval predetermined, respectively, by parameters $\hat{\gamma}$ and $\hat{\mu}$:

$$(x_{t^f} - \bar{x})^2 + (y_{t^f} - \bar{y})^2 \leq (h_{t^f} + \bar{r})^2 \tag{43}$$

$$\underline{h} \leq h_{t^f} \leq \overline{h} \tag{44}$$

$$-\hat{\gamma} \leq \gamma_{t^f} \leq \hat{\gamma} \tag{45}$$

$$-\hat{\mu} \leq \mu_{t^f} \leq \hat{\mu}. \tag{46}$$

The constraint regarding landing sites can be defined similarly, i.e., for any arc $(i,j), j \in L$, Constraint (47) ensures that the glider is within the landing site's covering region at the last time step t^f in that arc:

$$(x_{t^f} - \tilde{x})^2 + (y_{t^f} - \tilde{y})^2 + h_{t^f}^2 \leq \tilde{r}^2. \tag{47}$$

As discussed in Section 4, the glider's EOMs have been linearised around some steady-state conditions. In order to reduce the errors associated with the linearisation and discretisation of the EOMs, we introduce small-perturbation constraints, as in Inequalities (48) and (49), on the control variables Cl (lift coefficient) and μ (bank angle). Experiments showed that these constraints help to reduce the linearisation errors without compromising our algorithm's performance. With $\alpha_t = \frac{t}{N-1}$, the small perturbation constraints are defined as

$$\alpha(Cl_{eq} - \delta) + (1-\alpha)Cl_{lb} \leq Cl_t \leq \alpha(Cl_{eq} + \delta) + (1-\alpha)Cl_{ub}, \forall t \in T, \tag{48}$$

$$\alpha(\mu_{eq} - \delta) + (1-\alpha)\mu_{lb} \leq \mu_t \leq \alpha(\mu_{eq} + \delta) + (1-\alpha)\mu_{ub}, \forall t \in T. \tag{49}$$

By means of Constraints (48) and (49), at the first time steps associated with arc (i,j) the glider's control variables are allowed to vary between their lower and upper bounds. As time progresses, i.e. the value of t increases, these bounds approach the control's steady-state values, thus reducing the EOMs linearisation/discretisation errors.

The TO subproblem associated with arc (i,j) can be summarised as follows. The objective function (50) reformulates Objective (40) for a single arc. Constraints (34–35) define the discretised EOMs of the glider, while Constraints (36) bound the error associated with discretisation. W.l.o.g. we set $\tau^o = 0$ for any arc (i,j). Constraint (51) defines the discretisation step size. In order to avoid arbitrarily small flight times within the allowed error, Constraint (52) limits τ^f from below. This lower limit is computed as the Euclidean distance between the waypoints' locations divided by the maximum glider's airspeed. If the end point of arc (i,j) belongs to the set of waypoints, our optimisation subproblem includes visiting Constraints (43–46). Otherwise, if the ending point is a landing site, i.e., $j \in L$, our subproblem includes Constraint (47) instead. We also include the small perturbation Constraints (48–49). Finally, Constraints (53) and (54) define the domain of the optimisation variables.

$$\begin{array}{llll}
(NLP) & \min & \tau^f + \sum_{t \in T} \|\boldsymbol{\varepsilon}_t\| & (50) \\
& \text{s.t.} & (34\text{–}35) & \\
& & (36) & \\
& & \eta = \dfrac{\tau^f}{N-1} & (51) \\
& & \tau^f \geq \bar{\tau}^f & (52) \\
& & (43\text{–}46) \text{ or } (47) & \\
& & (48\text{–}49) & \\
& & \boldsymbol{\varepsilon}_t, \mathbf{y}_t \in \mathbb{R}^6, \mathbf{u}_t \in \mathbb{R}^2, \forall t \in T & (53) \\
& & \tau^f, \eta \in \mathbb{R} & (54)
\end{array}$$

5.3. Iterative flight time minimisation based on a single-phase SOCP subproblem

In the last section we presented a direct collocation method to optimise the trajectory of a single glider flying through a given arc $(i,j) \in A$. However, as we will empirically show in Section 7, finding such trajectories is a computationally expensive task. Therefore, in this section, we propose an iterative method for minimising the flight time by solving a series of SOCP subproblems. We will refer to this version as *Iterative TO* (or `i-TrajectoryOptimisation()`). The so-called Iterative Sequential Trajectory Optimisation (i-STO) version of our STO algorithm substitutes the `TrajectoryOptimisation()` subroutine (Algorithm 1, line 7) by the `i-TrajectoryOptimisation()` procedure (Algorithm 2).

For a fixed flight time τ^f, subproblem *NLP* can be reformulated as the *SOCP* subproblem below. The main differences between problem *SOCP* and *NLP* lie in the objective function (55), which only involves the minimisation of the sum of the norm of errors ($\boldsymbol{\varepsilon}_t$), and the absence of constraints (51) and (52), since the final time τ^f is a known parameter in *SOCP*.

$$
\begin{aligned}
(SOCP) \quad \min \quad & \sum_{t \in T} \|\boldsymbol{\varepsilon}_t\| && (55)\\
\text{s.t.} \quad & (34\text{–}35)\\
& (36)\\
& (43\text{–}46) \text{ or } (47)\\
& (48\text{–}49)\\
& (53\text{–}54)
\end{aligned}
$$

The `i-TrajectoryOptimisation()` method, depicted in Algorithm 2 works as follows. The algorithm starts with the initialisation of the arc's flight time τ^f with an estimated lower-bound for the flight duration (which might be infeasible), here denoted as $\bar{\tau}^f$ (line 2), and the initialisation of the *status* of the current arc (line 3). We estimate $\bar{\tau}^f$ as follows. For a given arc $(i,j) \in A$, let $\tilde{d}_{ij} = ||(\bar{x}_i, \bar{y}_i, \bar{h}_i) - (\bar{x}_j, \bar{y}_j, \bar{h}_j)||_2$ be an estimate for the flight distance from i to j. With the maximum airspeed v_{max}, we arrive at the estimate $\bar{\tau}^f = \tilde{d}_{ij}/v_{max}$. While τ^f is not greater than an upper limit $\tau^{f,ub}$ (lines 4–8), the SOCP subproblem is solved for the fixed τ^f (line 5). If an optimal trajectory is found (line 6), the algorithm halts and returns the optimal solution (line 9). Otherwise, the arc's estimated minimum flight time is increased by $\delta\tau$ and a new SOCP round is attempted. However, if no optimal solution is found, the `while` loop finishes and `status` $\leftarrow$ `not optimal` is returned to STO.

Algorithm 2 Iterative TO

1: Procedure `i-TrajectoryOptimisation`$(\mathbf{y}^o, \mathbf{u}^o, \tau^o)$
2: $\tau^f \leftarrow \bar{\tau}^f$
3: *status* $\leftarrow$ not optimal
4: **while** $\tau^f \leq \tau^{f,ub}$ **do**
5: $\quad [status, \tau^f, \mathbf{y}(\tau), \mathbf{u}(\tau), \boldsymbol{\varepsilon}(\tau)] \leftarrow \text{SOCP}(\mathbf{y}^o, \mathbf{u}^o, \tau^o, \tau^f)$
6: $\quad$ **if** *status* = optimal **then**
7: $\quad\quad$ **brake**
8: $\quad \tau^f \leftarrow \tau^f + \delta\tau$
9: **return** $[status, \tau^f, \mathbf{y}(\tau), \mathbf{u}(\tau), \boldsymbol{\varepsilon}(\tau)]$
10: **end** `i-TrajectoryOptimisation`.

6. A matheuristic routing algorithm

This section describes our proposed matheuristic algorithm for solving the GRTOP, called Iterated Local Search (ILS)-STO. Our approach consists of a multi-start ILS (Lourenço et al., 2010) combined with a Set Partitioning (SP) integer programming formulation, and the STO heuristics described in Section 5.

As before, for a given arc $(i,j) \in A$, let $\tilde{d}_{ij} = ||(\bar{x}_i, \bar{y}_i, \bar{h}_i) - (\bar{x}_j, \bar{y}_j, \bar{h}_j)||_2$ be an estimate for the flight distance from i to j. Let $\tilde{v}_{ij}$ be the equilibrium airspeed (v_{eq}, as computed in Section 2) between i and j. We estimate the flight time between i and j as $\tilde{\tau}_{ij} = \tilde{d}_{ij}/\tilde{v}_{ij}$. Hence, the total flight time of an arbitrary route r can be estimated by

$$\tilde{f}(r) = \sum_{(i,j)\in r} \tilde{\tau}_{ij}. \tag{56}$$

Note that determining the value of $\tilde{f}(r)$ is trivial since each $\tilde{\tau}_{ij}, (i,j) \in A$, can be pre-computed in constant time. On the other hand, estimating approximation error values of a given route does not appear to be trivial. Therefore, such errors are not taken into consideration at this step. The estimate cost of a solution s is thus computed as $\tilde{g}(s) = \max_{r\in s} \tilde{f}(r)$.

Computing the actual flight time and error values between waypoints i and j is computationally expensive, because it requires solving the corresponding TO subproblems, as discussed in Section 5. Therefore, we resort to the estimation described above while generating an initial solution and performing local search to improve the scalability of the method. Yet, we are still forced at times to compute the actual flight times and error values to then compute the actual (or true) objective value of a solution s as follows:

$$g(s) = \max_{r\in s}\{f(r)\} + \sum_{r\in s}\sum_{(i,j)\in r}\sum_{t\in T} \boldsymbol{\varepsilon}_{ijt}. \tag{57}$$

We choose to limit the computation of Expression (57) to solutions that our routing heuristic has identified as locally optimal. After each successful trajectory computation, the estimate flight time matrix is updated with the corresponding actual (true) flight times of solution s. Therefore, the flight time matrix tends to converge to its actual values as the algorithm progresses.

Algorithm 3 shows the pseudocode of ILS-STO. The matheuristic procedure performs I_R restarts (lines 3–17) where at each of them an initial solution is generated using a greedy randomised insertion heuristic (line 4). The method iteratively tries, for I_{ILS} iterations, to improve the initial solution employing local search (line 8) and perturbation (line 14) mechanisms using the estimate route costs $\tilde{f}(\cdot)$. The local search algorithm is described in Section 6.2, while the perturbation mechanism is described in Section 6.3. The actual cost of locally optimal solutions is computed using the aforementioned STO heuristics and the estimate flight time matrix is updated from the true solution values stored in s (line 9). Recall that computing (57) actually requires the solution of a sequence of nonlinear TO subproblems. Solving TOPs is an inherently difficult task, therefore we expect step 9 to be the main bottleneck of our framework. If a solution s is improved after the local search phase, then the best current solution s' is updated (lines 10–12). Note that within local search (line 8) solutions are evaluated using $\tilde{f}(\cdot)$. Next, the pool of routes (*Rpool*) is updated by adding the feasible routes from s (line 13). The best solution s^* found after each restart is updated in lines (16–17), if necessary. Finally, the algorithm tries to find the optimal combination of feasible routes stored in *Rpool* by solving a SP-based problem (see Section 6.4) using a general purpose Mixed-Integer Linear Programming (MILP) solver (line 18).

6.1. Constructive procedure

Two insertion strategies, sequential and parallel, and two insertion criteria, nearest and cheapest, are employed in the constructive procedure. At each restart, one strategy and criterion are chosen at random. In the sequential strategy, one individual route is considered at each insertion iteration sequentially, whereas, in the parallel insertion, all routes are candidates to receive an unrouted waypoint. The nearest insertion criterion selects the closest inserted waypoint for all unrouted ones and the insertion is performed immediately after it in the corresponding route. The cheapest insertion computes the minimum insertion cost and the waypoint associated with such a value is selected to be inserted in the corresponding position. The routes generated by the constructive procedure will be used in the next step (local search) of our algorithm.

Algorithm 3 ILS-STO

Procedure $\texttt{ILS-STO}(I_R, I_{ILS})$
$s^* \leftarrow \texttt{NULL}; g^* \leftarrow \infty$; *Rpool* $\leftarrow$ NULL
for i:=$1, \ldots, I_R$ **do**
  $s \leftarrow$ `GenerateInitialSolution()`
  $s' \leftarrow$ auxiliary solution with very large cost ($g(s') \leftarrow \infty$)
  iter $\leftarrow 0$
  while *iter* $\leq I_{ILS}$ **do**
    $s \leftarrow$ `LocalSearch`(s)
    Compute $g(s)$ as in Expression (57) and update flight time and error matrices
    if $g(s) < g(s')$ **and** s is feasible **then**
      $s' \leftarrow s$
      iter $\leftarrow 0$
    Rpool $\leftarrow$ *Rpool* $\cup$ feasible routes from s
    $s \leftarrow$ `Perturb`(s')
    iter $\leftarrow$ *iter* $+ 1$
  if $g(s') < g^*$ **then**
    $s^* \leftarrow s'$; $g^* \leftarrow g(s')$
$s^* \leftarrow$ `SP`(s^*, *Rpool*); $g^* \leftarrow g(s^*)$
return s^*, g^*
end ILS-STO.

6.2. Local search

We apply a Randomized Variable Neighbourhood Descent (RVND) procedure (Mladenovic & Hansen, 1997; Subramanian, 2012) for the local search. Let $\mathcal{N}$ be the set of inter-route neighbourhoods, that is, those involving more than one route. We recall that a neighbourhood η of a solution s is a set of solutions that are close to s in a given search domain space. RVND starts by randomly choosing a neighbourhood $\eta \in \mathcal{N}$, and determining the best improving move. If no improvements have been found, then a neighbourhood other than η is selected at random and so on until all neighbourhoods fail to improve the best current solution. In case of improvement, then an RVND search is executed in the modified routes only, with intra-route neighbourhoods, that is, operators involving moves within the route.

The classical vehicle routing inter-route neighbourhood operators implemented are:

- `Shift(1,0)` — A waypoint is moved from one route to another one.
- `Swap(1,1)` — A waypoint from one route is interchanged with a waypoint from another route.
- `2-opt`* — Two arcs are removed, one from each pair of routes, and two arcs are inserted in such a way that two new routes are formed.

Note that the neighbourhood operators mentioned above are only applied when at least one of the routes has an estimate flight time $\tilde{f}(.)$ that is the estimate maximum route flight time of the current solution (i.e., the $\tilde{f}(.)$ corresponding to the makespan). In the case of `Shift(1,0)`, we only consider moving a waypoint from the route whose value $\tilde{f}(.)$ matches the maximum. Any other move would not lead to an improvement in the makespan.

The following intra-route Travelling Salesman Problem (TSP)–based neighbourhood moves were implemented:

- `Reinsertion` — One waypoint is removed and reinserted in another position of the route.
- `Exchange` — Two waypoints are interchanged.
- `2-opt` — Two arcs are removed and another two are inserted to form a new route.

The neighbour solutions associated with the intra-route operators are evaluated in the same way as in the TSP. After performing the intra-route local search, the algorithm updates, if necessary, the estimate maximum flight duration.

6.3. Perturbation

Perturbations are necessary to escape local optimum solutions. In this paper, we apply simple yet effective perturbation operators as follows. If a solution s contains more than one feasible route, we apply random `Shift(1,0)` or `Swap(1,1)` operators between the makespan route and any other route from s chosen at random. Otherwise, if there is only one feasible route in a solution, we perform a random shuffle in a partial sequence of this route to obtain a new perturbed solution. The mechanisms described above are repeated a random number of times to achieve an effective perturbation step.

6.4. A Set Partitioning-based approach

Recall that $f(r)$ is the actual (true) flight time of feasible route $r \in S$, being S the set of all routes of V' starting at 0 and ending at $l \in L$. Also, let $S_i \subseteq S$ be the set of all feasible routes containing the waypoint $i \in V$. Define y_r as a binary variable that assumes value 1 if a route $r \in S$ is in the solution and 0 otherwise, and f^* as the variable associated with the makespan. Also, let $\varepsilon(r)$ represent the error associated with route r. We can now write an SP-based formulation for the GRTOP as described in Section 3 as follows:

$$\min \quad f^* + \sum_{r \in S} \|\boldsymbol{\varepsilon}(r)\| y_r \tag{58}$$

$$\text{s.t.} \quad \sum_{r \in S_i} y_r = 1 \qquad i \in V \tag{59}$$

$$\sum_{r \in S} y_r \leq n_g \tag{60}$$

$$f^* \geq f(r) y_r \qquad r \in S \tag{61}$$

$$y_r \in \{0, 1\}. \qquad r \in S \tag{62}$$

Objective function (58) minimises the total solution cost as defined in Objective (40). Constraints (59) state that there should be exactly one route associated with each waypoint $i \in V$. Constraint (60) imposes an upper bound on the number of gliders. Constraints (61) are responsible for the makespan computation. Finally, Constraints (62) define the domain of the variables. Since it is prohibitively expensive to determine all feasible routes, we solve a restricted version of formulation (58–62) which is composed of a subset of all feasible routes obtained by ILS-STO.

7. Computational experiments

In this section, the computational performance of the proposed TO-based algorithms is evaluated, namely the i-STO and the STO-NLP variants. Moreover, we assess the performance of the integrated routing and TO (ILS-STO) algorithm. In Section 7.1, we introduce the test instances used in our computational experiments. Section 7.2 presents a comparison between the proposed algorithms and the performance of different NLP software on the solution of TO subproblems. In Section 7.3, experiments regarding the influence of the discretisation size on the proposed algorithms' performance are carried out. The influence of the error bounding constraints on solution quality and computations is studied in Section 7.4. Finally, Section 7.5 presents the computational results of the proposed ILS-STO integrated matheuristic on the solution of the GRTOP.

Both versions of our TO heuristics and the integrated matheuristic ILS-STO were coded in C++ (and compiled with gcc v. 9.3.0) and executed on an Intel i7 CPU with 3.60GHz and 24GB of RAM running under Linux Mint 20.2 64bits (kernel 5.4.0-90-generic). The NLP subproblems described in Section 5.2 were coded using the `AMPL` (v. 20220323) modelling language through its C++ API (v. 2.0.6). A list of the employed NLP solvers is provided in Section 7.2. The SOCP subproblems described in Section 5.3 were coded and solved using the solver `CPLEX` (v. 12.7) limited to a single thread.

7.1. Benchmark instances

To better assess the performance of the proposed algorithms, computational experiments were performed across a range of different instances and under several scenarios. Here, we adopt the instances generated by Coutinho et al. (2019). Such instances contain from 2 to 8 waypoints and up to 2 landing sites and are defined over $1km^2$ (the so-called *small range*) and $25km^2$ (the so-called *medium range*) areas. In addition, we create a set of new large-range randomly generated instances with a large number of waypoints to test our algorithms more comprehensively. All generated instances as well as an instance generator coded in `Python` have been made available at Coutinho et al. (2022).

The generation of the proposed larger instances was carried out as follows. We have generated instances having $n \in \{10, \dots, 50\}$ waypoints and $m \in \{3, 4, 5\}$ landing zones. The maximum fleet size, i.e. the maximum number of available gliders, is computed as $n_g = \lfloor n/2 \rfloor$. This value was adopted in the experiments presented in Section 7.5. Table 2 shows the values of the parameters defining waypoints and landing zones as well as the dimensions of their containing volumes (units are indicated as necessary). Notation in Table 2 follows the one defined in Section 3. Five different instances were created for each combination of the number of waypoints and landing zones. Let $\mathcal{U}[a, b]$ denote the continuous uniform distribution from a to b. The launching altitude h_o, in km, has been chosen from $\mathcal{U}[4, 5]$. These limits on the launching altitude are based on the values used by Crispin (2016) for the same physical glider model flying over an area of $10km^2$. We will refer to the instances created in this section as *large range* instances (represented by "L" in the instance name). These instances span an airspace of $100km^2$, according to Table 2. For most of our experiments, the discretisation size N has been set to 50, unless otherwise indicated.

Table 2: Parameters defining the geometry of waypoints and landing zones.

Waypoints	a	b	Landing sites	a	b
$\bar{x}(km)$	0	10	$\tilde{x}(km)$	0	10
$\bar{y}(km)$	0	10	$\tilde{y}(km)$	0	10
$\bar{h}$	0	0	$\tilde{h}$	0	0
$\bar{r}(m)$	10	25	$\tilde{r}(m)$	10	25
$\underline{h}(m)$	50	100	$x_o(km)$	0	10
$\bar{\bar{h}}(m)$	200	300	$y_o(km)$	0	10

Due to the extensive number of generated instances, it is not possible to report all of our currently available results. Moreover, such descriptions would become tedious and cumbersome. Therefore, instances were grouped according to their class (small, medium and large range) and the number of waypoints. For example, `GRTOP-L10` represents the group of large range instances with 10 waypoints. Most results are reported in summary tables containing minimum, maximum, average results and standard deviations for each group of instances. Full computational results can be found in this paper's supplementary material.

7.2. Choosing an appropriate solver for the nonlinear subproblems

The main computational bottleneck of our routing algorithm lies in the repeated solution of NLPs which are TO problems representing flight dynamics in their constraints. Our approach is flexible enough to use any off-the-shelf NLP solver or a specially designed algorithm like i-STO. Comparing different solvers is not a trivial task, as various performance criteria like solution feasibility, optimality, and computing times need to be taken into account (Pintér & Kampas, 2013). In this section, we compare both versions of our STO algorithm with each other, where we use a variety of different NLP software within STO-NLP (STO-NLP). Accordingly, these NLP solvers have to solve a variety of problems as described in Subsection 5.2. For our tests we used the global optimisation solvers `BARON` (v. 21.1.13), `LGO` (v. 2015-01-17), `Octeract Engine` (v. 3.5.0) and `Couenne` (v. 0.5.7), and the local solvers `WORHP` (v. 1.14) and `IPOPT` (v. 3.12.13). As already stated, the SOCP subproblems generated by i-STO were solved by `CPLEX` (v. 12.7).

The experiments presented here were executed as follows. For each instance of each group, a random route was chosen and provided as input to the TO algorithms. For example, considering instances from the group `GRTOP-S8`, routes containing a random permutation of 8 waypoints and a random final landing site were generated and used as input to i-STO and STO-NLP. Here the discretisation size was set to $N = 50$ on each arc of the random routes. A time limit of 60s was imposed on all solvers for the solution of each subproblem.

Table 3 shows the results of our experiments. Regarding STO-NLP, we highlight that solvers `BARON`, `LGO`, `Couenne` and `WORHP` all failed to yield feasible solutions within the specified time limit. These were, therefore, omitted from our reports. In addition, we point out that STO-NLP's version running `Octeract-engine` was not able to solve all the instances from each group. Table 3 is organised in the following way. Column **Group** shows each group of instances organised by the number of waypoints and class. Average numbers per group of instances are presented in this table, but only for the ones in which a feasible solution was found. Next, column **Est.** shows a lower limit on the total flight time of the provided random route, computed as explained in Section 5.2. Column **#Oct.** describes how many instances from each group were solved by STO-NLP when the solver `Octeract-engine` was employed. This is to emphasise that only i-STO and STO-NLP (running `IPOPT`) were capable of finding solutions for all instances. The next columns are organised into subgroups showing the performance of each algorithm regarding flight times, errors, step sizes and computing times, respectively. Flight times (subgroup **Flight**) are reported as the sum of flight times for each arc of a route. Errors (subgroup **Error**) are calculated as the sum of the maximum error (i.e., the sum of the ∞-norm of $\boldsymbol{\varepsilon}$), among all state variables, for each discretised time interval and each arc. For example, the errors for group `GRTOP-L50` are shown as the sum of N times 51 maximum error terms along the whole trajectory. Step sizes (subgroup **Step**) are presented as the average step size along the provided route, i.e., the sum of each arc's step size divided by the route's length. Finally, subgroup **Time** presents the total running times of the proposed algorithms. For each class of instances, the respective overall minimum, maximum, average and standard deviation values are shown in rows **Group min.**, **Group max.**, **Group avg.** and **Group std.**, respectively. Bold numbers represent the minimum values among a subgroup of columns. The character "-" is shown if no results were obtained for the corresponding group of instances.

Table 3: Computational performance of the proposed algorithms for the three groups of instances and different optimisation software.

			Flight			Error			Step			Time		
Group	Est.	#Oct.	i-STO	IPOPT	Oct.	i-STO	IPOPT	Oct.	i-STO	IPOPT	Oct.	i-STO	IPOPT	Oct.
GRTOP-S2	44.040	10	**45.240**	63.105	63.190	16.656	16.450	**16.411**	**0.231**	0.322	0.322	**0.174**	0.822	44.151
GRTOP-S3	57.181	10	**58.881**	83.212	83.529	20.666	17.125	**16.857**	**0.240**	0.340	0.341	**0.242**	0.913	63.273
GRTOP-S4	77.170	10	**79.270**	102.893	103.097	19.158	14.466	**13.959**	**0.270**	0.350	0.351	**0.284**	1.093	71.738
GRTOP-S5	83.510	10	**83.510**	108.021	109.101	25.617	19.711	**19.362**	**0.243**	0.315	0.318	**0.301**	1.306	90.512
GRTOP-S6	100.341	9	**101.141**	121.004	122.322	25.707	15.220	**13.418**	**0.258**	0.309	0.312	**0.375**	1.422	82.850
GRTOP-S7	111.444	7	**112.244**	148.935	153.056	28.716	22.184	**20.710**	**0.255**	0.338	0.347	**0.409**	1.586	105.254
GRTOP-S8	120.249	10	**122.149**	153.384	152.459	29.152	24.335	**23.074**	**0.249**	0.313	0.311	**0.466**	1.748	119.921
GRTOP-S9	128.263	9	**129.763**	143.487	137.571	22.638	19.297	**19.082**	**0.241**	0.266	0.255	**0.530**	1.904	125.208
GRTOP-S10	147.109	7	**148.409**	187.511	198.877	30.167	**26.229**	27.252	**0.252**	0.319	0.338	**0.559**	2.145	142.114
Group min.	44.040	7	**45.240**	63.105	63.190	16.656	14.466	**13.418**	**0.231**	0.266	0.255	**0.174**	0.822	44.151
Group max.	147.109	10	**148.409**	187.511	198.877	30.167	**26.229**	27.252	**0.270**	0.350	0.351	**0.559**	2.145	142.114
Group avg.	96.405	9	**97.660**	123.833	125.933	24.118	19.610	**19.163**	**0.249**	0.317	0.318	**0.370**	1.446	93.753
Group std.	36.339	1	**36.362**	42.342	45.433	4.996	**4.317**	4.811	**0.013**	0.028	0.033	**0.139**	0.477	34.308
GRTOP-M2	174.382	5	**174.382**	348.120	365.127	128.719	51.794	**43.878**	**0.890**	1.776	1.863	**0.186**	1.030	80.757
GRTOP-M3	241.393	8	**241.393**	348.461	323.724	90.213	**53.524**	54.523	**0.985**	1.422	1.321	**0.250**	1.300	82.253
GRTOP-M4	338.984	3	**338.984**	553.776	544.354	124.556	68.145	**49.482**	**1.153**	1.884	1.852	**0.298**	1.481	104.516
GRTOP-M5	399.925	3	**401.026**	734.563	573.655	183.136	78.113	**66.159**	**1.169**	2.142	1.672	**0.382**	1.821	113.322
GRTOP-M6	435.047	6	**435.047**	746.151	559.348	204.697	74.039	**55.617**	**1.110**	1.903	1.427	**0.424**	1.910	125.741
GRTOP-M7	541.969	3	**541.969**	984.017	920.086	266.971	105.388	**99.052**	**1.229**	2.231	2.086	**0.474**	2.382	171.270
GRTOP-M8	558.258	1	**558.258**	996.227	777.047	341.388	100.467	**63.980**	**1.139**	2.033	1.586	**0.520**	2.764	141.049
GRTOP-M9	670.471	3	**670.471**	1183.688	1036.573	349.692	88.830	**87.026**	**1.244**	2.196	1.923	**0.540**	2.828	140.611
GRTOP-M10	777.656	0	**777.656**	1439.633	-	359.701	**102.628**	-	**1.323**	2.448	-	**0.608**	3.250	134.262
Group min.	174.382	0	**174.382**	348.120	323.724	90.213	51.794	**43.878**	**0.890**	1.422	1.321	**0.186**	1.030	80.757
Group max.	777.656	8	**777.656**	1439.633	1036.573	359.701	105.388	**99.052**	**1.323**	2.448	2.086	**0.608**	3.250	171.270
Group avg.	462.738	3	**462.838**	829.308	646.021	227.181	80.010	**66.265**	**1.132**	1.992	1.714	**0.407**	2.095	122.346
Group std.	211.536	2	**211.506**	394.102	265.910	106.579	20.824	**20.302**	**0.146**	0.338	0.278	**0.151**	0.805	31.752
GRTOP-L10	1541.169	0	**1541.369**	2754.741	-	470.405	**279.582**	-	**2.621**	4.685	-	**0.588**	3.981	107.912

Continued on next page

Table 3: Computational performance of the proposed algorithms for the three groups of instances and different optimisation software.

			Flight			Error			Step			Time		
Group	Est.	#Oct.	i-STO	IPOPT	Oct.	i-STO	IPOPT	Oct.	i-STO	IPOPT	Oct.	i-STO	IPOPT	Oct.
GRTOP-L20	2759.521	0	**2759.521**	4970.499	-	894.977	**512.487**	-	**2.560**	4.611	-	**1.118**	6.839	135.708
GRTOP-L30	4207.686	0	**4207.686**	7212.597	-	1194.836	**655.380**	-	**2.683**	4.600	-	**1.644**	9.778	89.105
GRTOP-L40	5351.336	0	**5351.336**	9428.211	-	1677.321	**849.139**	-	**2.600**	4.581	-	**2.188**	12.810	99.897
GRTOP-L50	6734.541	0	**6735.007**	11518.271	-	1851.402	**1096.827**	-	**2.643**	4.521	-	**2.713**	15.465	105.192
Group min.	1541.169	0	**1541.369**	2754.741	-	470.405	**279.582**	-	**2.560**	4.521	-	**0.588**	3.981	89.105
Group max.	6734.541	0	**6735.007**	11518.271	-	1851.402	**1096.827**	-	**2.683**	4.685	-	**2.713**	15.465	135.708
Group avg.	4124.280	0	**4124.471**	7165.333	-	1201.535	**681.404**	-	**2.622**	4.600	-	**1.650**	9.760	108.946
Group std.	2082.136	0	**2082.232**	3520.206	-	564.947	**321.918**	-	**0.048**	0.063	-	**0.852**	4.625	18.193

From the results in Table 3 one can observe that i-STO is very effective regarding flight time minimisation. This is due to its iterative strategy of increasing the flight time from a possibly infeasible value (the estimated minimum $\bar{\tau}^f$) up to when the first feasible trajectory is found. On average, i-STO finds trajectories that are 21.14%, 24.98% and 42.44%, (for small, medium and large range instances, respectively) shorter in duration than the shortest trajectories found by STO-NLP (running either `IPOPT` or `Octeract-engine` solvers). Nonetheless, this strategy reflects negatively on the error values found by i-STO. Table 3 shows that STO-NLP (considering both NLP solvers) performs better than i-STO regarding error values for all groups of instances. On average, considering the three classes of instances, STO-NLP (considering the best run between `IPOPT` and `Octeract-engine`) finds errors that are 20.54%, 70.83% and 43.29% smaller than i-STO, respectively.

The reason behind this behaviour can be explained by the fact that i-STO relegates the minimisation of the error term to a "secondary" objective for a given (fixed) flight duration (5.3). The i-STO's error values could be improved by adopting a less greedy flight minimisation strategy. However, preliminary experiments showed that adopting a different minimisation algorithm (e.g., the bisection method) would significantly increase computing times. Since i-STO is called many times during the execution of the routing algorithm, we decided by design to go for the less computationally expensive strategy, i.e., the greedy one. As opposed to i-STO, STO-NLP tackles the flight time and error minimisation in the same objective (Section 5.2). That means it can better explore the trade-off between these two terms during the trajectory computation.

Concerning integration step sizes (subgroup **Step**), i-STO shows a better refinement, in terms of length (duration), than STO-NLP. Recall that the step size reflects how often the state of each glider is verified and how fine the control over each glider's flight is. For each class of instances (small, medium and large), i-STO is on average providing control over each glider every 0.25, 1.13 and 2.62 seconds, respectively. Note also that the small, medium and large classes of instances are defined over areas that are $1km^2$, $25km^2$ and $100km^2$ in size, respectively. Therefore, i-STO provides a good level of refinement for control points and reference trajectories given the size of the airspace considered in each class of instance.

Lastly, columns in subgroup **Time** show that i-STO is much faster regarding computing times than STO-NLP. On average, the former is 74.41%, 80.57% and 83.09% faster than the fastest STO-NLP version (running `IPOPT`). This is due to two main reasons. First, i-STO's subproblems consist of SOCPs problems (convex by definition) that are, in the current state-of-the-art, solved much more efficiently than the STO-NLP's nonconvex NLP subproblems. Second, the greedy flight time minimisation approach adopted in i-STO allows for fast computation of feasible trajectories for each arc of a route within STO's general decomposition framework. One can also notice that, in general, computing times of the global optimisation solver `Octeract-engine` are prohibitive within the proposed algorithm. Therefore, from now on all of our experiments involving STO-NLP will employ `IPOPT` as an NLP subproblem solver.

7.3. Discretisation step size

In this section, we study the effects of varying the discretisation step size of the proposed algorithms. These tests were executed as follows. For each instance, a random route was generated and fed to i-STO and STO-NLP for different values of N, namely, 20, 40, 80 and 100. As indicated in Section 7.2, the software `IPOPT` was used for solving STO-NLP's subproblems. All other parameters are set as in the experiments in the previous section.

Tables 4 and 5 show the results of our tests for i-STO and STO-NLP, respectively. Columns have been grouped according to their respective discretisation step size. The meaning of each column remains the same as in previous tables. We have omitted flight time results from Tables 4 and 5 since we could not observe significant variations due to the changing discretisation step sizes. Full results can be found in this paper's online supplement.

Table 4: Computational performance of i-STO for different discretisation sizes.

	N=20			N=40			N=80			N=100		
Group	Error	Step	Time	Error	Step	Time	Error	Step	Time	Error	Step	Time
GRTOP-S min.	42.695	0.631	0.070	23.287	0.308	0.150	11.902	0.153	0.348	9.643	0.123	0.461
GRTOP-S max.	83.057	0.714	0.244	38.182	0.348	0.502	20.713	0.172	1.248	15.859	0.137	1.622
GRTOP-S avg.	65.539	0.662	0.156	30.971	0.324	0.319	15.461	0.161	0.793	11.838	0.129	1.039
GRTOP-S std.	14.865	0.029	0.064	5.376	0.014	0.128	2.948	0.007	0.335	2.152	0.005	0.423
GRTOP-M min.	158.614	2.428	0.067	88.173	1.183	0.139	52.313	0.584	0.295	46.840	0.466	0.379
GRTOP-M max.	666.851	3.471	0.261	427.923	1.691	0.448	207.399	0.835	0.995	155.978	0.666	1.350
GRTOP-M avg.	438.431	3.013	0.164	277.485	1.468	0.308	129.447	0.725	0.661	97.468	0.578	0.900
GRTOP-M std.	184.935	0.354	0.069	121.249	0.173	0.115	54.017	0.085	0.259	39.061	0.068	0.364
GRTOP-L min.	799.644	6.332	0.232	497.918	3.085	0.469	271.429	1.524	0.968	214.846	1.216	1.243
GRTOP-L max.	3851.251	6.754	1.051	2391.316	3.290	2.187	1358.063	1.625	4.578	1080.871	1.296	5.811
GRTOP-L avg.	2293.483	6.581	0.639	1435.946	3.206	1.330	815.280	1.583	2.780	647.869	1.263	3.537
GRTOP-L std.	1202.390	0.166	0.329	744.778	0.081	0.689	427.711	0.040	1.447	341.901	0.032	1.833

Table 5: Computational performance of STO-NLP for different discretisation sizes.

	N=20			N=40			N=80			N=100		
Group	Error	Step	Time	Error	Step	Time	Error	Step	Time	Error	Step	Time
GRTOP-S min.	28.024	0.704	0.344	14.974	0.343	0.661	10.129	0.170	1.427	8.806	0.135	1.869
GRTOP-S max.	50.319	0.968	1.013	32.301	0.461	1.750	20.745	0.225	3.886	16.575	0.180	5.033
GRTOP-S avg.	40.787	0.858	0.674	24.104	0.413	1.194	14.952	0.203	2.689	12.075	0.162	3.504
GRTOP-S std.	8.171	0.091	0.237	6.147	0.041	0.396	3.774	0.019	0.919	2.780	0.016	1.140
GRTOP-M min.	100.955	3.479	0.462	60.020	1.678	0.841	39.132	0.835	1.689	33.542	0.668	2.395
GRTOP-M max.	228.210	6.144	1.486	128.657	3.023	2.673	80.763	1.507	5.077	70.126	1.204	7.169
GRTOP-M avg.	168.665	5.064	0.933	96.608	2.494	1.698	59.710	1.241	3.273	52.918	0.991	4.748
GRTOP-M std.	50.828	0.845	0.375	27.647	0.428	0.670	16.157	0.214	1.219	14.101	0.171	1.740
GRTOP-L min.	567.325	10.555	1.698	336.740	5.233	3.091	194.656	2.629	6.205	168.756	2.104	7.605
GRTOP-L max.	2000.871	11.790	6.926	1320.929	5.923	12.601	747.056	2.985	24.678	671.836	2.388	31.112
GRTOP-L avg.	1253.402	11.154	4.304	800.712	5.560	7.808	456.521	2.798	15.412	407.554	2.239	19.277
GRTOP-L std.	566.218	0.478	2.090	388.978	0.265	3.813	218.948	0.136	7.420	198.454	0.108	9.392

Our results show that both algorithms scale well regarding increasing discretisation sizes. However, a few differences can be observed when comparing STO-NLP and i-STO. We noticed that computing times for STO-NLP increase more dramatically as N increases. Average computing times for each group of instances (`GRTOP-S`, `GRTOP-M` and `GRTOP-L`) increase by 0.88, 0.74 and 2.90 seconds for i-STO when N goes from 20 to 100, while for STO-NLP these differences are 2.83, 3.82 and 14.97 seconds, respectively. Similar conclusions can be drawn about step sizes. Errors are expected to decrease as N increases (Zhao, 2004). Such observations can also be made about our results. As opposed to computing times and step sizes, i-STO seems to be more sensitive to variations in N with respect to these attributes than STO-NLP. While errors decrease by 53.70, 340.96 and 1645.61 on average (per group of instances) for i-STO, theses differences are 28.71, 115.75, and 845.85, for STO-NLP, respectively.

Figure 6 summarises our findings and gives a few more insights about our experiments. This figure is arranged in the following way. Each row shows plots for average errors, step sizes and computing times in the vertical axis for each group of instances. Horizontal axes represent the value of N. The first two columns (Error and Step size) show that, on average, as N approaches 100 the solution of i-STO's subproblems approximates the solution of the NLP subproblems generated by STO-NLP. At the same time, i-STO scales much better concerning computing times than STO-NLP. Accordingly, for smaller discretisation step sizes, i-STO is capable of efficiently finding good quality solutions with competitive computing times.

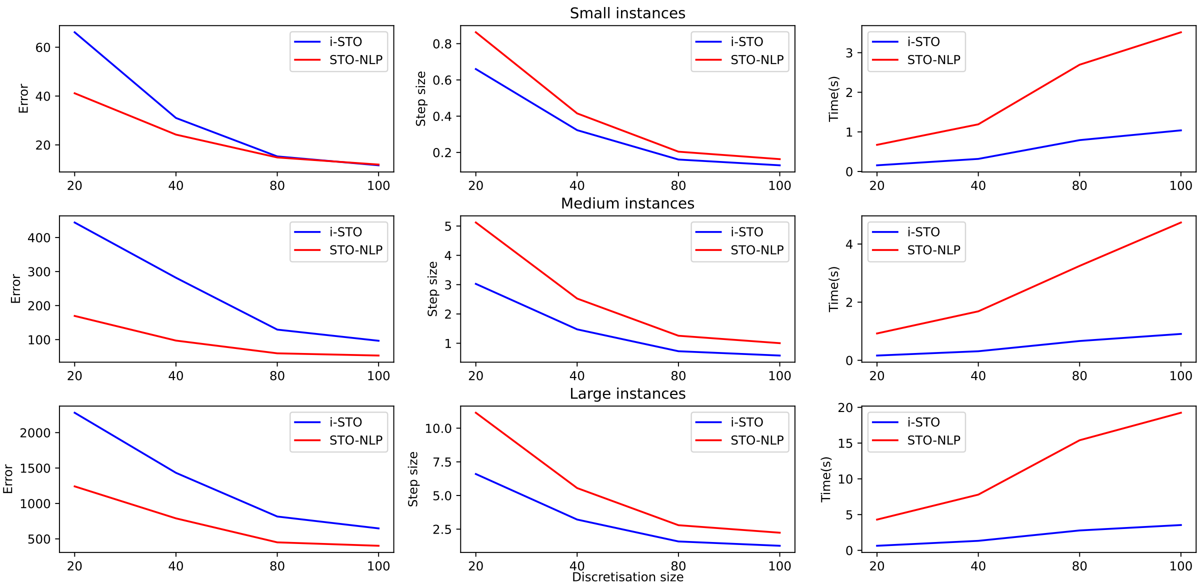


Figure 6: Average error, step size and computing times against the discretisation size for the three groups of instances.

7.4. Influence of the error bounding constraints on solution quality and algorithmic performance

In this section, we test the effectiveness of the error bounding constraints (41) and the error minimisation term in the objective function (40). With this purpose in mind, the following experiment was designed. We have considered three scenarios: (i) the first one (*Scn1*) consists of the current model; (ii) in the second one (*Scn2*), we relax the error bounding constraints (41) while keeping the rest of the model unchanged; (iii) in the third scenario (*Scn3*), we keep the error bounding constraints, but remove the error minimisation term from the objective function (40). For each instance, both i-STO and STO-NLP algorithms were initialised with the same random route across all scenarios. The remaining settings follow from the experiments on Section 7.2.

Tables 6 and 7 show the results of our experiments. Columns in these tables have been grouped according to each scenario defined above. The meaning of each column remains the same as in the previous sections.

From the results in Tables 6 and 7 it is clear that the error bounding constraints and the minimisation of the error improve the solutions found by both i-STO and STO-NLP algorithms in the following sense. Let us define, for a given algorithm and group of instances, the average error gap between two different scenarios i and j as $gap_{ij} = 100 \times (\max\{Error_i, Error_j\} - \min\{Error_i, Error_j\}) / \max\{Error_i, Error_j\}$. Thus, considering i-STO's results and the three groups of instances (`GRTOP-S`, `GRTOP-M` and `GRTOP-L`), one can notice that the gaps between scenarios *Scn1* and *Scn2* are 75.46%, 28.67% and 8.40%, respectively. While the gaps between scenarios *Scn1* and *Scn3* are 74.2%, 87.08% and 95.93%, respectively. Similar figures can be observed regarding STO-NLP. The gaps between *Scn1* and *Scn2* are 21.23%, 0.72% and

0.21%, and the gaps between *Scn1* and *Scn3* are 34.11%, 91.98% and 96.36%, respectively for the three groups of instances. Therefore, keeping both the error bounding constraints (41) and the second term of the objective function (40) in the TO subproblems helps to keep errors small. In addition, *Scn2* provides for smaller errors than *Scn3*, which suggests that the error minimisation term in the objective function has a bigger influence on keeping errors small.

Regarding flight times, the proposed algorithms behave slightly differently regarding the three generated scenarios. For i-STO, no significant changes in flight times could be detected for scenarios *Scn1*, *Scn2* and *Scn3*, considering all three classes of instances. We believe this is due to the prioritisation of the flight time minimisation during the iterative solution of its TO subproblems. The same can be said about STO-NLP and scenarios *Scn1* and *Scn2*.

However, some interesting observations can be made regarding the flight times found by STO-NLP under *Scn3*. Let us recall that in *Scn3* the error term is removed from the objective function, meaning that only the flight time is minimised subject to the error bounding and other constraints. In *Scn3*, STO-NLP is capable of finding average, minimum and maximum flight times that are very similar to the values found by i-STO in all three scenarios. Such results can be interpreted as a confirmation of the effectiveness of i-STO's iterative flight minimisation strategy. By comparing STO-NLP and i-STO under *Scn3*, one notices that STO-NLP finds error values that are, on average, 37.39% smaller than the errors found by i-STO. However, STO-NLP's computing times are on average 94.21% higher than its iterative counterpart.

Finally, a trade-off can be acknowledged between the performance of both TO algorithms flight times, errors and computing times. Keeping in mind that in an integrated routing and TO framework the TO code must be called multiple times, we opted for employing only i-STO in the next set of experiments.

Table 6: Influence of the error bounding constraints and error minimisation on solution quality and the computational performance of the i-STO algorithm.

	Scn1				Scn2				Scn3			
Group	Flight	Error	Step	Time	Flight	Error	Step	Time	Flight	Error	Step	Time
GRTOP-S min.	45.240	16.656	0.231	0.235	44.040	29.752	0.225	0.176	45.140	49.054	0.230	0.078
GRTOP-S max.	149.207	27.508	0.271	0.635	148.107	164.174	0.271	0.502	149.406	123.446	0.271	0.176
GRTOP-S avg.	98.643	22.075	0.252	0.434	97.725	89.952	0.250	0.335	98.534	87.309	0.252	0.126
GRTOP-S std.	36.329	3.794	0.015	0.147	36.291	44.850	0.016	0.122	36.417	25.274	0.015	0.033
GRTOP-M min.	172.564	108.810	0.880	0.219	172.264	139.560	0.879	0.193	172.464	601.691	0.880	0.046
GRTOP-M max.	749.223	361.788	1.277	0.583	749.223	469.813	1.277	0.593	749.223	2834.994	1.277	0.176
GRTOP-M avg.	457.720	216.435	1.125	0.388	457.629	303.436	1.124	0.377	457.665	1675.767	1.124	0.099
GRTOP-M std.	203.788	89.808	0.143	0.139	203.847	126.857	0.144	0.150	203.796	786.044	0.143	0.046
GRTOP-L min.	1527.501	400.465	2.510	0.625	1527.301	443.031	2.510	0.634	1527.301	12039.599	2.510	0.233
GRTOP-L max.	6455.703	1821.976	2.598	2.813	6455.569	2004.684	2.597	2.625	6455.636	44729.947	2.597	1.368
GRTOP-L avg.	3989.213	1157.720	2.556	1.711	3989.098	1263.888	2.555	1.622	3989.117	28422.584	2.555	0.750
GRTOP-L std.	1968.323	570.170	0.036	0.879	1968.358	623.906	0.036	0.803	1968.382	12997.827	0.036	0.446

Table 7: Influence of the error bounding constraints and error minimisation on solution quality and the computational performance of the STO-NLP algorithm.

	Scn1				Scn2				Scn3			
Group	Flight	Error	Step	Time	Flight	Error	Step	Time	Flight	Error	Step	Time
GRTOP-S min.	63.105	14.500	0.284	1.058	62.972	14.730	0.284	0.976	45.260	14.498	0.231	0.747
GRTOP-S max.	178.564	27.074	0.364	2.334	178.091	37.566	0.363	2.107	149.603	46.098	0.271	1.946
GRTOP-S avg.	126.163	20.048	0.328	1.702	125.844	25.450	0.327	1.504	98.667	30.425	0.252	1.352
GRTOP-S std.	42.330	4.361	0.029	0.467	42.113	7.512	0.029	0.414	36.474	10.977	0.015	0.417
GRTOP-M min.	333.147	52.686	1.579	1.231	332.797	54.751	1.578	1.149	172.488	305.889	0.880	0.915
GRTOP-M max.	1298.328	104.255	2.340	3.086	1298.329	104.255	2.340	2.914	749.223	1750.475	1.277	3.348
GRTOP-M avg.	791.746	80.516	1.963	2.049	791.611	81.103	1.962	1.931	457.669	1004.125	1.124	1.945
GRTOP-M std.	345.603	20.388	0.272	0.683	345.681	19.620	0.273	0.638	203.790	511.862	0.143	0.871

Continued on next page

Table 7: Influence of the error bounding constraints and error minimisation on solution quality and the computational performance of the STO-NLP algorithm.

	Scn1				Scn2				Scn3			
Group	Flight	Error	Step	Time	Flight	Error	Step	Time	Flight	Error	Step	Time
GRTOP-L min.	2670.641	260.145	4.271	3.927	2671.596	260.220	4.271	3.864	1527.301	6794.300	2.510	3.984
GRTOP-L max.	11094.943	1023.943	4.565	16.445	11094.067	1024.559	4.564	14.762	6455.503	28762.080	2.597	24.574
GRTOP-L avg.	6857.822	646.933	4.411	10.168	6867.216	648.294	4.418	9.237	3989.079	17794.343	2.555	12.956
GRTOP-L std.	3347.965	298.297	0.129	5.031	3347.117	298.622	0.125	4.364	1968.335	8723.192	0.036	8.155

7.5. Routing a fleet of gliders

In this section, we analyse the performance of the proposed i-STO algorithms when it is used together with our vehicle routing metaheuristic. The overall efficiency of the algorithm, in terms of the size of instances solved, is explored first. We then assess if the set partitioning formulation refinement stage has a beneficial effect on the solution approach. Next, we provide some insights on the impact of the fleet size on the time required to collect aerial information, and how fast information on ground targets will be available with variable fleet size and varying number of points of interest. Finally, based on real-world scenarios, we perform computational experiments to illustrate the application of gliders in disaster assessment.

We ran ILS-STO for each problem instance; the average results obtained per group are reported in Table 8. In this set of experiments, the discretisation size was set to $N = 50$ points and a time limit of 60s was imposed for the solution of TO subproblems. Recall that the maximum fleet size was set to $n_g = \lfloor n/2 \rfloor$, where n is the number of waypoints in a given instance. In this table, column **Group** keeps the same meaning as in the previous tables. **Util. (%)** represents the fleet utilisation, which is computed as the number of gliders used in the solution divided by the maximum number of gliders available in the fleet. Column **Obj.** presents the objective function value, while **Mks** and **Error** represent the objective's makespan and error terms, respectively. We also provide a **Max.err** column, showing the error in the route in which the error component is the highest. **Flight** provides the sum of the flight times of all the airborne gliders. **Iter** is the number of LS iterations. Finally, column **Time(s)** is the average CPU time in seconds spent by ILS-STO, whereas **TO (%)** is the percentage of CPU time spent by the TO algorithm.

Table 8: Average aggregate computational results of the ILS-i-STO heuristic for the three groups of instances.

Group	Util.(%)	Obj.	Mks	Error	Max.err	Flight	Iter	Time(s)	TO(%)
GRTOP-S2	100.000	64.913	46.725	18.188	18.188	46.725	30.000	5.506	99.891
GRTOP-S3	100.000	77.228	57.872	19.356	19.356	57.872	30.000	8.095	99.871
GRTOP-S4	100.000	72.796	48.935	23.861	16.458	90.457	33.400	13.785	99.891
GRTOP-S5	100.000	79.217	52.463	26.753	17.821	97.525	33.400	14.898	99.857
GRTOP-S6	83.000	77.690	50.127	27.563	15.278	111.213	42.300	23.226	99.848
GRTOP-S7	83.000	80.838	52.269	28.569	15.979	117.549	42.500	26.527	99.838
GRTOP-S8	80.000	87.330	48.187	39.143	18.240	140.989	51.500	40.083	99.846
GRTOP-S9	72.500	75.265	49.035	26.230	12.822	130.973	44.200	34.957	99.820
GRTOP-S10	66.000	94.037	51.307	42.730	19.345	157.484	51.800	50.613	99.830
Group min.	66.000	64.913	46.725	18.188	12.822	46.725	30.000	5.506	99.820
Group max.	100.000	94.037	57.872	42.730	19.356	157.484	51.800	50.613	99.891
Group avg.	86.409	78.933	51.047	28.483	16.878	105.000	40.082	24.892	99.855
Group std.	13.532	9.435	3.713	8.765	2.321	39.261	8.671	16.286	0.027
GRTOP-M2	100.000	259.755	166.709	93.046	93.046	166.709	30.000	5.843	99.893
GRTOP-M3	100.000	274.704	198.594	76.109	76.109	198.594	30.000	7.607	99.861
GRTOP-M4	100.000	341.574	188.452	153.123	120.794	351.325	32.300	12.203	99.861
GRTOP-M5	100.000	388.995	214.225	174.770	129.024	402.024	33.200	14.291	99.849
GRTOP-M6	83.000	406.498	195.787	210.712	143.945	424.693	36.400	19.522	99.833
GRTOP-M7	79.600	384.393	207.787	176.606	118.812	442.352	43.300	26.594	99.826
GRTOP-M8	75.000	479.049	213.018	266.031	137.422	573.584	45.100	32.814	99.827
GRTOP-M9	85.000	454.998	216.592	238.406	147.697	650.236	48.500	38.520	99.811

Continued on next page

Table 8: Average aggregate computational results of the ILS-i-STO heuristic for the three groups of instances.

Group	Util.(%)	Obj.	Mks	Error	Max.err	Flight	Iter	Time(s)	TO(%)
GRTOP-M10	76.000	455.902	218.465	237.437	115.454	713.730	53.300	49.715	99.812
Group min.	75.000	259.755	166.709	76.109	76.109	166.709	30.000	5.843	99.811
Group max.	100.000	479.049	218.465	266.031	147.697	713.730	53.300	49.715	99.893
Group avg.	88.509	380.425	200.437	178.944	118.737	436.699	39.582	23.879	99.843
Group std.	10.887	81.625	18.482	68.988	25.365	197.479	8.942	15.952	0.029
GRTOP-L10	77.333	689.669	417.535	272.133	134.627	1373.611	49.333	42.626	99.821
GRTOP-L20	66.667	919.388	415.534	503.854	164.173	2264.470	53.667	93.009	99.774
GRTOP-L30	63.667	1148.661	406.837	741.824	199.116	3172.100	52.800	137.937	99.789
GRTOP-L40	70.000	1477.538	433.503	1044.035	198.355	4647.002	52.467	180.800	99.779
GRTOP-L50	72.533	1599.269	407.710	1191.560	222.127	5489.059	52.800	234.308	99.772
Group min.	63.667	689.669	406.837	272.133	134.627	1373.611	49.333	42.626	99.772
Group max.	77.333	1599.269	433.503	1191.560	222.127	5489.059	53.667	234.308	99.821
Group avg.	70.171	1160.495	417.351	745.300	182.164	3401.273	52.010	137.945	99.790
Group std.	5.413	375.536	10.949	376.757	35.024	1683.415	1.743	76.157	0.020

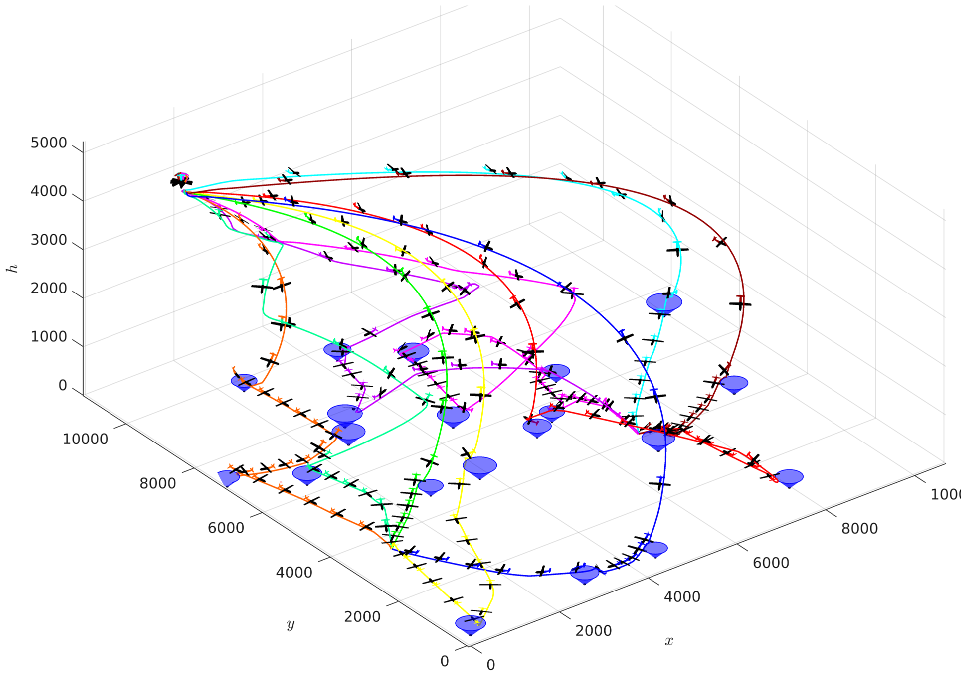
The overall average utilisation of the fleet is 81.70% on average across all instances and varies from 70.17% (for the instances with a large area and a large number of gliders) to 86.41% for the smaller instances. This demonstrates the need for a fleet of gliders, and that a high utilisation rate is achieved to minimise the makespan. The algorithm can also be used to identify a suitable number of gliders given the number of waypoints to be surveyed. In particular, on average roughly 10 gliders are used to survey instances with 50 waypoints efficiently.

The makespan represents on average 64.64%, 52.69% and 35.96% of the overall **Obj.**, for the small, medium and large instances, respectively. The overall routing time (i.e., the sum of the flying times of all gliders) compares favourably to the solution makespan, showcasing that all gliders employed in the mission are on average flying routes of comparable length to the longest. By multiplying the average makespan by the average number of gliders used for each group of instances we obtain 5051.38 seconds (approx. 84 minutes), whereas the overall average flight time for all gliders and instance groups is 3942.97 seconds (approx. 65 minutes), representing a gap of roughly only 20%. That shows a good utilisation of all gliders in the fleet and comparable flying durations.

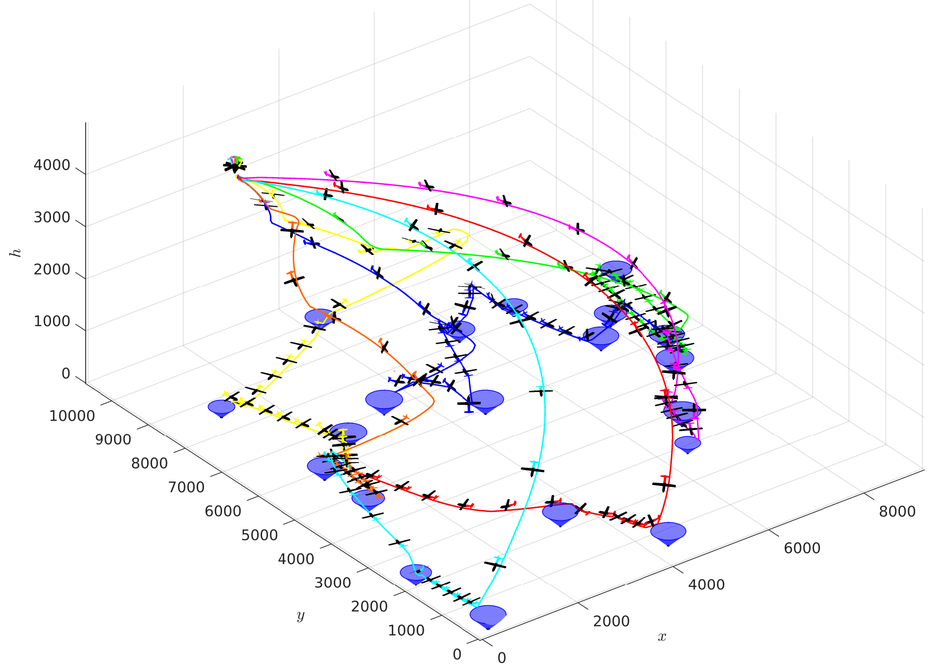
Assessing solution quality is not an easy task, as this combines errors in the position and orientation of each glider and in the control variables at each time-discretisation point. It is though interesting to notice that the average total errors are 28.48, 178.94 and 745.30 for the small, medium and large instances respectively. We recall that the coordinates of the gliders over time and thus those error terms are measured in meters, and the overall flying range (or airspace) for the small, medium, and large instances span 1, 25, and 100 km^2. These errors seem therefore relatively small when compared with the number of time-discretisation points, the number of gliders and the scale of the airspace. The **Max.err** column showcases that the glider routes with the maximum sum of error terms are still within reasonable values.

The computing times of the proposed algorithm are quite small, and even the largest instances with 50 waypoints require on average 234.31 seconds. The small and medium size instances require on average 24.89 and 23.88 seconds, as it seems a larger range does not affect the algorithm computing times. The larger set of instances requires on average 137.95 seconds. This suggests that the new TO optimisation algorithm combined with a fast and effective metaheuristic allows for solving much larger instances. The number of local search iterations varies between 39.58 and 52.01, and tends to increase with the size of the instance.

The most remarkable computing time aspect though is highlighted in column **TO(%)**, where the percentage of time spent in the TO algorithm is given. It can be noticed that this is consistently over 99% of the overall computing time for all instance sizes, proving that the TO problem remains the most challenging and time-consuming component of the algorithm. This is the part of the code which still represents a bottleneck, and further heuristic approximations might be necessary to either solve larger instances or obtain solutions in faster computing times. Figure 7 shows feasible solutions of two different instances.



(a) Solution of `grtopL_203_2`.



(b) Solution of `grtopL_203_5`.

Figure 7: Depiction of the optimal solutions of two large range instances.

Table 9 shows the effect of applying the SP-based approach after the multi-start ILS procedure. The average objective function values before and after solving the SP formulation are reported (namely, **ILS** and **SP**). Column **Gap(%)** shows the average percentage improvement achieved using the SP module. This improvement is calculated as $100 \times [(\text{ILS} - \text{SP})/\text{ILS}]$. Finally, columns **Time(s)** and **SP(s)** report the average computing times and the time spent by the SP module, respectively, in seconds.

Overall, the SP-based approach improves the objective function value on average by 2.17%, 2.45% and

18.78% on the small, medium and large sets, respectively. We highlight that the time required to solve the SP formulation (never higher than fractions of a second) is negligible when compared against the overall running times of ILS-STO. These results suggest that the SP module is useful in our framework, as it helps the method to improve the average objective function value with little added computational effort, especially on larger instances.

Table 9: Effectiveness of the SP postprocessing for the three groups of instances (average aggregate results).

Group	ILS	SP	Gap(%)	Time(s)	SP(s)
GRTOP-S2	64.913	64.913	0.000	5.506	0.001
GRTOP-S3	77.228	77.228	0.000	8.095	0.002
GRTOP-S4	72.796	72.796	0.000	13.785	0.002
GRTOP-S5	79.217	79.217	0.000	14.898	0.003
GRTOP-S6	77.690	77.690	0.000	23.226	0.052
GRTOP-S7	80.884	80.838	0.060	26.527	0.113
GRTOP-S8	91.199	87.330	3.413	40.083	0.089
GRTOP-S9	78.805	75.265	4.376	34.957	0.044
GRTOP-S10	101.916	94.037	7.986	50.613	0.131
Group min.	64.913	64.913	0.000	5.506	0.001
Group max.	101.916	94.037	7.986	50.613	0.131
Group avg.	81.043	78.933	2.166	24.892	0.052
Group std.	12.049	9.435	3.115	16.286	0.052
GRTOP-M2	259.755	259.755	0.000	5.843	0.001
GRTOP-M3	274.704	274.704	0.000	7.607	0.002
GRTOP-M4	341.574	341.574	0.000	12.203	0.003
GRTOP-M5	388.995	388.995	0.000	14.291	0.010
GRTOP-M6	406.674	406.498	0.032	19.522	0.004
GRTOP-M7	405.759	384.393	3.405	26.594	0.072
GRTOP-M8	501.734	479.049	3.565	32.814	0.052
GRTOP-M9	473.289	454.998	4.105	38.520	0.122
GRTOP-M10	492.412	455.902	7.911	49.715	0.088
Group min.	259.755	259.755	0.000	5.843	0.001
Group max.	501.734	479.049	7.911	49.715	0.122
Group avg.	391.490	380.425	2.448	23.879	0.043
Group std.	91.683	81.625	3.021	15.952	0.048
GRTOP-L10	737.372	689.669	6.142	42.626	0.037
GRTOP-L20	1103.945	919.388	16.718	93.009	0.158
GRTOP-L30	1456.170	1148.661	20.988	137.937	0.141
GRTOP-L40	2057.323	1477.538	27.662	180.800	0.153
GRTOP-L50	2180.542	1599.269	26.156	234.308	0.102
Group min.	737.372	689.669	6.142	42.626	0.037
Group max.	2180.542	1599.269	27.662	234.308	0.158
Group avg.	1493.324	1160.495	18.781	137.945	0.113
Group std.	604.573	375.536	8.789	76.157	0.051

Finally, we present some results regarding the effects of the fleet size on the makespan. Table 10 presents the average aggregate results obtained on the GRTOP-L groups with 10, 20, 30, 40, and 50 waypoints, by varying the fleet size from only 1 glider to $n_g = \lfloor n/2 \rfloor$. In this table, column **Fleet** shows the maximum number of available gliders (fleet size). The remaining columns keep the same meaning as in the previous tables. In order to keep our presentation short and convenient, for each group of instances we have omitted some of the rows. Complete results can be accessed through this paper's online supplement.

Table 10: Average aggregate results for the three groups of instances with varying maximum fleet sizes.

Group	Fleet	Util.(%)	Obj.	Mks	Error	Max.err	Flight	Iter	Time(s)	TO(%)
	1	100.000	1138.482	782.872	355.610	355.610	782.872	32.933	21.862	99.489
	2	100.000	734.669	519.214	215.454	146.080	998.128	40.000	28.437	99.745
GRTOP-L10	3	90.933	683.981	454.167	229.813	135.740	1135.767	40.867	30.526	99.766

Continued on next page

Table 10: Average aggregate results for the three groups of instances with varying maximum fleet sizes.

Group	Fleet	Util.(%)	Obj.	Mks	Error	Max.err	Flight	Iter	Time(s)	TO(%)
	4	88.333	666.122	436.726	229.396	110.432	1369.235	47.133	37.575	99.808
	5	74.667	671.138	411.200	259.938	136.315	1315.040	47.733	39.701	99.830
GRTOP-L20	4	100.000	883.030	470.362	412.668	169.190	1725.958	53.933	78.586	99.679
	5	94.667	888.979	433.996	454.983	174.613	1859.691	51.333	77.346	99.709
	6	89.800	890.779	424.597	466.181	160.660	2005.099	49.533	75.138	99.729
	7	80.533	901.320	425.823	475.497	162.994	2025.237	50.333	78.830	99.757
	8	76.400	906.420	420.242	486.177	160.010	2151.568	55.733	89.677	99.779
GRTOP-L30	5	100.000	1071.940	454.102	617.837	207.057	2096.440	53.133	113.738	99.633
	6	94.333	1078.550	435.643	642.908	225.865	2234.441	49.133	106.632	99.631
	7	83.400	1078.425	450.357	628.067	181.439	2336.764	53.067	117.160	99.678
	11	82.400	1094.466	406.778	687.687	180.955	3122.643	49.933	118.587	99.754
	12	73.533	1096.742	411.090	685.651	165.351	3044.727	56.467	134.185	99.766
GRTOP-L40	5	100.000	1296.447	494.401	802.045	310.651	2278.822	48.200	135.378	99.472
	6	90.933	1243.192	485.050	758.142	254.481	2393.350	50.133	142.035	99.535
	10	88.000	1353.788	431.569	922.221	233.566	3271.919	52.600	158.638	99.711
	15	75.800	1433.193	420.232	1012.961	248.507	4027.680	44.333	139.329	99.746
	20	70.000	1575.472	428.647	1146.824	207.722	4664.651	55.667	185.048	99.794
GRTOP-L50	5	100.000	1261.953	514.298	747.655	230.758	2376.447	48.867	169.451	99.322
	10	92.667	1276.980	402.422	874.559	205.865	3272.497	60.333	223.269	99.653
	15	74.800	1359.261	410.144	949.116	211.921	3786.545	53.933	206.559	99.708
	20	67.333	1450.429	420.988	1029.440	210.947	4392.169	49.933	197.949	99.722
	25	68.267	1484.163	423.477	1060.686	186.266	5339.435	57.400	238.754	99.744

One can observe that the utilisation tends to decrease as the fleet size increases, especially for smaller instances. In addition, the makespan also decreases as there are more gliders available for visiting waypoints. However, due to airspace defined for this group of instances, the makespan tends to converge to roughly 400 seconds as more gliders are allowed. The sum of errors tends to increase as more gliders are added to the solution. This is expected since the addition of more gliders means that more EOMs need to be solved within our framework. CPU times tend to be smaller for instances with fewer gliders, however, the increase in computing time is typically smaller than the benefit obtained by reducing the makespan up to its convergence value.

Figure 8 illustrates how much the average makespan improves, per instance group, as one glider is added to the fleet. From Figure 8 we can verify that the average improvements are substantial when the initial few gliders are added, but not that prominent after the fleet size reaches a certain number. It is also possible to see that such improvements tend to increase with the size of the instance. We highlight that this type of analysis may potentially help practitioners determine a fleet size appropriate to the application.

The numerical results in this section indicate that our approach provides consistent solutions for problems with up to 25 gliders and up to 50 waypoints. Our routing algorithm provides solutions with good utilisation of the given glider fleet under reasonable computing times. Numerical errors bestowed on the solutions due to the discretisation of time-dependent dynamics scale well with problem sizes.

8. Concluding remarks

In this paper, we presented heuristic approaches for efficiently solving the GRTOP. In the GRTOP, flight dynamics takes a major role in route and trajectory planning. We modelled the problem as a multi-phase Optimal Control (OC) problem, in which the flight dynamics are allowed to change within a given route. Different flight modes were considered to compute steady-flight conditions, which in turn were used for linearising the gliders' EOMs. Next, by applying a numerical integration method, namely, direct collocation, we transformed the infinite-dimensional OC problem into an MINLP with a bounded error approximation. In this formulation, each phase of the TO is associated with an arc in a directed graph.

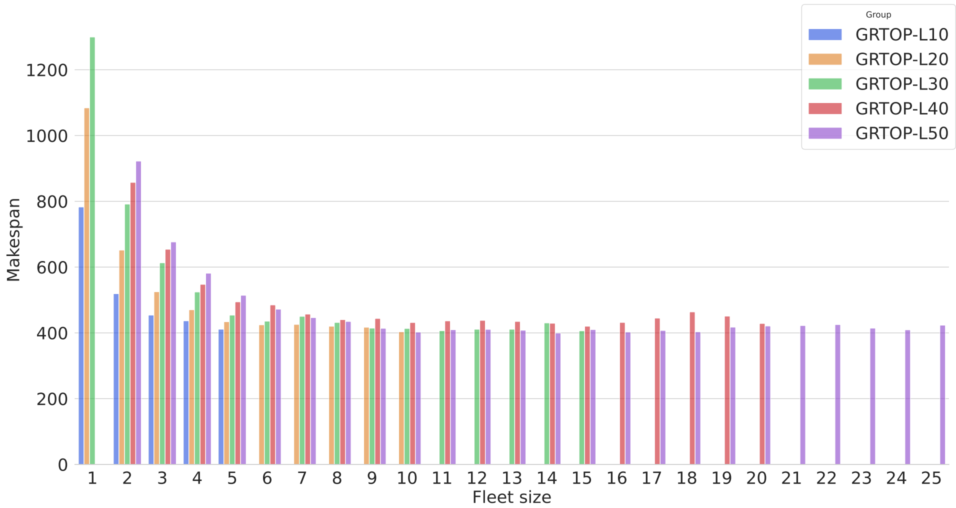


Figure 8: Average makespan improvement as one glider is added to the fleet.

The so-called ILS-STO framework takes advantage of the model's structure and is composed of two main building blocks: (i) two STO algorithms to find feasible trajectories for a fixed sequence of waypoints; and (ii) a routing ILS-based matheuristic, which combines iterated local search and a set-partitioning formulation, for finding sequences of waypoints that can be evaluated by STO.

Our STO approach decomposes the multi-phase TO problem into a sequence of single-phase subproblems, which are solved either by an NLP solver (STO-NLP) or by an iterative method (i-STO). Next, each subproblem is solved according to the sequence defined by the provided route. A feasible trajectory is then constructed by patching the solutions of each subproblem together in the provided order.

We performed several numerical experiments on instances from the literature and 75 new randomly generated instances to understand the behaviour of the different TO algorithms. Our results showed that STO-NLP and i-STO performed very differently in terms of solution values and accuracy. In summary, we observed that STO-NLP is capable of finding solutions with smaller error values at the expense of higher computing times. On the other hand, i-STO finds solutions with a lower makespan and higher error error, but in shorter computing times. We also showed that both algorithms scale well with increasing discretisation sizes.

Further computational experiments showed that ILS-STO is capable of finding feasible GRTOP solutions in short computing times. The time required to find a locally optimal solution was never worse than 235 seconds. Our experiments also showed that the SP-based formulation often helps with improving the objective function value. In addition, we studied the effects of varying the maximum allowed number of gliders in the fleet on the obtained solutions. Results showed that a fleet containing between 7 and 10 gliders usually produces better solutions in terms of makespan and fleet utilisation for the considered instances.

Future research avenues involving GRTOP or related problems may consider alternative objective functions, such as optimising the fleet size and mission costs. Alternatively, equity objectives (e.g., optimising latency or the latest arrival) could also be taken into account. Moreover, the launching position of the gliders can also be optimised as it might affect surveying operation times. Finally, by employing more accurate gliding dynamics one could achieve more realistic solutions. We expect that such an approach would require more sophisticated TO methods though.

Regarding the ILS-STO algorithm, our next research steps would include embedding STO into the local search phase of ILS. For example, let $r_1 = (0, 1, 2, 3, 4, 5, 6, 7)$ be the current best solution with cost c_1. Let us suppose that the neighbour solution $r_2 = (0, 1, 2, 3, 4, 6, 5, 7)$ needs to be evaluated, i.e., the solution cost c_2 must be computed. Note that these solutions are identical until the fifth position, therefore STO would run only from the sixth position onwards since the cost of the segment $0-1-2-3-4$ would remain the same. The challenge of performing such modification to our method consists of integrating the local search and STO without overly compromising the running times of the algorithm.

Finally, it would be interesting to extend the ILS-STO algorithm to other autonomous systems such as unmanned underwater vehicles and powered UAVs. By replacing the gliders' dynamics with the appropriate EOMs one could easily apply the methodology presented in this article for solving many classes of unmanned vehicle routing and TO problems.

Acknowledgements

The authors would like to thank the RNLI and Dr. Andràs Sóbester from the University of Southampton for suggesting this problem. The first author received grants from CNPq [Grant no. 202241/2041-9]. The work reported in this paper was undertaken as part of the Made Smarter Innovation: Centre for People-Led Digitalisation, at the University of Bath, University of Nottingham, and Loughborough University. The last author received funding from CNPq [Grants no. 406245/2021-5 and 309580/2021-8] and by the Paraíba State Research Foundation (FAPESQ) [Grant no. 041/2023].

References

Agatz, N., Bouman, P., & Schmidt, M. (2018). Optimization approaches for the traveling salesman problem with drone. *Transportation Science*, *52*, 965–981. doi:10.1287/trsc.2017.0791.

Aretoulaki, E., Ponis, S. T., & Plakas, G. (2023). Complementarity, interoperability, and level of integration of humanitarian drones with emerging digital technologies: A state-of-the-art systematic literature review of mathematical models. *Drones*, *7*. URL: `https://www.mdpi.com/2504-446X/7/5/301`. doi:10.3390/drones7050301.

Betts, J. T. (2001). *Practical methods for optimal control using nonlinear programming* volume 1 of *Advances in design and control*. Philadelphia, PA: Society for Industrial and Applied Mathematics. doi:10.1137/1.9780898718577.

Blanchard, H. P. (1967). *ELEMENTARY GLIDING: A Manual approved by the British Gliding Association for Pupil Glider Pilots*. Kimberley House, Vaughan Way, Leicester: British Gliding Association.

Coutinho, W., Fliege, J., & Battarra, M. (2022). Glider routing and trajectory optimisation instances. URL: `https://github.com/waltonpcoutinho/GRTOP_instances`.

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. doi:j.cie.2018.04.037.

Coutinho, W. P., Fliege, J., & Battarra, M. (2016). A conic-programing-based approach for trajectory optimisation of unmanned gliders. In J. Bleach (Ed.), *58th Annual Conference on Operational Research Society, OR 2016* (pp. 183–187). The Operational Research Society.

Coutinho, W. P., Fliege, J., & Battarra, M. (2019). Glider routing and trajectory optimisation in disaster assessment. *European Journal of Operational Research*, *274*, 1138–1154. doi:j.ejor.2018.10.057.

Crispin, C. (2016). *Path Planning Algorithms for Atmospheric Science Applications of Autonomous Aircraft Systems*. Ph.D. Thesis University of Southampton Southampton, UK.

Fisch, F. (2011). *Development of a Framework for the Solution of High-Fidelity Trajectory Optimization Problems and Bilevel Optimal Control Problems*. Ph.D. Thesis Technical University of Munich Munich, Germany.

How, J. P., Frazzoli, E., & Chowdhary, G. V. (2015). Linear Flight Control Techniques for Unmanned Aerial Vehicles. In K. P. Valavanis, & G. J. Vachtsevanos (Eds.), *Handbook of Unmanned Aerial Vehicles* (pp. 529–576). Springer Netherlands. doi:10.1007/978-90-481-9707-1˙49.

Kanistras, K., Martins, G., Rutherford, M., & Valavanis, K. (2015). Survey of unmanned aerial vehicles (uavs) for traffic monitoring. In *Handbook of Unmanned Aerial Vehicles* (pp. 2643–2666). Dordrecht: Springer Netherlands. doi:10.1007/978-90-481-9707-1˙122.

Keane, A., Scanlan, J., Lock, A., Ferraro, M., Spillane, P., & Breen, J. (2017). Maritime flight trials of the Southampton university laser sintered aircraft: Project abatross. *The Aeronautical Journal of the Royal Aeronautical Society*, *121*, 1502–1529. URL: https://eprints.soton.ac.uk/411713/. doi:10.1017/aer.2017.71.

Keviczky, T., Borrelli, F., Fregene, K., Godbole, D., & Balas, G. (2008). Decentralized Receding Horizon Control and Coordination of Autonomous Vehicle Formations. *IEEE Transactions on Control Systems Technology*, *16*, 19–33.

Khoufi, I., Laouiti, A., & Adjih, C. (2019). A survey of recent extended variants of the traveling salesman and vehicle routing problems for unmanned aerial vehicles. *Drones*, *3*. URL: https://www.mdpi.com/2504-446X/3/3/66. doi:10.3390/drones3030066.

Lourenço, H. R., Martin, O. C., & Stützle, T. (2010). Iterated local search: Framework and applications. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (pp. 363–397). Springer US volume 146 of *International Series in Operations Research & Management Science*. doi:10.1007/978-1-4419-1665-5˙12.

Mersheeva, V. (2015). *UAV Routing Problem for Area Monitoring in a Disaster Situation*. Ph.D. Thesis Alpen-Adria-Universität Klagenfurt Austria.

Mladenovic, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, *24*, 1097–1100. doi:10.1016/S0305-0548(97)00031-2.

Morandi, N., Leus, R., Matuschke, J., & Yaman, H. (2023). The traveling salesman problem with drones: The benefits of retraversing the arcs. *Transportation Science*, (pp. 1–19). doi:10.1287/trsc.2022.0230. Article in Advance.

Mustapa, Z., & Saat, S. (2016). Autonomous attitude control of a quadcopter unmanned aerial vehicle (uav). *Journal of Telecommunication, Electronic and Computer Engineering*, *7*, 153–160.

Nedjati, A., Vizvari, B., & Izbirak, G. (2016). Post-earthquake response by small UAV helicopters. *Natural Hazards*, *80*, 1669–1688. doi:10.1007/s11069-015-2046-6.

Nex, F., & Remondino, F. (2013). UAV for 3d mapping applications: a review. *Applied Geomatics*, *6*, 1–15. doi:10.1007/s12518-013-0120-x.

Öztürk, D. T., & Köksalan, M. (2023). Biobjective route planning of an unmanned air vehicle in continuous space. *Transportation Research Part B: Methodological*, *168*, 151–169. doi:10.1016/j.trb.2023.01.001.

Pintér, J., & Kampas, F. (2013). Benchmarking nonlinear optimization software in technical computing environments. *TOP*, *21*, 133–162. doi:https://doi.org/10.1007/s11750-011-0209-5.

Richards, A., Schouwenaars, T., How, J. P., & Feron, E. (2002). Spacecraft Trajectory Planning with Avoidance Constraints Using Mixed-Integer Linear Programming. *Journal of Guidance, Control, and Dynamics*, *25*, 755–764. doi:10.2514/2.4943.

Russell, J. (1996). *Performance and Stability of Aircraft*. Oxford: Butterworth-Heinemann. doi:10.1016/B978-0-340-63170-6.X5000-2.

Rysdyk, R. (2006). Unmanned Aerial Vehicle Path Following for Target Observation in Wind. *Journal of Guidance, Control, and Dynamics*, *29*, 1092–1100. doi:10.2514/1.19101.

Stengel, R. F. (2004). *Flight Dynamics*. Princeton University Press. doi:10.2307/j.ctt1287kgx.

Subramanian, A. (2012). *Heuristic, Exact and Hybrid Approaches for Vehicle Routing Problems*. Ph.D. Thesis Universidade Federal Fluminense Niterói, Brazil.

Teschl, G. (2012). *Ordinary differential equations and dynamical systems* volume 140. American Mathematical Soc.

Xia, Y., Batta, R., & Nagi, R. (2017). Controlling a fleet of unmanned aerial vehicles to collect uncertain information in a threat environment. *Operations Research*, *65*, 674–692. doi:10.1287/opre.2017.1590.

Yuan, C., Zhang, Y., & Liu, Z. (2015). A survey on technologies for automatic forest fire monitoring, detection, and fighting using unmanned aerial vehicles and remote sensing techniques. *Canadian Journal of Forest Research*, *45*, 783–792. doi:10.1139/cjfr-2014-0347.

Zhao, Y. J. (2004). Optimal patterns of glider dynamic soaring. *Optimal Control Applications and Methods*, *25*, 67–89. doi:10.1002/oca.739.

Zhou, F., Li, X., & Ma, J. (2017). Parsimonious shooting heuristic for trajectory design of connected automated traffic part i: Theoretical analysis with generalized time geography. *Transportation Research Part B: Methodological*, *95*, 394–420. doi:10.1016/j.trb.2016.05.007.