Autonomous Robots (2024) 48:27
https://doi.org/10.1007/s10514-024-10179-z

Optimal policies for autonomous navigation in strong currents using fast marching trees

Bernardo Martinez Rocamora Jr.[1] · Guilherme A. S. Pereira[1]

Received: 2 December 2023 / Accepted: 24 September 2024 / Published online: 22 October 2024

Abstract
Several applications require that unmanned vehicles, such as UAVs and AUVs, navigate environmental flows. While the flow can improve the vehicle's efficiency when directed towards the goal, it may also cause feasibility problems when it is against the desired motion and is too strong to be counteracted by the vehicle. This paper proposes the flow-aware fast marching tree algorithm (FlowFMT*) to solve the optimal motion planning problem in generic three-dimensional flows. Our method creates either an optimal path from start to goal or, with a few modifications, a vector field-based policy that guides the vehicle from anywhere in its workspace to the goal. The basic idea of the proposed method is to replace the original neighborhood set used by FMT* with two sets that consider the reachability from/to each sampled position in the space. The new neighborhood sets are computed considering the flow and the maximum speed of the vehicle. Numerical results that compare our methods with the state-of-the-art optimal control solver illustrate the simplicity and correctness of the method.

Keywords Path planning · Motion policy · Strong currents · Strong winds · Fast marching tree · Holonomic robot

1 Introduction

In diverse applications, autonomous aerial and aquatic vehicles are required to work in environments with strong natural flows. In these applications, the flow can both assist or constrain the vehicle's motion. For instance, NASA's IceNode buoyant robots exploit ocean currents to move along the basal ice-ocean interface of the Antarctic ice shelves, acquiring long-duration melt rate measurements (Rossi et al., 2021). In another example, balloons developed by Loon provide internet to remote places (Nagpal & Samdani, 2017). Rather than flying against the winds, these balloons take advantage of the atmospheric winds to navigate without propulsion.

In an extraterrestrial application, NASA has considered the use of aerial robots to explore the cloud layer of the Venusian atmosphere (VEXAG, 2019), where recent observations have found evidence of phosphine (Greaves et al., 2020), a compound associated with microbial presence. The strong winds of Venus, which may be as fast as 100 ms^{-1} (Rossi et al., 2023), can help the vehicle to circulate the planet in five days but can also prevent some latitudes from being reached sooner than that (Martinez Rocamora Jr et al., 2022). To help with these applications, the methods proposed in this paper and illustrated in Fig. 1 compute optimal trajectories or motion policies for vehicles navigating natural flows.

The motion planning problem under the influence of environmental flows was approached from many different perspectives. Alvarez et al. (2004) developed a method using a genetic algorithm for minimum energy path planning of autonomous underwater vehicles (AUVs) in two- and three-dimensional (2D and 3D) environments for time and space-varying ocean currents. While the method considers complex ocean currents, it does not constrain the vehicle's maximum speed, assuming that the vehicle can always counteract the ocean currents to reach a nominal speed. Garau et al. (2005) applied a grid-based search (A*) method to find minimum energy paths for AUVs in 2D ocean environments. The velocity of the flow was considered to be always smaller than the velocity of the vehicle relative to the current. When the vehicle's speed relative to the fluid is smaller than the speed of the fluid relative to the inertial frame, we say that the vehicle is subjected to strong winds or currents. To han-

✉ Bernardo Martinez Rocamora Jr.
bernardo.rocamora@gmail.com

Guilherme A. S. Pereira
guilherme.pereira@mail.wvu.edu

[1] Department of Mechanical, Materials and Aerospace Engineering, Benjamin M. Statler School of Engineering and Mineral Sciences, West Virginia University, 1306 Evansdale Dr., Morgantown, WV 26506, USA

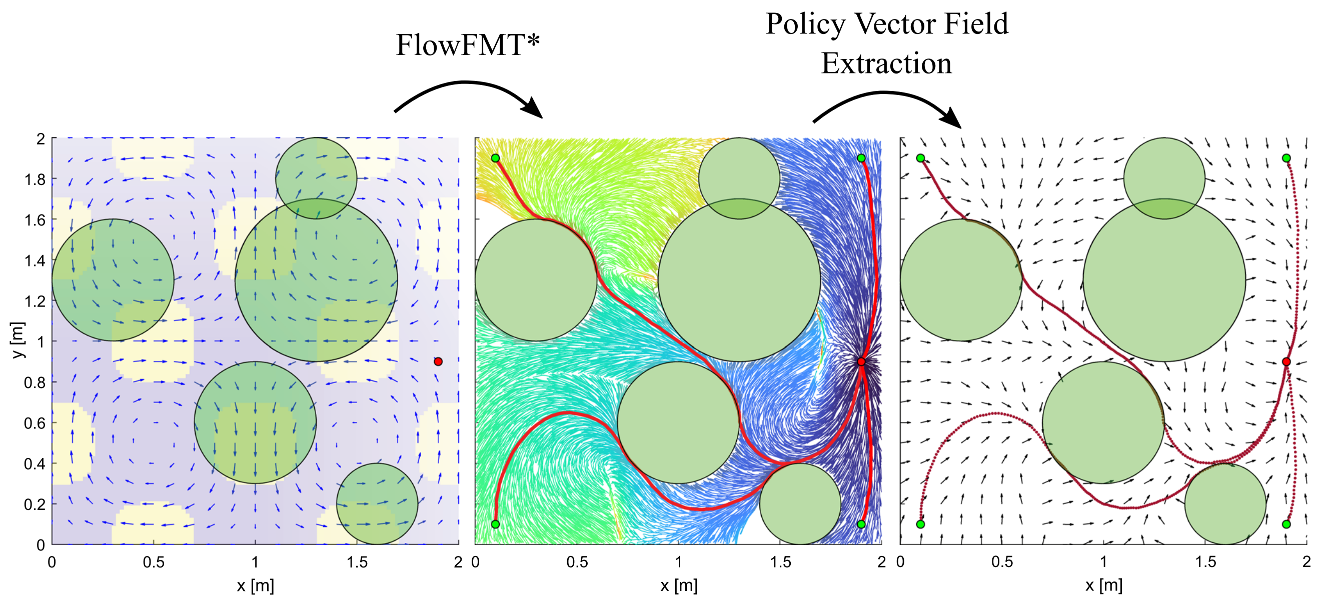


Fig. 1 Given an environment with a time-invariant flow (→) and prohibited regions (🟩), the Policy-FlowFMT* algorithm proposed in this paper creates a tree (represented in the center by vertices colored by their costs) that encodes the optimal motion policy to a specified goal (●). Besides being directly used to provide optimal trajectories to the goal (—), the tree can also generate a vector field (→), which can be applied online for robot control (· · ·)

dle incorrectness and incompleteness issues caused by the presence of strong currents, a sliding wavefront expansion method was proposed by Soulignac (2011). More recently, other path planning methods based on A* for such 2D environments were introduced by Koay and Chitre (2013); Kularatne et al. (2018). Minimum-time and minimum-energy cost functions were developed by Kularatne et al. (2018) using flow-oriented coordinates, which considered the feasibility problem implicitly. Most of the existing research for AUVs has considered 2D or quasi-two-dimensional environments, but Kulkarni and Lermusiaux (2020) proposed a level-set solution based on exact differential equations for time-varying and 3D scenarios. Tri-dimensional path planning for AUVs was also considered by Zhai et al. (2022), who developed an efficient method that partitions the flow into a set of regions with constant fields, thus transforming the problem into a mixed integer optimization problem (MIP), and uses a combination of branch-and-bound breath-first-search and the method of evolving junctions (Li et al., 2017) to compute the final optimal path.

The problem was also tackled in the realm of Unmanned Aerial Vehicles (UAVs). Path planners for a gliding aircraft under complex wind fields were studied by Langelaan (2008); Chakrabarty and Langelaan (2013). The proposed planner grow a tree of feasible trajectories from a discrete set of allowable inputs and weigh the branches using a cost function that accounts for changes in total energy and distance to the goal. A real-time environment-aware planner has been proposed by Oettershagen et al. (2017) based on a variation of the Rapidly Exploring Random Trees star (RRT*) (Karaman & Frazzoli, 2011) algorithm that approximates the dynamics of the aircraft using Dubins' Airplane (Chitsaz & LaValle, 2007) and incorporates non-uniform wind field data in its heuristics. The method calculates the Dubins Airplane paths considering zero wind and then deforms the paths using the wind field. Later, it uses an iterative method to generate a virtual goal that will lead to the actual goal position after the path is deformed by the wind. However, this iterative method fails for strong currents.

In a previous work of our group, we used a sampling-based approach, based on the RRT algorithm, to create trajectories considering the strong winds of the atmosphere of Venus (Martinez Rocamora Jr et al., 2022). We first calculate connections without considering the wind and then integrate the wind along the calculated paths to calculate a wind drift. Although efficient, our method finds sub-optimal trajectories. An extension of the method to find optimal trajectories would require rewiring the search tree, as the RRT* does, which is difficult to perform under the influence of strong winds. Very recently, a few research groups focused their attention on applying the Fast Marching Tree star (FMT*) algorithm (Janson et al., 2015), which can find optimal paths faster and without tree rewiring. FMT* is used by Lee et al. (2017) for energy-optimal planning of underwater gliders navigating time-invariant flows. The connection between nodes is performed by calculating the "trim states" required to achieve straight-line motion. Bonin et al. (2023) adapted the FMT* algorithm to time-optimal motion planning of ultralight gliders navigating in the time-invariant wind flows. The connections between nodes are calculated using a simplified optimal control problem that relies on a two-point boundary solver. Both previous papers consider 3D scenarios and raise reachability problems due to strong opposing flows. However, different from the analytical procedure proposed in the present paper, these methods numerically calculate the reachability set by solving the vehicle's equation of motion.

This paper presents a different sampling-based algorithm based on FMT* to solve the optimal motion planning problem for holonomic vehicles navigating strong 3D environmental flows. Unlike previous FMT*-based methods, our algorithm explicitly handles the feasibility problem created by situations where the flow speed is larger than the vehicle's maximum speed relative to the flow. To achieve that, we extended the use of reachability cones defined by Soulignac (2011) to 3D and used them to remove unreachable vertices in constructing the search tree. We also use reachability cones to determine accurate minimum-time and minimum-energy cost functions for the problem. Additionally, based on a few modifications to the proposed path planning algorithm, we present a method that finds a motion policy for the vehicle. The policy is represented as a vector field that serves as a feedback planner that optimally guides the vehicle from any valid configuration to its goal. An illustration of the method is shown in Fig. 1. In summary, the main contributions of this paper are (1) a motion planner based on FMT* that accounts for the feasibility of the trajectory in the presence of strong flows and prohibited regions (e.g., obstacles), (2) a method to obtain minimum-time and minimum-energy cost functions based on the reachability condition, (3) a method that creates a vector-field policy for optimal navigation in strong flows. We provide the source code of our methods and, for comparison, the setup environment for the equivalent optimal control solver (OCS).

The rest of this paper is organized as follows. The next section introduces the problem we are solving in the paper. The proposed methods and algorithms are explained in Sect. 3 and experiments are provided in Sect. 4. Conclusions and future work are presented in Sect. 5.

2 Problem definition

2.1 Environment

In this paper, a single robot navigates within a fluid (most commonly air or water) situated in a tri-dimensional (3D) environment $\mathcal{W} \subset \mathbb{R}^3$. The fluid is assumed to be flowing, and its motion is modeled as a time-independent vector field $F_c : \mathcal{W} \mapsto \mathbb{R}^3$ that maps each position $\vec{x} = [x, y, z]^\mathsf{T} \in \mathcal{W}$ to a flow velocity vector $\vec{c}(\vec{x}) = [c_x(\vec{x}), c_y(\vec{x}), c_z(\vec{x})]^\mathsf{T} \in \mathbb{R}^3$. The vector field F_c that represents the flow is continuous, i.e., given any point $\vec{x}_0 \in \mathcal{W}$, there is a ball $B_r(\vec{x}_0) = \{\vec{x} \in \mathcal{W} \mid \|\vec{x} - \vec{x}_0\| < r\}$ of radius $r \in \mathbb{R}^+$ centered in $\vec{x}_0$, where $\|F_c(\vec{x}) - F_c(\vec{x}_0)\| < \epsilon$, for every arbitrary $\epsilon \in \mathbb{R}^+$. Intuitively, if we consider two points in space that are sufficiently close to each other, the velocities of the flow at these points are also similar. Finally, we assume that the environment may contain prohibited regions, $\mathcal{O}$, which may be seen as obstacles for the vehicle but not for the fluid (e.g., "no-fly zones" for an aircraft). Therefore, the region where the vehicle can safely navigate, or free space, is given by $\mathcal{W}_{free} = \mathcal{W} \backslash \mathcal{O}$.

2.2 Robot

We assume a vehicle whose state space is given by $\mathcal{X} = X \times \dot{X}$, where $X \subseteq \mathcal{W}$ is the space of vehicle positions, and $\dot{X}$ is its respective tangent space. We consider that the vehicle is small (meter scale) compared to the environment and the distances traveled (kilometer scale). With this assumption in mind, we use a point with no orientation to represent the vehicle. In practice, this point can be seen as the vehicle itself or, for vehicles with more complicated dynamic models, as a setpoint that could be tracked with the aid of a nonlinear controller. Therefore, the vehicle's configuration space is equal to $\mathcal{W}$, where the vehicle is represented by its position $\vec{x}$ in the global reference frame. At position $\vec{x}$, the velocity of the vehicle in the global reference frame is given by

$$\vec{v}_g(\vec{x}) = \vec{v}_r(\vec{x}) + \vec{c}(\vec{x}), \tag{1}$$

where $\vec{v}_r(\vec{x})$ is the velocity of the vehicle relative to the flow and $\vec{c}(\vec{x})$ is the velocity of the flow. The relative velocity is bounded by a maximum speed v_r^{max}, such that $\|\vec{v}_r(\vec{x})\| \leq v_r^{max}$ for all $\vec{x} \in \mathcal{W}$. We assume that the flow speed can be greater than the vehicle's speed relative to the flow (i.e., $\|\vec{c}(\vec{x})\| \geq v_r^{max}$), which means that the vehicle may not be able to counteract the flow all over the environment.

2.3 Problem statement

The motion planning problem solved in this paper is to find the best trajectory σ^* in the set Σ of all feasible trajectories $\sigma(t) \colon [0, t_f] \to \mathcal{W}_{free}$, parameterized by $t \in [0, t_f]$, that move the vehicle from an initial configuration $\vec{x}_S$ at $t = 0$ to a set of admissible goal configurations $\vec{x}_G \in X_{goal}$ in a finite and arbitrary time $t = t_f$. Notice that t_f is not necessarily the same for all trajectories $\sigma \in \Sigma$. The cost function $\mathbf{C}(\sigma)$ to be minimized by our problem maps a trajectory to a positive real value $\mathbf{C} : \Sigma \mapsto \mathbb{R}^+$ and respects the constraints imposed on the vehicle by the environment. Time and energy cost functions will be minimized in this paper. Once these functions are defined, our problem is posed as:

$$
\begin{aligned}
\min_{\sigma} \quad & \mathbf{C}(\sigma) \\
\text{s.t.} \quad & \vec{v}_g(\sigma(t)) = \vec{v}_r(\sigma(t)) + \vec{c}(\sigma(t)) \,, \\
& \sigma(0) = \vec{x}_S \,, \\
& \sigma(t_f) = \vec{x}_G \,, \\
& \sigma(t) \in \mathcal{W}_{free}.
\end{aligned}
\tag{2}
$$

3 Methodology

This section presents our motion planning algorithm. First, we introduce minimum-time and minimum-energy cost functions necessary to specify our problem completely. Then, we define neighborhood functions that are created using reachability cones. Finally, we describe the Flow-Aware Fast Marching Tree (FlowFMT*) algorithm and an adaptation to compute motion policies.

3.1 Flow-aware cost functions

The problem presented in the previous section is not completely specified without a cost function. The literature uses two main categories of cost functions, depending on the task (Kirk, 2004). The first one specifies the minimum time problem, which tries to guide the vehicle to its goal as fast as possible, regardless of the energy spent. The other type specifies the minimum energy problem, which is useful when the vehicle has a finite amount of energy available (e.g., battery) and is required to spend this energy efficiently. Functions in these two main categories are defined next.

3.1.1 Minimum-time cost function

In early attempts to solve the minimum-time problem, the cost functions used were incorrect in the presence of strong flows (i.e., $\|\vec{c}(\vec{x})\| \geq \|\vec{v}_r(\vec{x})\|$ for any workspace position $\vec{x}$). That meant that trajectories found using such cost functions were not necessarily feasible. The limited reachability problem due to strong flows is illustrated in Fig. 2. For simplicity, in this figure and along the rest of this paper the index $\vec{x}$ is omitted from the vehicle's and flow's velocity components, i.e, $\vec{c}(\vec{x})$ is written as $\vec{c}$, $\vec{v}_r(\vec{x})$ as $\vec{v}_r$, and $\vec{v}_g(\vec{x})$ as $\vec{v}_g$.

In this paper, we provide an extension to 3D of the correct minimum-time cost function proposed by Soulignac (2011) to solve 2D problems (we recommend the reader to read the discussion about cost function incorrectness in (Soulignac, 2011)). Minimum-time cost functions are usually defined for continuous trajectories and n-waypoints discretizations,

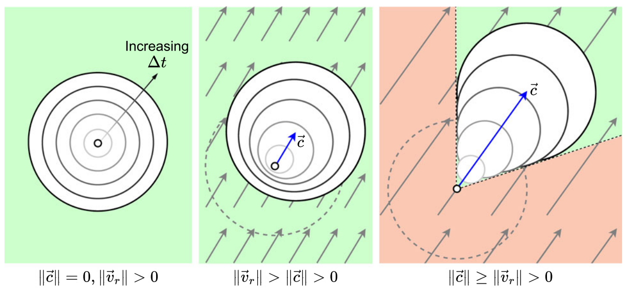

Increasing
Δt
$\vec{c}$
$\vec{c}$
$\|\vec{c}\| = 0, \|\vec{v}_r\| > 0$
$\|\vec{v}_r\| > \|\vec{c}\| > 0$
$\|\vec{c}\| \geq \|\vec{v}_r\| > 0$

Fig. 2 The reachable region for a vehicle moving on a constant flow is greatly affected by the ratio between the flow velocities and the vehicle's velocity relative to the flow. The circles represent iso-temporal lines given by the position reached by the vehicle after moving towards any direction for a fixed amount of time at a constant speed $\|\vec{v}_r\|$. When the flow velocity $\|\vec{c}\| = 0$, the circles are concentric and expand outwards with time, thus allowing the vehicle to reach the entire space (green). With $\|\vec{v}_r\| > \|\vec{c}\| > 0$, the circles are drifted accordingly, indicating that the vehicle travels less when moving against the flow but still reaches the entire space (green). When $\|\vec{c}\| \geq \|\vec{v}_r\| > 0$, the vehicle cannot counteract the flow, which creates a region that cannot be reached (red) even if the vehicle's velocity relative to the flow is directly opposed to the flow (Color figure online)

respectively, as:

$$
C_t(\sigma) = \int_0^{t_f} dt = t_f \quad \text{and} \quad C_t(\sigma) \approx \sum_{k=0}^{n} \Delta t^{(k)} \,. \tag{3}
$$

Although these equations are correct, previous work computed the travel time between two sufficiently close configurations (Δt) with simple heuristics that accounted for the traveled distance and an additional term that considered the relative heading of the flow and the vehicle (Petres et al., 2007) (thus assuming constant speed) or a ratio between the traveled distance and the sum of the vehicle speed and the speed of the flow opposing the movement (Garau et al., 2005; Blackmore et al., 2010). None of these heuristics were able to handle the strong flow condition.

When the flow speed gets larger than the maximum vehicle's speed relative to the flow, the drift caused by the flow cannot be compensated, thus creating regions that cannot be reached by the vehicle (Fig. 2). To create discrete functions that consider this characteristic, we follow the same procedure adopted by Soulignac (2011) but extend it to 3D. We assume that the vehicle can change direction instantaneously and that the minimum time path between two waypoints under a constant flow vector is a straight line if the connection is feasible. We also assume that two consecutive waypoints are close enough so that the flow is constant between them. Notice that these assumptions are quite realistic in large environments sampled with a very high number of waypoints. Thus, considering two positions $\vec{x}(t_A)$ and $\vec{x}(t_B)$, we can define a directional edge as $\vec{d} = \vec{x}(t_B) - \vec{x}(t_A) = [d_x, d_y, d_z]^{\mathsf{T}}$. The time interval between these two positions is $\Delta t = t_B - t_A$. By integrating the model in (1) and considering the vehicle's relative velocity $\vec{v}_r$ and flow velocity $\vec{c}$ to be constant along this edge we have:

$$\vec{d} = \vec{v}_g \Delta t = (\vec{v}_r + \vec{c})\Delta t \Leftrightarrow \begin{cases} (v_{r,x} + c_x)\Delta t = d_x \\ (v_{r,y} + c_y)\Delta t = d_y \\ (v_{r,z} + c_z)\Delta t = d_z. \end{cases} \tag{4}$$

By incorporating this equation on the inner product of the relative velocity ($\vec{v}_r \cdot \vec{v}_r = v_{r,x}^2 + v_{r,y}^2 + v_{r,z}^2$), we obtain an equation for the 3D cone shown in Fig. 3 as:

$$(\vec{c} \cdot \vec{c} - \vec{v}_r \cdot \vec{v}_r)\Delta t^2 - 2(\vec{d} \cdot \vec{c})\Delta t + \vec{d} \cdot \vec{d} = 0\,. \tag{5}$$

An in-depth discussion about the possible outcomes of this equation is derived by Soulignac (2011). In the next few paragraphs, we try to summarize these outcomes and connect them to our method.

In the case that $\vec{c} \cdot \vec{c} = \vec{v}_r \cdot \vec{v}_r$, (5) degenerates to a first-order equation, and the solution is simply $\Delta t = (\vec{d} \cdot \vec{d})/(2(\vec{d} \cdot \vec{c}))$. When $\vec{c} \cdot \vec{c} \neq \vec{v}_r \cdot \vec{v}_r$, the time cost for the straight line path between the two vertices is obtained by solving the quadratic equation for Δt as:

$$\Delta t = \frac{\vec{d} \cdot \vec{c} \pm \sqrt{(\vec{d} \cdot \vec{c})^2 - (\vec{c} \cdot \vec{c} - \vec{v}_r \cdot \vec{v}_r)(\vec{d} \cdot \vec{d})}}{(\vec{c} \cdot \vec{c} - \vec{v}_r \cdot \vec{v}_r)}\,. \tag{6}$$

In the case where $\vec{c} \cdot \vec{c} < \vec{v}_r \cdot \vec{v}_r$, the determinant $((\vec{d} \cdot \vec{c})^2 - (\vec{c} \cdot \vec{c} - \vec{v}_r \cdot \vec{v}_r)(\vec{d} \cdot \vec{d}))$ is positive, but the product of the two roots is negative, indicating that only one solution is valid (the negative sign solution, which results in positive Δt). Thus, in this case ($\vec{c} \cdot \vec{c} < \vec{v}_r \cdot \vec{v}_r$), the sign of the determinant defines whether or not the connection is feasible. When the computation of Δt in (6) returns complex numbers, the solution does not provide any physical meaning except indicating that the connection is unfeasible. In fact, for some given $\vec{v}_r$, $\vec{c}$, and $\vec{d}$, the cone angle β, as shown in Fig. 3, is derived from the geometry of the problem as:

$$\cos\beta = \frac{\sqrt{\vec{c} \cdot \vec{c} - \vec{v}_r \cdot \vec{v}_r}}{\|\vec{c}\|}\,. \tag{7}$$

To check if a straight line connection between two points is feasible, we can check if $\cos(\beta) \leq \cos(\beta_{max})$, where $\cos(\beta_{max})$ is found when $\|\vec{v}_r\| = v_r^{max}$. Thus, the complex solutions can be ruled out *a priori* (without computing the determinant in (6)) by assessing this inequality.

Interestingly, when the flow velocity is stronger than the relative velocity ($\vec{c} \cdot \vec{c} > \vec{v}_r \cdot \vec{v}_r$), the two possible values for Δt are positive and valid. In this case, the lower-time solution should be chosen for the minimum-time problem. Therefore, considering that the path is feasible and the vehicle moves with its maximum speed (i.e., $\vec{v}_r \cdot \vec{v}_r = (v_r^{max})^2$), we obtain the minimum-time solution between the two points for both the 2D and 3D cases by taking the negative sign solution of (6). Finally, to compute the total cost of a trajectory, the time cost Δt calculated using (6) is used in (3) for each segment that composes the n piece-wise linear path traversed by the vehicle from start to goal. Notice that the resulting minimum-time cost function is equivalent to the one derived by Zhai et al. (2022), both higher-dimension extensions of the cost function proposed by Soulignac (2011).

3.1.2 Minimum-energy cost functions

Common energy-based cost functions for continuous and discrete trajectories are, respectively, given by:

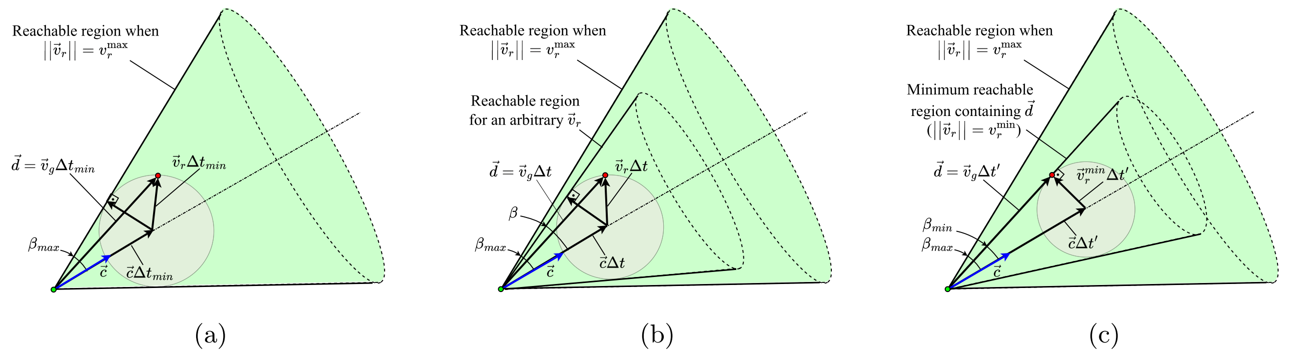
$$C_e(\sigma) = \int_0^{t_f} P(t)dt \quad \text{and}$$
$$C_e(\sigma) \approx \sum_{k=0}^{n} P^{(k)} \Delta t^{(k)}\,, \tag{8}$$

where P is the power required to overcome the drag. This required power can be modeled as a polynomial of the vehicle's speed relative to the flow to represent diverse forms of drag (e.g., viscous, pressure, or lift-induced drag):

$$P(t) = \sum_{i=0}^{n} \alpha_i \|\vec{v}_r(t)\|^i\,. \tag{9}$$

From (9) and (8), if the power model has $n > 1$, we observe that the influence of the vehicle speed is predominant in the energy cost. Thus, to minimize the cost function, it is necessary to minimize the vehicle's speed along the planned trajectory. However, this is not trivial as small vehicles' speeds relative to the flow may yield feasibility problems, since they reduce the reachable region as shown in Fig. 2. Further, this type of energy-based cost function can create situations where the resultant energy cost could be as small as wanted as long as it is acceptable to have very long terminal times.

Some methods like Zhai et al. (2022) or Kularatne et al. (2018) address this issue by combining time- and energy-based cost functions. This is achieved by adding a component that is proportional to time (known as hotel cost), which penalizes larger final times, to the energy-cost function shown above. In this work, we propose an alternative to find the energy cost for each segment of the discrete trajectories without explicitly including such a term. For a given pair of sampled positions (start and goal), our solution finds the minimum speed v_r^{min} that guarantees that the vehicle can still reach the goal position from the start position. To find



Reachable region when $||\vec{v}_r|| = v_r^{\max}$
$\vec{d} = \vec{v}_g \Delta t_{min}$
$\vec{v}_r \Delta t_{min}$
β_{max}
$\vec{c}$
$\vec{c}\Delta t_{min}$
(a)
Reachable region when $||\vec{v}_r|| = v_r^{\max}$
Reachable region for an arbitrary $\vec{v}_r$
$\vec{d} = \vec{v}_g \Delta t$
$\vec{v}_r \Delta t$
β
β_{max}
$\vec{c}$
$\vec{c}\Delta t$
(b)
Reachable region when $||\vec{v}_r|| = v_r^{\max}$
Minimum reachable region containing $\vec{d}$ ($||\vec{v}_r|| = v_r^{\min}$)
$\vec{d} = \vec{v}_g \Delta t'$
$\vec{v}_r^{min} \Delta t'$
β_{min}
β_{max}
$\vec{c}$
$\vec{c}\Delta t'$
(c)

Fig. 3 Computation of the reachable region. Given a time interval Δt, the displacement due to the flow velocity is given by $\vec{c}\Delta t$, while the displacement due to the vehicle's velocity relative to the flow is the surface of a sphere of radius $\vec{v}_r \Delta t$. Reachability cones are found by considering different values for these two parameters $\vec{v}_r$ and Δt as shown in Fig. 2. In **a**, the largest reachable cone is defined by β_{max}, which is obtained from when $\|\vec{v}_r\| = v_r^{max}$ using (7). This case also corresponds to the minimum time Δt_{min} connecting the start position (●) to the goal position (●). In **b**, considering $\|\vec{v}_r\| < v_r^{max}$, the cone angle β is smaller than β_{max}. In **c**, a cone defined by β_{min} is the limit case containing the vector $\vec{d}$ which connects the start position (●) to the goal (●) position. The geometry of this minimum reachable cone is used to calculate a v_r^{min}, which is used in our minimum-energy cost function. Notice that a lower speed increases the time that it takes to move by $\vec{d}$, i.e., $\Delta t < \Delta t'$ (Color figure online)

the minimum speed, we use the limit situation in which the surface of the reachability cone contains the vector $\vec{d}$ that connects start and goal samples. From the geometry of the problem, illustrated by Fig. 3, we obtain:

$$\frac{\sqrt{\vec{c}\cdot\vec{c} - (v_r^{min})^2}}{\|\vec{c}\|} = \frac{\vec{d}\cdot\vec{c}}{\|\vec{d}\|\|\vec{c}\|} \Leftrightarrow$$

$$v_r^{min} = \sqrt{\vec{c}\cdot\vec{c} - \left(\frac{\vec{d}\cdot\vec{c}}{\|\vec{d}\|}\right)^2}, \qquad (10)$$

which can be used with (9) and (6) to calculate $P(t)^{(k)}$ and $\Delta t^{(k)}$, which are then used in (8) to calculate the energy cost for each edge of the path.

3.2 Flow-aware neighborhood sets

Sampling-based motion planners, such as the one proposed in this paper, usually rely on the concept of neighborhood, which is usually used to specify a set of vertices (samples) that are candidates to be connected to the current vertices of a graph during its construction. One of the novelties of our method is the division of the neighborhood set into posterior and anterior sets. The posterior neighborhood set of a given vertex v, as shown in Fig. 4a, is the set of vertices that respect two conditions: 1) the vertices can be reached from v (i.e., they are contained in the reachability cone defined by v) and 2) the vertices are within a distance d_{max} from v. On the other hand, the anterior neighborhood set of a vertex v, as shown in Fig. 4b, is the set of vertices that respect two conditions: 1) the reachability cone defined by these vertices contains v and 2) the vertex v is within a distance d_{max} from these vertices. The definition of these two sets allows us to handle the reachability constraint caused by the presence of strong currents when creating a sample-based algorithm, as explained next.

3.3 Flow-aware fast marching tree

Using the previously defined cost functions and neighborhood sets, this section presents specialized Fast Marching Tree star algorithms (FMT*) (Janson et al., 2015) that consider the reachability constraints imposed by the strong currents. We first show the standard version of the algorithm, for which the tree grows from the start position until it reaches the goal region. In the second part of the section, we slightly change the algorithm so it expands the tree from the goal position to every feasible point sampled in the environment and creates a policy.

3.3.1 Flow-aware FMT*

The Flow-Aware FMT* algorithm (FlowFMT*) proposed in this paper is shown in Algorithm 1. Changes with respect to the original FMT* algorithm (Janson et al., 2015) are displayed in red. The algorithm creates a tree graph with an empty edge set and the vertex set defined by m random samples of the space, the start, and the goal positions (line 2). Three sets of vertices (lines 3–4) are created: the set of unvisited vertices, which is initialized with all vertices but the start position; the open set, which initially has the start position; and the closed set, which is empty at the beginning. Then, two structures (N^A and N^P) that save two sets of neighbors (see Sect. 3.2) for each vertex are initialized (lines 6–9).

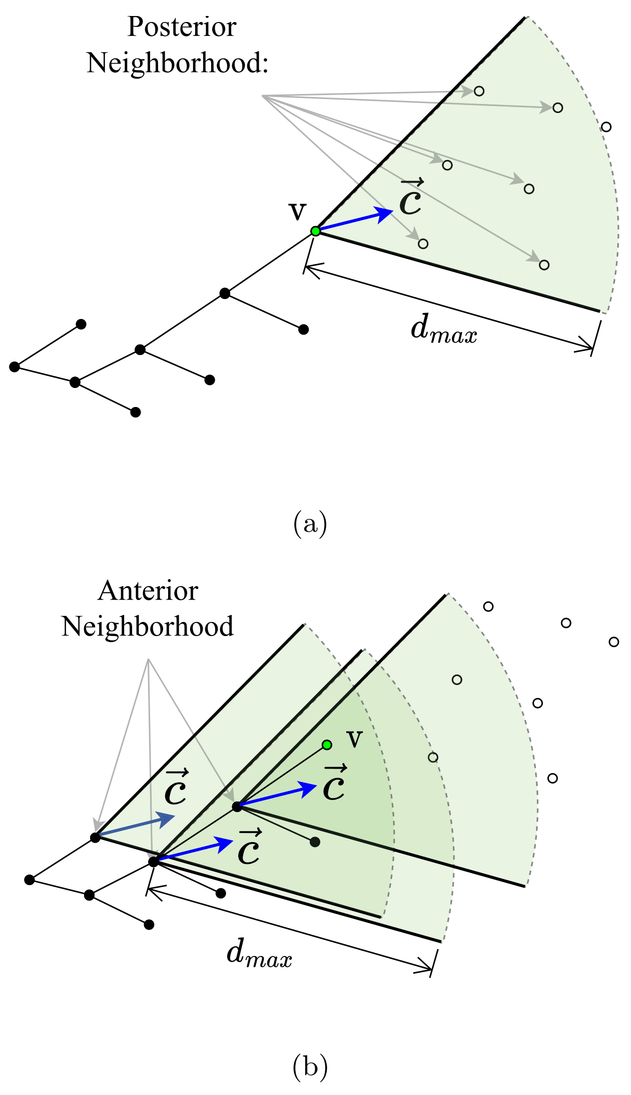


Fig. 4 Two types of neighborhood sets are defined for our method, substituting the standard Near() function used by the original FMT* algorithm. Given a vertex v, we define **a** the posterior neighborhood, with vertices that can be reached from v, and **b** the anterior neighborhood, with vertices that can reach the vertex v

Algorithm 1 Flow-Aware FMT* Algorithm

```
1: function $\sigma$ = FLOWFMT*($x_S$, $x_G$, $V_{pool}$)
2:     $V \leftarrow x_S \cup V_{pool} \cup x_G$, $E \leftarrow \varnothing$
3:     $V_{unvisited} \leftarrow V \backslash \{x_S\}$, $V_{open} \leftarrow x_S$
4:     $V_{closed} \leftarrow \varnothing$
5:     $z \leftarrow x_S$
6:     $N_z^P \leftarrow Post.Nbhd(\varnothing, V \backslash \{z\}, z, r_n)$
7:     $N^P \leftarrow Save(\varnothing, N_z^P, z)$
8:     $N_z^A \leftarrow Ant.Nbhd(\varnothing, V \backslash \{z\}, z, r_n)$
9:     $N^A \leftarrow Save(\varnothing, N_z^A, z)$
10:    while $z \neq x_G$ do
11:        $V_{open,new} \leftarrow \varnothing$
12:        $X_{near} = N_z^P \cap V_{unvisited}$
13:        for $x \in X_{near}$ do
14:            $N_x^A \leftarrow Ant.Nbhd(N^A, V \backslash \{x\}, x, r_n)$
15:            $N^A \leftarrow Save(N^A, N_x^A, x)$
16:            $Y_{near} \leftarrow N_x^A \cap V_{open}$
17:            $y_{min} \leftarrow \text{argmin}_{y \in Y_{n.}} \{c(y) + Cost(y, x)\}$
18:            if $CollisionFree(y_{min}, x)$ then
19:                $E \leftarrow E \cup \{(y_{min}, x)\}$
20:                $V_{open,new} \leftarrow V_{open,new} \cup \{x\}$
21:                $V_{unvisited} \leftarrow V_{unvisited} \cap \{x\}$
22:                $c(x) = c(y_{min}) + Cost(y_{min}, x)$
23:            end if
24:        end for
25:        $V_{open} \leftarrow (V_{open} \cup V_{open,new}) \backslash \{z\}$
26:        $V_{closed} \leftarrow V_{closed} \cup \{z\}$
27:        if $V_{open} = \varnothing$ then
28:            return $\varnothing$
29:        end if
30:        $z \leftarrow \text{argmin}_{y \in V_{open}} \{c(y)\}$
31:        $N_z^P \leftarrow Post.Nbhd(N^P, V \backslash \{z\}, z, r_n)$
32:        $N^P \leftarrow Save(N^P, N_z^P, z)$
33:    end while
34:    return $GetPath(x_G, T(V_{open} \cup V_{closed}, E))$
35: end function
```

The algorithm executes until no vertices are found on the open set or a path to the goal is found (lines 10 and 29). At each iteration, the minimum cost vertex of the open set is computed (line 30), and the posterior neighborhood of this vertex is calculated or retrieved (lines 31–32). This set is intersected with the set of unvisited vertices (line 12), and for each of the vertices x in the intersection, a search for a locally optimal one-step connection is performed by testing connections from all the vertices y in the intersection of the anterior neighborhood and the open set to x (lines 13–17). Up to this point, no collision checks are done. Once the locally-optimal one-step connection is found, it is tested for collision (line 18). If no collisions are detected, the edge is added to the tree (line 19). The newly added vertex is added to the open set and removed from the unvisited set (lines 20–21). After looping through all vertices in the posterior neighborhood, the query vertex is removed from the open set and added to the closed set (lines 25–26). The neighborhood sets are calculated using the AnteriorNeighborhood (shortened to Ant.Nbhd) and PosteriorNeighborhood (shortened to Post.Nbhd) functions as shown in Algorithm 2. The difference between the two functions is given by the definition of $\vec{d}$ (line 11).

3.3.2 Computing a policy using flow-aware FMT*

More useful than simply obtaining a path from the start to the goal, a policy defines paths from all positions in the space to the goal. A policy can be represented by a vector field, which can be used as a feedback system to compensate for disturbances (LaValle, 2006). Since the FMT* method is a graph-based level-set method that computes increasing levels of the cost function as the tree marches to cover the environment, it can be easily used to compute a policy. We then propose a few modifications to the Flow-Aware FMT* algorithm (Algorithm 1) to make this possible: (i) the root of the tree (lines 2–5) is set to the goal position; (ii) the anterior (lines 14–15) and posterior (lines 31–32) neighborhood sets are swapped; (iii) the order of the vertices on the cost function and collision checking functions (lines 18–19, and

Algorithm 2 Anterior/Posterior Neighborhood Set Computation Functions

```
1: function $N_q$ = ANT./POST.NBHD($N, V$, q, $r_n$)
2:   if $\exists N$(q) then
3:     return $N$(q)
4:   end if
5:   $N_q \leftarrow \varnothing$
6:   $\vec{q} \leftarrow GetPosition$(q)
7:   $\vec{c} \leftarrow \mathbf{F}_c(\vec{q})$
8:   $\cos(\beta_{max}) \leftarrow \sqrt{\vec{c}\cdot\vec{c} - (v_r^{max})^2}/\|\vec{c}\|$
9:   for v $\in V$ do
10:    $\vec{x} \leftarrow GetPosition$(v)
11:    $\vec{d} \leftarrow \vec{q} - \vec{x}$ (Ant.) or $\vec{d} \leftarrow \vec{x} - \vec{q}$ (Post.)
12:    $\cos\beta \leftarrow (\vec{d}\cdot\vec{c})/(\|\vec{d}\|\|\vec{c}\|)$
13:    if $\|\vec{d}\| < r_n$ then
14:      if $\|\vec{c}\| \geq v_r^{max}$ then
15:        if $\cos\beta \geq \cos(\beta_{max})$ then
16:          $N_q \leftarrow N_q \cup$ v
17:        end if
18:      else
19:        $N_q \leftarrow N_q \cup$ v
20:      end if
21:    end if
22:  end for
23:  return $N_q$
24: end function
```

22) are inverted; (iv) the stopping criteria (lines 10 and 27) is changed to only interrupt the procedure if the open set becomes empty.

The outcome of these modifications is shown in Algorithm 3. Changes with respect to the standard Flow-Aware FMT* algorithm (Algorithm 1) are displayed in red. Notice that Algorithm 3 builds a tree from the goal position but still considers that the direction of motion of the vehicle is towards the goal. The dynamic programming procedure (line 17) considers only valid connections given the sets defined in Sect. 3.2.

A policy is encoded in the resulting tree. Once this tree is found, the vehicle's velocity in the global reference frame $\vec{v}_g$ can be calculated using a multivariate interpolation around each position $\vec{x}$ in the continuous space. We propose using an inverse distance weighting interpolation of the vertices of the tree that are close to $\vec{x}$ to extract the policy vector field as:

$$\vec{v}_r(\vec{x}) = \overline{\vec{v}_g}(\vec{x}) - \vec{c}(\vec{x}) = \frac{\sum_{i=0}^{k} w_i(\vec{x})\,\vec{v}_{g,i}}{\sum_{i=0}^{k} w_i(\vec{x})} - \vec{c}(\vec{x}), \qquad (11)$$

where $\vec{v}_{g,i}$ $(0 \leq i \leq k)$ are the vehicle's velocities in the global reference frame at the k vertices v_i of the tree that are within a distance r from $\vec{x}$ (i.e., $\mathrm{v}_i \in V \cap \mathcal{B}_r(\vec{x})$). The process is illustrated by Fig. 5. The weight function is given by $w_i(\vec{x}) = 1/(\vec{x} - \vec{x}_i)^p$ where p is a parameter (commonly, $p = 2$). Notice that we assume that $\vec{x}$ is located in a region from which the goal can be reached.We do not include any further development in this paper, but simple heuristics, like counting the number of vertices of the tree in the ball $\mathcal{B}_r(\vec{x})$ can help identify whether $\vec{x}$ belongs to a feasible region or not. From (1) the vehicle's velocity relative to the flow can be computed by subtracting the interpolated value from the flow velocity at this point.

Algorithm 3 Flow-Aware FMT* Algorithm (Policy Version)

```
1: function $T$ = POLICY- FLOWFMT*($\mathrm{x}_G, V_{pool}$)
2:   $V \leftarrow \mathrm{x}_G \cup V_{pool}$, $E \leftarrow \varnothing$
3:   $V_{unvisited} \leftarrow V\backslash\{\mathrm{x}_G\}$, $V_{open} \leftarrow \mathrm{x}_G$
4:   $V_{closed} \leftarrow \varnothing$
5:   z $\leftarrow \mathrm{x}_G$
6:   $N_z^P \leftarrow Post.Nbhd.(\varnothing, V\backslash\{z\}, z, r_n)$
7:   $N^P \leftarrow Save(N^P, N_z^P, z)$
8:   $N_z^A \leftarrow Ant.Nbhd.(\varnothing, V\backslash\{z\}, z, r_n)$
9:   $N^A \leftarrow Save(N^A, N_z^A, z)$
10:  while $V_{open} \neq \varnothing$ do
11:    $V_{open,new} \leftarrow \varnothing$
12:    $X_{near} = N_z \cap V_{unvisited}$
13:    for x $\in X_{near}$ do
14:      $N_x^P \leftarrow Post.Nbhd(N^P, V\backslash\{x\}, x, r_n)$
15:      $N^P \leftarrow Save(N^P, N_x^P, x)$
16:      $Y_{near} \leftarrow N_x^P \cap V_{open}$
17:      $y_{min} \leftarrow \mathrm{argmin}_{y \in Y_n}\{c(y) + Cost(x, y)\}$
18:      if $CollisionFree(y_{min}, x)$ then
19:        $E \leftarrow E \cup \{(y_{min}, x)\}$
20:        $V_{open,new} \leftarrow V_{open,new} \cup \{x\}$
21:        $V_{unvisited} \leftarrow V_{unvisited} \cap \{x\}$
22:        $c(x) = c(y_{min}) + Cost(x, y_{min})$
23:      end if
24:    end for
25:    $V_{open} \leftarrow (V_{open} \cup V_{open,new})\backslash\{z\}$
26:    $V_{closed} \leftarrow V_{closed} \cup \{z\}$
27:    z $\leftarrow \mathrm{argmin}_{y \in V_{open}}\{c(y)\}$
28:    $N_z^A \leftarrow Ant.Nbhd(N^A, V\backslash\{z\}, z, r_n)$
29:    $N^A \leftarrow Save(N^A, N_z^A, z)$
30:  end while
31:  return $T$ $(V_{open} \cup V_{closed}, E)$
32: end function
```

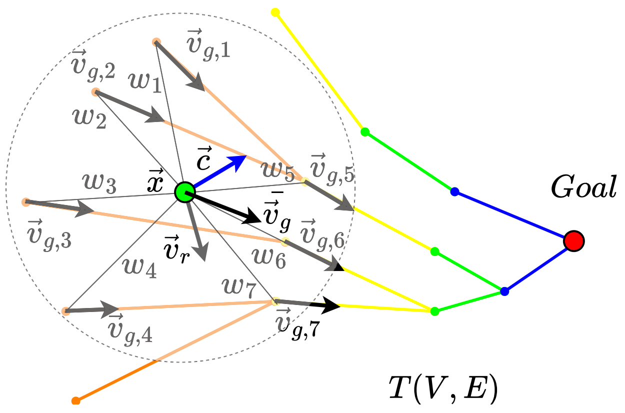


Fig. 5 Policy encoded by the tree $T(V, E)$ resulting from Algorithm 3. The global velocity $\vec{v}_{g,i}$ at the vertices i of the tree around the position of the robot $\vec{x}$ are combined to find the target global velocity $\overline{\vec{v}_g}$. By subtracting the local flow velocity $\vec{c}$, the robot's velocity $\vec{v}_r$, which is used to control the robot, can be retrieved as shown in (11)

3.3.3 Analysis

In this section, we discuss the computational complexity, the optimality, and the completeness of FlowFMT*. Notice that FlowFMT* is very similar to the original version of FMT*, as can observed by the highlighted lines in algorithms 1 and 3. The main difference is that, instead of a single spherical neighborhood set of radius r_n for each sample, FlowFMT* creates two spherical-conic neighborhood sets (intersection of a wind-driven elliptical cone and a sphere of radius r_n) for each sample. With this modification, we reduce the number of calls of functions $Cost()$ and $CollisionFree()$, which highly improves the computation time of the algorithm, but does not change its computational complexity when compared to the original FMT*. Therefore, for n samples, the computational complexity of FlowFMT* is still $O(n \log(n))$ in expectation, as discussed by Janson et al. (2015). To achieve this computational complexity, Algorithm 2 which finds the neighborhood sets for each query node should be implemented using KDTrees, so that the filtering by r_n happens in $O(\log(n))$.

The total number of samples in the two neighborhood sets of FlowFMT* can be larger or smaller than the number of samples of the spherical set of FMT*, depending on the relative speed of the wind. This, however, does not change the space complexity of the method, which also remains $O(n \log(n))$. The multiplicative constant can make FlowFMT* to use more memory for low-velocity flows and less memory for high-velocity flows.

For the original spherical neighborhood set with radius

$$r_n = \left(\frac{\lambda \log(n)}{n}\right)^{1/d}, \tag{12}$$

where λ is a positive constant and d is the space dimensionality, FMT* is proven to be asymptotically optimal (and therefore probabilistically complete) (Janson et al., 2015). The adoption of spherical-conic neighborhood sets by FlowFMT* does not change this characteristic because, in essence, the use of these sets is equivalent to the use of the original spherical set and a cost function $Cost(x, y)$ that would return infinite when sample y is unreachable from sample x. Thus, the idea behind FlowFMT*, which does not change the asymptotic optimality proof of FMT*, is to avoid several computations of the cost function (and also of $CollisionFree()$) by pre-selecting the neighboring samples that would not return an infinite cost (reachable neighbors). Hence, FlowFMT* inherits the characteristics of FMT* and remains asymptotically optimal.

The idea of reachable sets changes the overall behavior of FMT* in a few situations. Depending on the flow configurations, placement of prohibited regions (obstacles), and also on the maximum speed of the vehicle, there may exist a set of samples in the environment from where the vehicle cannot reach the goal. If the initial configuration belongs to this set, the motion planning problem is infeasible. In most cases, this happens when the vehicle is not able to counteract the flow speeds as it tries to avoid the prohibited regions and environment limits. Situations like this will be exemplified in the experiments of the next section. Although the presence of a set of samples that leads to infeasible problems does not change the optimality and completeness of either FMT* or FlowFMT*, it makes FlowFMT* more interesting. FMT* would find an element x of this set when the cost to all neighbors in the spherical set is infinite. FMT* would then connect x to one of its neighbors and at the end would return a path of infinity cost. On the other hand, FlowFMT* would explicitly define empty neighborhood sets for each sample that cannot reach the goal. This is a major advantage of FlowFMT*, which would immediately (without computing any cost or performing collision checks) return an empty path, instead of a path that cannot be physically followed by the vehicle, as returned by FMT*. Since FlowFMT* returns a path when the initial position can reach the goal and returns an empty path when this is not possible, FlowFMT* is probabilistically complete (a condition necessary for asymptotic optimality), even when it faces an infeasible motion planning problem. Notice that, with small modifications, FMT* could have the same behavior but at the cost of trying to connect samples in the infeasible region with all of their neighbors before detecting that the problem is actually infeasible.

The Policy-FlowFMT* algorithm has the same complexity as FlowFMT* since the differences between the algorithms are only related to the in-line ordering of some of its computations. We can then analyze the post-processing step given by (11). After the search tree is computed offline, this equation is supposed to be used at runtime. It is assumed that the tree resulting from Algorithm 3 would be stored in memory and the robot would only need to query for the vertices that are close to it. A simple search using KDTrees, which is $O(\log(n))$, can be used to assess the distance between the current robot position and the vertices of the tree and calculate the policy at $O(1)$. Therefore, the runtime complexity of Policy-FlowFMT* is $O(\log(n))$.

4 Experimental results

In this section, the performance of the proposed planner is evaluated in 2D and 3D scenarios, with and without prohibited regions. The resulting paths are always compared to the paths obtained by a state-of-the-art optimal control solver (OCS), the Imperial College London Optimal Control Software (ICLOCS2) (Nie et al., 2018). Additionally, by testing our method in environments previously published in the literature, we could also compare FlowFMT* with a Level Set Method (implemented by Zhai et al. (2022)), the Evolving Junction method (Zhai et al., 2022), true optimal solutions (Subramani et al., 2018), and a competitive grid-based approach (Kularatne et al., 2018). All the simulations were performed using an Intel® Core™ i9-9900K CPU at 3.6GHz, with 16 cores and 32GB of RAM.

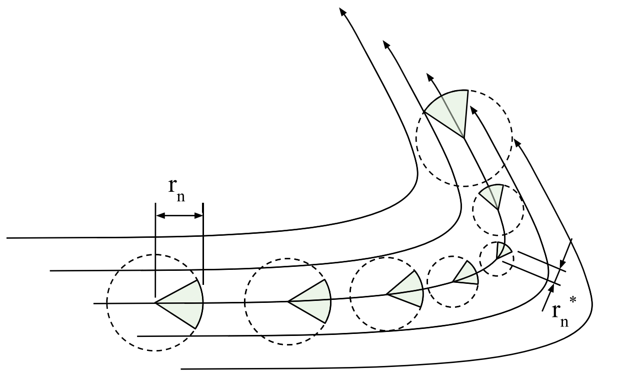


Fig. 6 Adaptive Neighborhood Radius for the FlowFMT*. The neighborhood size is decreased when the streamlines curve to improve on the assumption that the flow velocity is constant inside of the neighborhood

4.1 Canonical jet flow environment

The crossing of canonical jet flows are a set of illustrative problems described in prior works like Subramani et al. (2018) and Zhai et al. (2022), that bring possibilities to validate FlowFMT* on different aspects. One specific problem of this validation scenario is that it has discontinuities in the flow speeds, which contradicts our assumption that, in the limit, as the sample density gets higher, the flow speed between two neighboring samples tends to be constant. Although increasing the number of samples minimizes the effect of the discontinuity on the entire path, in our simulations we also included an adaptive neighborhood radius according to a vector-similarity metric based on the difference between the flow velocities in the start and goal vertices ($\vec{c}_{parent}$ and $\vec{c}_{child}$). The adaptive neighborhood radius is given by

$$r_n^* = r_n \left(1 - \frac{\|\vec{c}_{parent} - \vec{c}_{child}\|}{2c_{max}}\right), \tag{13}$$

where r_n is the original radius proposed for the FlowFMT* as shown in Equation (12). Figure 6 illustrates how this process works. Notice that r_n^* tends to r_n as our assumption of constant flow within the neighborhood region is better satisfied. If discontinuities or large gradients are present, the smaller radius will reduce the chances of connecting samples far from each other.

4.1.1 Two-dimensional jet

In this subsection we consider a vehicle that can move at a maximum speed $v_r^{max} = 10.0\text{ms}^{-1}$ relative to the flow. We limit the environment to a box of side 100 m, i.e., $\mathcal{W} = [0, 100] \times [0, 100]$ (the dimensional unit 'meters' is omitted in the rest of this section for simplicity). To match the scenario proposed by Subramani et al. (2018), we plan paths from the start position $\vec{x}_{start} = [20.0, 20.0]^\mathsf{T}$ to the goal position $\vec{x}_{goal} = [80.0, 80.0]^\mathsf{T}$. The flow environment is modeled as

$$\vec{c}(x, y) = \begin{cases} [20.0, 0.0]^\mathsf{T} \text{ if } y \geq 40 \text{ and } y \leq 60\,, \\ [0.0, 0.0]^\mathsf{T} \text{ otherwise}\,. \end{cases} \tag{14}$$

FlowFMT* is compared to the optimal results found using the methodology explained by Subramani et al. (2018), where a simple gradient-based numerical method is used to find the optimal solution to the jet crossing problem. The optimization is solved using Matlab (*fmincon* function). In Fig. 7, we show the optimal solution obtained numerically and solutions found using FlowFMT* for different number of samples. The cost using 25,600, 102,400, and 409,600 samples, were, respectively, $C_t(\sigma) = 6.2671$s, $C_t(\sigma) = 6.2608$s, and $C_t(\sigma) = 6.2569$s. The cost of the numerical solution was $C_t(\sigma) = 6.2523$s. Notice that the discontinuity in the flow field causes errors in FlowFMT*'s solution (a small shift compared to the optimal solution), but as the number of samples increases, and with the inclusion of the adaptive neighborhood radius this impact is significantly decreased.

Fig. 8 shows a solution computed using Policy-FlowFMT* with 102,400 samples. The tree starting at the goal position is shown in Fig 8a and the resultant vector field is shown in Fig 8b. Similar to the previous result, the computed trajectory is very close to the optimal one. Also, notice that, differently from other methods, our vector field explicitly indicates a region of the space from where the vehicle cannot reach the goal (no solution).

4.1.2 Three-dimensional jet

In this subsection, we consider a vehicle that can move at a maximum speed $v_r^{max} = 3.0\text{ms}^{-1}$ relative to the flow. We set the environment limits to the box $\mathcal{W} = [-10, 10] \times [-10, 10] \times [0, 20]$ (the dimensional unit 'meters' is omitted in the rest of this section for simplicity). To match what is found in the literature, in this section, we plan paths from the start position $\vec{x}_{start} = [0.0, 0.0]^\mathsf{T}$ to the goal position $\vec{x}_{goal} = [0.0, 20.0]^\mathsf{T}$. The flow environment is modeled as

$$\vec{c}(x, y, z) = \begin{cases} [0.5, 0.0, 0.0]^\mathsf{T} \text{ if } z \geq 0 \text{ and } z \leq 10\,, \\ [2.0, 1.0, 0.0]^\mathsf{T} \text{ if } z \geq 10 \text{ and } z \leq 15\,, \\ [0.0, 0.0, 0.0]^\mathsf{T} \text{ otherwise}\,. \end{cases} \tag{15}$$

FlowFMT* is compared to the optimal control solver ICLOCS2 (OCS), the Evolving Junction method (EJ) (Zhai et al., 2022), and the Level Set method (LS) when computing minimal-time paths. It is important to mention that LS is known to compute the shortest-time path over time-varying flows at the cost of longer computational times. On

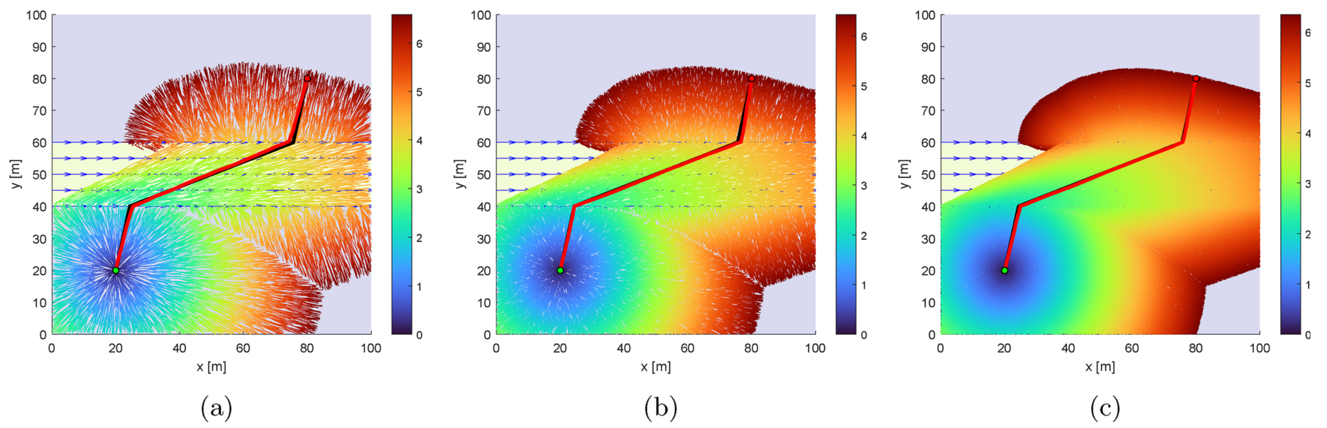


Fig. 7 Motion planning for a vehicle crossing a 2D jet flow using FlowFMT*. The robot moves from the start position (●) to the goal (●). We show the regions where the speed of the flow is greater (🟨) and smaller (🟦) than the maximum speed of the vehicle relative to the flow. The edges of the tree are colored based on the cost of the children's vertices. The numerical optimal solution is shown in black (—). The solution found with the adapted FlowFMT* is shown in red (—) for **a** 25,600, **b** 102,400, and **c** 409,600 samples (Color figure online)

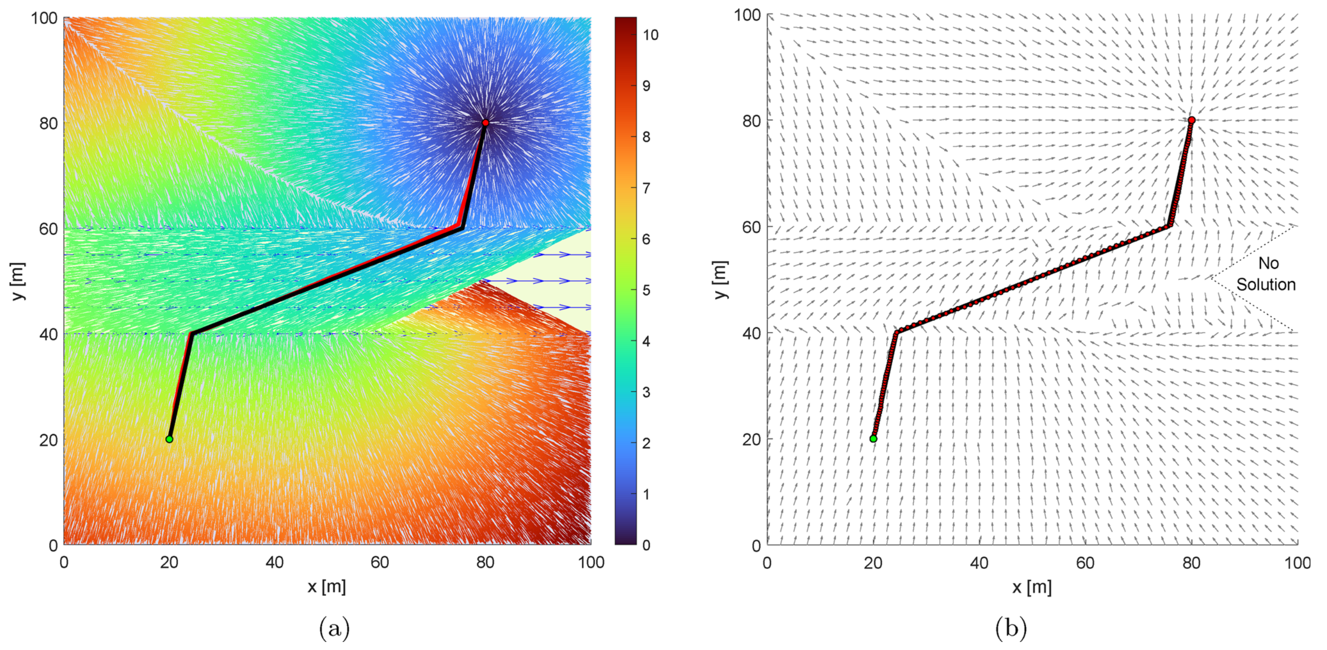


Fig. 8 Motion planning for a vehicle crossing a 2D jet flow using Policy-FlowFMT*. **a** Policy-FlowFMT* tree with the root at the goal position (●) for a two-dimensional jet flow. **b** Vector field computed with Policy-FlowFMT*. The figures show the following trajectories from start (●) to the goal (●): the optimal trajectory computed numerically (—), the trajectory from the tree computed by PolicyFlow* (—), and the trajectory computed by integrating the vector field (· · ·). From the regions where the field is not computed, the robot cannot reach the goal (no solution)

the other hand, EJ was proposed as an efficient solution that avoids errors induced by the discretization of the environment. Because it works on partitions of the flow, it explicitly handles discontinuous flows, as is the case of the environment used in this section.

Table 1 shows the comparison of the methods. The numerical results for EJ and LS came directly from Zhai et al. (2022). Besides the minimum-time cost, C_t, found by each method, Table 1 also shows the parameters of each path found, where angles θ_i and γ_i are elevation and azimuth angles of the path segments crossing the horizontal planes in $z = 10$ and $z = 15$ as defined by Zhai et al. (2022). Notice that all paths are very similar, showing that FlowFMT* converges to the optimal path, even in an environment with discontinuous flow. Figure 9 compares the trajectory found by FlowFMT* using 204,800 samples with trajectories found by OCS. The trajec-

Table 1 Comparison of Evolving Junction (EJ), Level Set (LS), ICLOCS2 (OCS), and FlowFMT* methods for the 3D jet flow environment. The segment angles for the FlowFMT* were estimated by calculating elevation and azimuth angles to the interpolated path crossing the horizontal planes in $z = 10$ and $z = 15$

	EJ	LS	OCS	FlowFMT*
θ_1	82.7924	83.5659	83.0881	83.7492
θ_2	62.0255	63.3118	63.4041	64.8360
θ_3	73.7397	73.80277	74.7784	75.3569
γ_1	−136.0775	−135.6592	−134.8846	−133.9709
γ_2	30.2293	30.2407	30.9735	32.8730
γ_3	−161.6199	−161.2246	−161.6005	−158.1282
C_t	6.9096	6.9826	6.9092	6.9090

tories found with EJ and LS were omitted from the plot due to their similarity to the other two trajectories. Since we did not implement the Evolving Junction (EJ) method, a time comparison between the methods cannot be made. Despite this, although EJ could be the best choice for environments with discontinuous and easy-to-segment flows, we believe that FlowFMT* would be a better choice for general flow environments, where the field cannot be easily partitioned into piece-wise constant subfields. The double-gyre flow environment, exploited in the next subsection, is an example of such an environment.

4.2 Double-gyre flow environment

In this subsection, we consider a vehicle that can move at a maximum speed $v_r^{max} = 0.05\text{ms}^{-1}$ relative to the flow. The environment is modeled using a three-dimensional (3D) version of double-gyre flow, which is usually used to evaluate motion planning on flows (see (Kularatne et al., 2018; Lee et al., 2017)), as:

$$\vec{c}(x, y, z) = \begin{cases} -\pi A \sin(\pi x/s)\cos(\pi y/s)\cos(\pi z/s) \\ \pi A \cos(\pi x/s)\sin(\pi y/s)\cos(\pi z/s) \\ \pi A \sin(\pi z/s)\,, \end{cases} \tag{16}$$

where A is the amplitude scaling factor that controls the maximum flow speed ($\max_{\vec{x}\in\mathcal{W}} \|\vec{c}(\vec{x})\|$), and s determines the characteristic length of the gyres. As suggested by Kularatne et al. (2018), we choose $A = 0.02$ so that the maximum flow speed is 0.0625ms^{-1}, which is greater than the vehicle's maximum speed relative to the flow. We limit the environment to a box of side 2 m, i.e., $\mathcal{W} = [0, 2] \times [0, 2] \times [0, 2]$ (the dimensional unit 'meters' is omitted in the rest of this paper for simplicity), and choose $s = 1$. To obtain the standard two-dimensional (2D) double-gyre flow, the slice where $z = 0$ is used.

4.2.1 Two-dimensional gyre

In a 2D environment ($z = 0$ in (16)), FlowFMT* is used to find paths from the start position $\vec{x}_{start} = [0.1, 0.1]^\mathsf{T}$ to a few different goal positions $X_{goal} = \{[0.1, 1.9]^\mathsf{T}, [1.5, 1.0]^\mathsf{T}, [1.9, 0.9]^\mathsf{T}, [1.9, 1.1]^\mathsf{T}, [1.9, 1.9]^\mathsf{T}\}$, as shown in Fig. 10. This figure shows the optimal-time paths for both minimum-time and minimum-energy cost functions and the resultant trees used to obtain these paths. The edges of the tree were colored by the cost of the children's vertices. The solution from the equivalent optimal control problem is shown for comparison. The general topology of the path for both methods is the same. Obtaining the best path from the OCS requires some tuning and an appropriate initial guess. Without these two requirements, the OCS can either take too long to converge (several minutes) or converge to a local optimum. We noticed that as we increased the number of samples, the path generated by FlowFMT* approached the best path obtained by the OCS without the extra work.

The absolute values of the minimum-time cost obtained by FlowFMT* and the OCS solution agree with the results obtained by Kularatne et al. (2018), even though the piecewise time cost between vertices was obtained using different formulations (ours is based on (Soulignac, 2011)). For comparison, considering the path from $[0.1, 0.1]^\mathsf{T}$ to $[1.9, 0.9]^\mathsf{T}$, the OCS solution obtained by Kularatne et al. (2018) presented a minimum-time cost of $C_t(\sigma) = 32.87$s while their proposed solution had $C_t(\sigma) = 32.92$s. Meanwhile, we observed a cost of $C_t(\sigma) = 32.86$s using OCS (set up with ICLOCS2), and $C_t(\sigma) = 32.88$s using FlowFMT*. Our method ran with 40,000 samples to match the spatial resolution of Kularatne et al. (2018). This indicates that both approaches generate similar valid solutions for the minimum-time problem.

Finding the true absolute values for the optimal paths considering the minimum-energy cost function is a more difficult task in the scenario involving gyres. Since the double-gyre flow model has parallel streamlines that circulate the center of each gyre, a vehicle without actuation would just drift around the center of its current gyre, carried by the flow. To reach a position in a different streamline while minimizing the energy spent, the vehicle must move from streamline to streamline with the minimum speed possible. In an ideal scenario, this speed and its resultant energy cost could be as small as we wanted. Clearly, very small costs would also result in solutions with an excessive final time. A discussion related to this problem was made by Kirk (2004). We then conclude that the true minimum energy cost is dependent on the minimum speed that the vehicle produces relative to the flow. This issue is not elucidated by Kularatne et al. (2018)

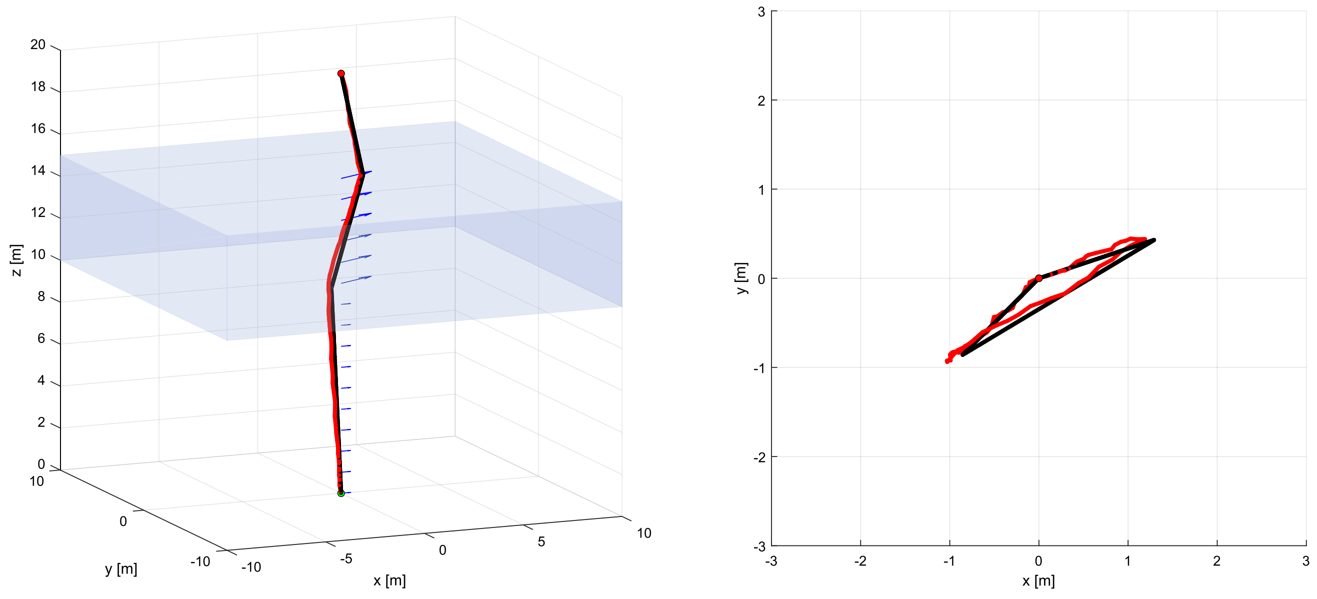


Fig. 9 Motion planning for a vehicle crossing a 3D jet flow. The robot moves from the start position (●) to the goal (●). The solution found with the adapted FlowFMT* using 204,800 samples is shown in red (—) and the optimal solution found using the OCS is shown in black (—) (Color figure online)

Table 2 Performance metric of the trajectories shown in Fig. 10 computed by FlowFMT* and optimal control method

	FlowFMT*		ICLOCS2	
Goal $\vec{x}_g$	C_t [s]	C_e [μJ]	C_t [s]	C_e [μJ]
$[1.9, 0.9]^\mathsf{T}$	32.89	1.120	32.86	2.532
$[1.9, 1.1]^\mathsf{T}$	35.12	1.779	35.06	1.747
$[1.5, 1.0]^\mathsf{T}$	34.47	0.834	34.43	0.737
$[1.9, 1.9]^\mathsf{T}$	30.15	1.681	30.11	1.204
$[0.1, 1.9]^\mathsf{T}$	27.58	1.584	27.62	2.143

Table 3 Performance and computational time metrics (in seconds) of the trajectories shown in Fig. 11 computed by FlowFMT* and the optimal control method (1) initialized with a straight line trajectory between start and goal and (2) initialized with the trajectory computed by FlowFMT*

	FlowFMT*		OCS (1)		OCS (2)	
#Obs	C_t	Δt_c	C_t	Δt_c	C_t	Δt_c
0	30.13	204.9	30.11	1.735	30.11	3.738
5	30.13	206.0	30.11	2.360	30.11	5.893
10	30.53	210.9	38.89	2.734	30.45	7.212
20	31.30	221.0	40.95	3.031	31.89	21.15
40	31.61	227.7	46.14	47.45	33.41	140.3
80	31.80	290.1	–	>1500	32.39	1299

when their OCS solution was explained. To allow the OCS to find a solution with a finite and reasonable time, we set the inferior limit of the vehicle's speed relative to the flow to 0.0001ms^{-1} (0.02% of the maximum relative speed). For FlowFMT*, we do not enforce a minimum speed since it is calculated using (10), which will be zero only in the rare cases where the next waypoint is completely aligned with the flow. Figure 10c and d show that our method can find solutions that are similar to the ones found by the OCS. Numerical comparisons for all the paths shown are presented in Table 2.

The average computational time spent by FlowFMT* to compute the paths in Fig. 10 was 74s, while the OCS was "generally" able to finish in a few seconds. However, we also observed that the OCS computational time can increase drastically (up to minutes) if obstacles (prohibited regions) are added and, consequently, more active constraints are in place, or if the problem is not set up correctly (poor initial guess, poor constraint limits, etc.). Furthermore, OCS solutions tend to converge to local minima, which does not happen with our graph-based solution. Depending on the initial guess, the OCS solutions either get stuck in suboptimal homotopies or show additional loops around gyres, introducing suboptimality.

This behavior is further explored in the experiment shown in Fig. 11, where we incrementally add randomly generated prohibited regions to a scenario and test FlowFMT* and OCS. The radii and center coordinates of the prohibited regions are sampled from uniform distributions, $U(0.02, 0.08)$ and $U(0.0, 2.0)$ respectively. In this experiment OCS is initialized in two forms: 1) with a straight

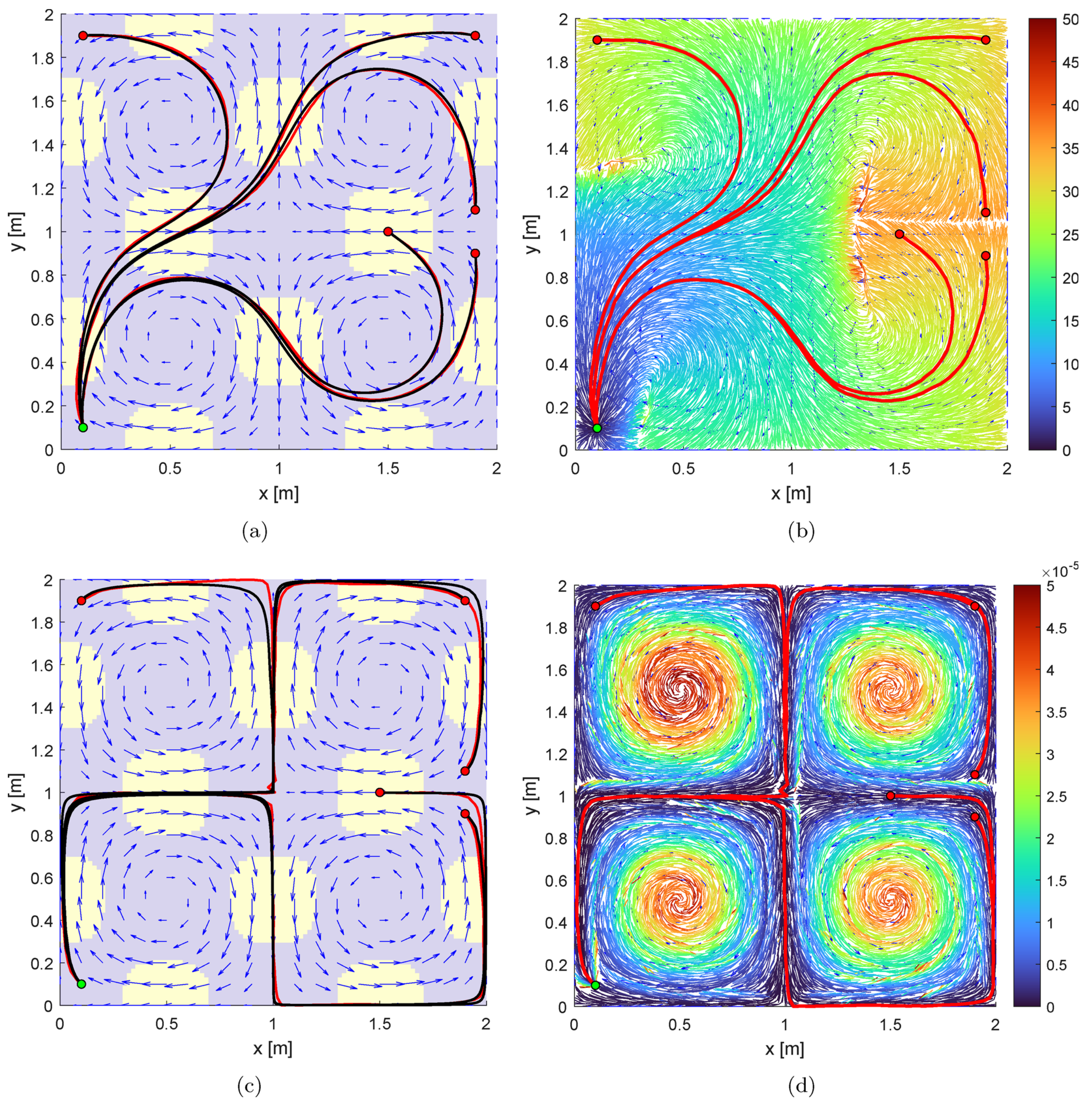


Fig. 10 Paths planned by FlowFMT* (—) in a two-dimensional double-gyre flow (→) compared to the associated optimal control problem (—). On the left, we show the regions where the speed of the flow is greater (🟨) and smaller (🟦) than the maximum speed of the vehicle relative to the flow. On the right, we show FlowFMT* trees, with vertices colored by their costs. The minimum-time cost function was used in (**a**) and (**b**) and the minimum-energy cost function in (**c**) and (**d**)

trajectory from start to end, and 2) using the optimal trajectory found with FlowFMT*. The optimal time costs and the computational times for each test are shown in Table 3. We observe that in the absence or with sparse distribution of prohibited regions (Fig. 11a and b), OCS can reach optimal trajectories in less time compared to trajectories of the same quality computed with FlowFMT*. However, as the number of prohibited regions grows, it is possible to see the trajectories found by OCS getting trapped in sub-optimal homotopies. Also, the computational time grows considerably. By comparing the computational time from situations with no prohibited regions (Fig. 11a) to environments with 80 prohibited regions (Fig. 11f) we noticed that the time grows 1.4× with FlowFMT*, mainly due to collision checking. On

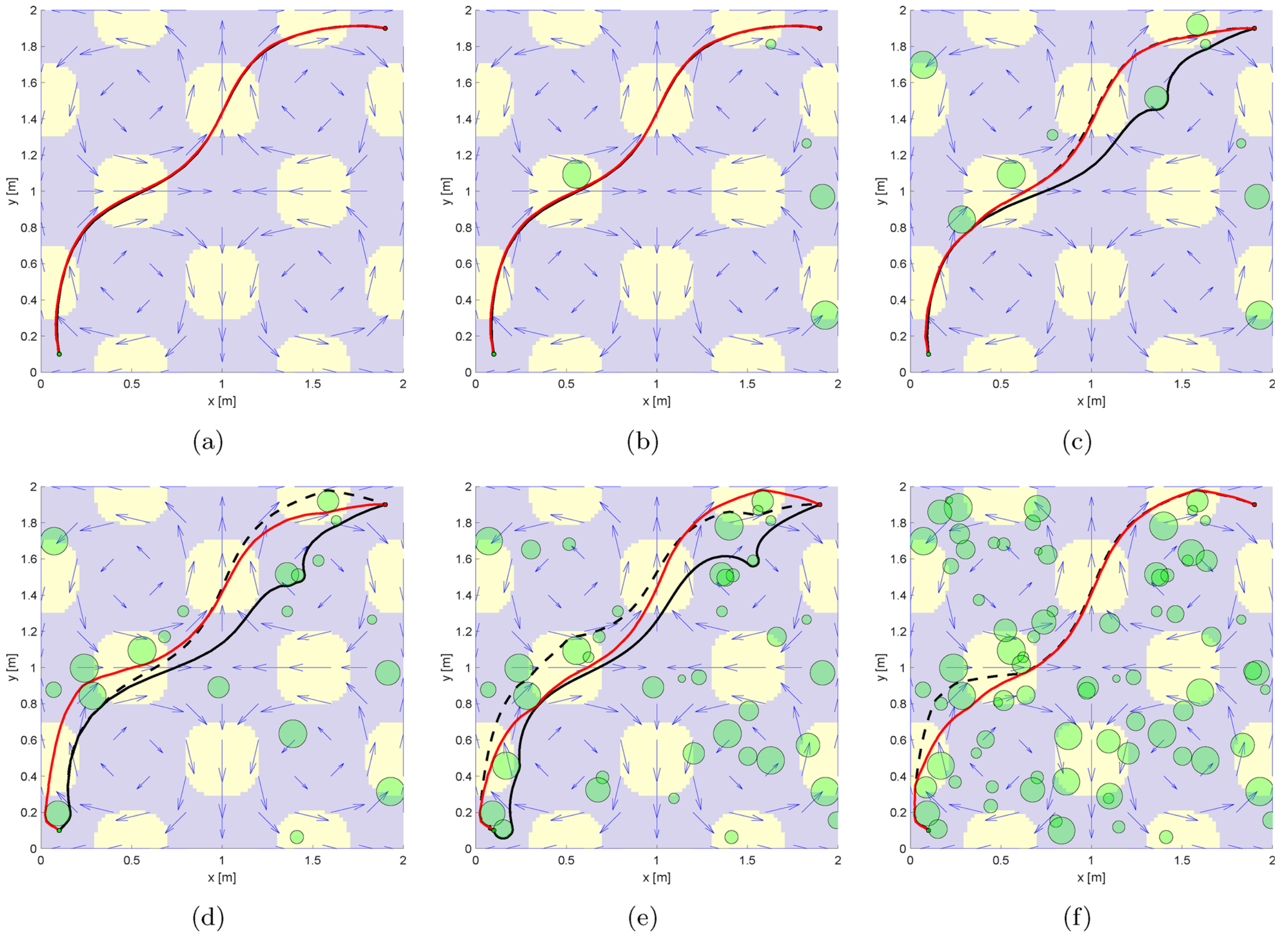


Fig. 11 Comparison of FlowFMT* and optimal control in a two-dimensional double-gyre flow with an increasing number of prohibited regions (🟩): **a** 0 regions, **b** 5 regions, **c** 10 regions, **d** 20 regions, **e** 40 regions, and **f** 80 regions. The solution trajectory computed by FlowFMT* (—) was found using 102,400 samples. For each scenario, two solutions were found with optimal control. The first was initialized with a simple straight trajectory (—) and the second using the trajectory found by FlowFMT* (- -)

the other hand, the computational time grows 347.5× for the OCS initialized with FlowFMT* trajectory. OCS with a straight line initialization did not converge before 25 min, when it was interrupted.

The Policy-FlowFMT* in Algorithm 3 was also compared to the solutions obtained with ICLOCS2. First, we found trajectories from 20 random initial positions to a single final position $\vec{x}_{goal} = [1.9, 0.9]^\mathsf{T}$ in the double-gyre environment without prohibited regions. We provided a straight line between the start and goal as the initial guess for the OCS. In our algorithm, we tested different resolutions, doubling the number of samples from 100×2^0 to 100×2^{10}. The results are shown in Fig. 12. It is possible to see that 2 of the 20 trajectories were dissonant, with FlowFMT* finding lower-cost solutions (in fact, better solutions). As mentioned before, this could be solved by providing a better initial guess for the OCS. Figure 12b shows how close the cost found by FlowFMT* gets to the "true" cost found by ICLOCS2 on the other 18 trajectories in terms of cost ratio (i.e., $C_t(\sigma_{\text{FlowFMT*}})/C_t(\sigma_{\text{OCS}})$). As expected, the cost ratio approaches one when the number of samples increases. Depending on the validity of the locally constant flow assumptions, FlowFMT* may not find trajectories with a cost that is necessarily higher than the OCS solutions. However, as the number of samples increases, the assumption becomes more valid, and the cost estimate becomes more realistic. Figure 12b also shows how the computational time to compute the policy increases as the number of samples increases. Despite that, remember that a policy is only computed once for each goal position, which can make this time diluted among the number of future trajectories that need to be computed using such a policy.

The Policy-FlowFMT* algorithm can also be used to compute a velocity vector field as shown in Fig. 13b for an environment with prohibited regions. This figure also shows simulated trajectories that illustrate how the vector field can

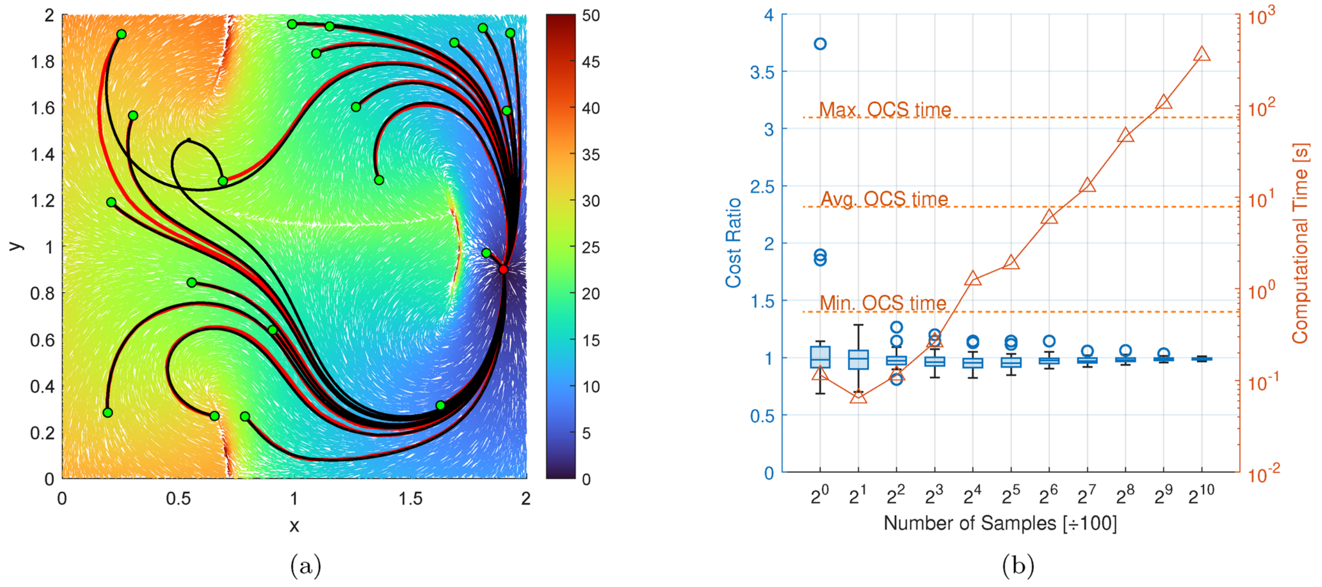


Fig. 12 Policy-FlowFMT* computed over the 2D double-gyre flow environment. In **a** we compare the solution for 20 random initial positions using our method and ICLOCS2. In **b** we show that our solution approaches the optimal solution with more samples at the cost of more computational time

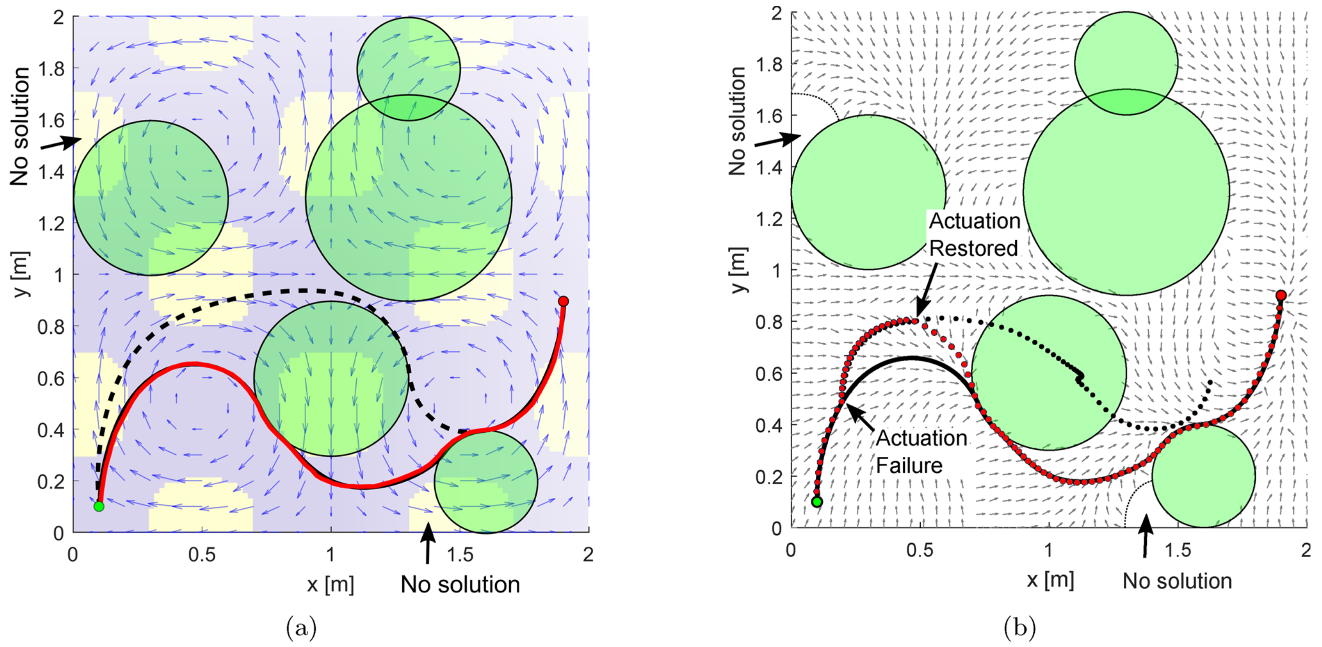


Fig. 13 Comparison with optimal control: **a** Comparison of FlowFMT* and optimal control in a two-dimensional double-gyre flow with prohibited regions (🟩). Both algorithms provide similar solutions when optimal control has a good initial guess (—). However, optimal control with a bad initial guess (- -) can return paths with a different homotopy. **b** Comparison of Policy-FlowFMT* and optimal control for real-time control in the presence of a disturbance. After an actuation fault, the vector field (→) created using Policy-FlowFMT* brings the vehicle to the goal following an optimal trajectory (···) while optimal control (···) leads the vehicle to cross a prohibited region

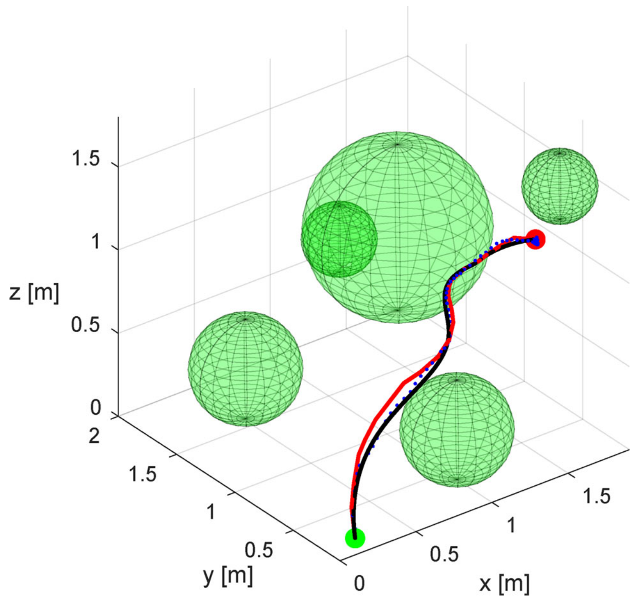


Fig. 14 Navigating in a 3D double-gyre flow with prohibited regions. The Policy-FlowFMT* solution (—) using 102,400 samples is compared to the solution (···) obtained from a simulation using the policy vector field (→), and the solution obtained from an optimal control problem solver (—)

help recover the vehicle after an actuation fault, while the OCS solution may lead to a collision. We considered that the OCS solution and the vector field are both used in real-time to control the vehicle. We then simulated that vehicle lost power for 10 s starting at $t = 5$ s. Figure 13b shows that at this moment, the vehicle drifts with the flow due to the lack of actuation. When the actuation is recovered, the OCS solution is not valid anymore, leading the vehicle to a prohibited region. In the best-case scenario, if a trajectory tracker was available, the vehicle maybe would be able to retake the path, at the cost of suboptimality of the resultant trajectory. In comparison, the vector field, no matter where the control is regained, would guide the vehicle through an optimal path from that position to the goal, as can be verified in Fig. 13b.

4.2.2 Three-dimensional gyres

Figure 14 shows the vehicle's trajectories from $\vec{x}_{start} = [0.1,\ 0.1,\ 0.1]^\mathsf{T}$ to $\vec{x}_{goal} = [1.9,\ 0.9,\ 0.9]^\mathsf{T}$ moving in a 3D double-gyre flow (as defined in (16)) with prohibited regions. We compared the performance of the optimal trajectories obtained by Policy-FlowFMT* using 102,400 samples and OCS (ICLOCS2). The OCS trajectory resulted in a time cost of $C_t(\sigma) = 28.18$s. The cost of the trajectory obtained directly from the tree built using Policy-FlowFMT* is $C_t(\sigma) = 28.91$s. By integrating the vector field resultant from the tree generated by Policy-FlowFMT* with a 0.1s time step, we obtain a trajectory that reaches the goal in 29.4s.

5 Conclusion

This paper presented a method for finding optimal paths and policies for holonomic vehicles moving in the presence of strong currents. We modified FMT* by defining new neighborhood sets that consider the directionality of the vehicle's movement due to the flow and reduce the number of calls to the cost function. These sets are defined using a reachability cone that restricts the space that can be reached when strong currents are found (i.e., the flow speed is greater than the vehicle's relative speed to the flow). The reachability cone is also used to define a minimum-energy cost function for this type of environment.

Additionally, we provide simple modifications to our original path-planning algorithm to calculate global policies to a goal. By using these methods, we can obtain paths with costs similar to the paths obtained using optimal control techniques without significantly increasing the computation time of the solution. Moreover, the proposed methods highly simplify the introduction of obstacles and other constraints, guaranteeing that the optimal homotopy is found when the number of samples is high enough. Finally, we show through a simulation that the policy generated by our method is inherently more resilient to actuation failures than optimal control. Therefore, the methods in this paper should be preferred over previous solutions, especially if the environment contains obstacles and when replanning is not desirable.

Our future work includes the application of the proposed methodology to the motion planning problem of aerial vehicles, also known as aerobots, for *in-situ* exploration of the atmosphere of Venus. These aerobots are up to three times slower than the superrotation winds present on the planet, and even if they offer 3D control, their reachability is constrained by the strong winds. To reach some specific locations, the motion planner needs to find trajectories that circumnavigate the planet. Consequently, for this application, the algorithms proposed in this paper need to be extended to allow for periods of time when the vehicle loses actuation (night side of the planet) due to the lack of solar energy. In these regions, the method needs to be able to let the aerobot drift passively with the flow.

Acknowledgements The authors thank L5 Automation, Inc. for supporting B. Martinez Rocamora Jr. during the revision process of this manuscript.

Author Contributions Conceptualization, B.M.R.J. and G.A.S.P.; methodology, B.M.R.J. and G.A.S.P.; software, B.M.R.J.; validation, B.M.R.J. and G.A.S.P.; formal analysis, B.M.R.J. and G.A.S.P.; investigation, B.M.R.J. and G.A.S.P.; resources, G.A.S.P.; data curation, B.M.R.J.; writing-original draft preparation, B.M.R.J.; writing-review and editing, B.M.R.J. and G.A.S.P.; visualization, B.M.R.J.; supervision, G.A.S.P.; project administration, G.A.S.P.; funding acquisition, G.A.S.P. All authors have read and agreed to the published version of the manuscript.

Funding This work was partially funded by NASA under grant #80NSSC23M0168. B. Martinez Rocamora Jr. was funded by the Statler Ph.D. Fellowship while he was a student at the Department of Mechanical and Aerospace Engineering at West Virginia University.

Data Availibility Not applicable.

Material availability Not applicable.

Code availability The code for this research is available in the following Git repository: https://bitbucket.org/wvufarolab/flowfmtstar.

Declarations

Conflict of interest The authors declare no Conflict of interest.

Ethical approval and consent to participate Not applicable.

Consent for publication Not applicable.

References

Alvarez, A., Caiti, A., & Onken, R. (2004). Evolutionary path planning for autonomous underwater vehicles in a variable ocean. *IEEE Journal of Oceanic Engineering, 29*(2), 418–429. https://doi.org/10.1109/JOE.2004.827837

Blackmore, L., Kuwata, Y., Wolf, M. T., Assad, C., Fathpour, N., Newman, C.,& Elfes, A. (2010). Global reachability and path planning for planetary exploration with montgolfiere balloons. IEEE international conference on robotics and automation. pp. 3581–3588

Bonin, L., Guitart, A., Delahaye, D., Prats, X., & Feron, E. (2023). Computing optimal trajectories for light soaring aircraft using fast marching tree star. Retrieved 2023-02-13, from https://hal-enac.archivesouvertes.fr/hal-03923994

Chakrabarty, A., & Langelaan, J. (2013). UAV flight path planning in time varying complex wind-fields. 2013 American control conference. pp. 2568–2574

Chitsaz, H., & LaValle, S. M. (2007). Timeoptimal paths for a Dubins airplane. IEEE conference on decision and control. pp. 2379–2384.

Garau, B., Alvarez, A.,& Oliver, G. (2005). Path planning of autonomous underwater vehicles in current fields with complex spatial variability: an A* Approach. IEEE international conference on robotics and automation pp. 194–198.

Greaves, J. S., Richards, A. M. S., Bains, W., Rimmer, P. B., Sagawa, H., Clements, D. L., Seager, S., Petkowski, J. J., Sousa-Silva, C., Ranjan, S., Drabek-Maunder, E., Fraser, H. J., Cartwright, A., Mueller-Wodarg, I., Zhan, Z., Friberg, P., Coulson, I., Lee, E., & Hoge, J. (2020). Phosphine gas in the cloud decks of Venus. *Nature Astronomy, 5*(7), 655–664. https://doi.org/10.1038/s41550-020-1174-4

Janson, L., Schmerling, E., Clark, A., & Pavone, M. (2015). Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International Journal of Robotics Research, 34*(7), 883–921.

Karaman, S., & Frazzoli, E. (2011). Samplingbased algorithms for optimal motion planning. *The International Journal of Robotics Research, 30*(7), 846–894.

Kirk, D. E. (2004). Optimal control theory: An introduction. Courier Corporation.

Koay, T.-B., & Chitre, M. (2013). Energy-efficient path planning for fully propelled AUVs in congested coastal waters. 2013 MTS/IEEE OCEANS-Bergen. pp. 1-9.

Kularatne, D., Bhattacharya, S., & Hsieh, M. A. (2018). Going with the flow: A graph based approach to optimal path planning in general flows. *Autonomous Robots, 42*(7), 1369–1387. https://doi.org/10.1007/s10514-018-9741-6

Kulkarni, C. S., & Lermusiaux, P. F. (2020). Three-dimensional time-optimal path planning in the ocean. *Ocean Modelling, 152*, 101644. https://doi.org/10.1016/j.ocemod.2020.101644

Langelaan, J.W. (2008). Tree-based trajectory planning to exploit atmospheric energy. American control conference. pp. 2328–2333.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge University Press.

Lee, J. J. H., Yoo, C., Hall, R., Anstee, S.,& Fitch, R. (2017). Energy-optimal kinodynamic planning for underwater gliders in flow fields. Australasian conference on robotics and automation.

Li, W., Lu, J., Zhou, H., & Chow, S.-N. (2017). Method of evolving junctions: A new approach to optimal control with constraints. *Automatica, 78*, 72–78.

Martinez Rocamora Jr., B., Juan, A. P. I., & Pereira, G. A. S. (2022). Towards finding energy efficient paths for hybrid airships in the atmosphere of venus. International conference on unmanned aircraft systems. pp. 386-393

Nagpal, L., & Samdani, K. (2017). Project loon: Innovating the connectivity worldwide. IEEE international conference on recent trends in electronics, information & communication technology. pp. 1778–1784

Nie, Y., Faqir, O., & Kerrigan, E. C. (2018). ICLOCS2: Try this optimal control problem solver before you try the rest. International conference on control. pp. 336–336

Oettershagen, P., Achermann, F., Müller, B., Schneider, D& Siegwart, R. (2017). Towards fully environment-aware UAVs: Real-time path planning with online 3D wind field prediction in complex terrain. Retrieved 2021-11-18, from arXiv:1712.03608

Petres, C., Pailhas, Y., Patron, P., Petillot, Y., Evans, J., & Lane, D. (2007). Path planning for autonomous underwater vehicles. *IEEE Transactions on Robotics, 23*(2), 331–341.

Rossi, F., Branch, A., Schodlok, M. P., Stanton, T., Fenty, I. G., Hook, J. V., & Clark, E. B. (2021). Stochastic Guidance of Buoyancy Controlled Vehicles under Ice Shelves using Ocean Currents. IEEE/RSJ international conference on intelligent robots and systems. pp. 8657–8664.

Rossi, F., Saboia, M., Krishnamoorthy, S., & Vander Hook, J. (2023). Proximal exploration of venus volcanism with teams of autonomous buoyancy-controlled balloons. *Acta Astronautica, 208*, 389–406.

Soulignac, M. (2011). Feasible and optimal path planning in strong current fields. *IEEE Transactions on Robotics, 27*(1), 89–98. https://doi.org/10.1109/TRO.2010.2085790

Subramani, D. N., Wei, Q. J., & Lermusiaux, P. F. (2018). Stochastic time-optimal path-planning in uncertain, strong, and dynamic flows. *Computer Methods in Applied Mechanics and Engineering, 333*, 218–237. https://doi.org/10.1016/j.cma.2018.01.004

VEXAG (2019). Roadmap for Venus Exploration (Tech. Rep.). Venus Exploration Analysis Group.

Zhai, H., Hou, M., Zhang, F., & Zhou, H. (2022). Method of evolving junction on optimal path planning in flows fields. *Autonomous Robots, 46*(8), 929–947.

Bernardo Martinez Rocamora Jr. received his bachelor's degree in aeronautical engineering and his master's degree in mechanical engineering from the University of São Paulo, in São Carlos, SP, Brazil, in 2016 and 2019, respectively. He received his Ph.D. degree in aerospace engineering from Benjamin M. Statler School of Engineering and Mineral Sciences at West Virginia University, WV, USA in December 2023. During his Ph.D., he was a member of the Field and Aerial Robotics (FARO) Laboratory and the recipient of the Statler Ph.D. Fellowship. He is currently a robotics research engineer for L5 Automation, Inc., and he is working on the development of autonomous strawberry harvesters. His research interests include field robotics, autonomous systems and robot motion planning.

Guilherme A. S. Pereira received his bachelor's and master's degrees in electrical engineering from the Federal University of Minas Gerais (UFMG), Brazil, in 1998 and 2000, respectively, and his PhD degree in computer science from the same university in 2003. He received the Gold Medal Award from the Engineering School of UFMG for the first place among the electrical engineer students in 1998. From 11/2000 to 05/2003, he was a visiting scientist at the GRASP Laboratory of the University of Pennsylvania, USA. He was also a visiting scholar at The Robotics Institute of Carnegie Mellon University, USA, from 08/2015 to 07/2016. From 07/2004 to 07/2018, he was a full-time professor at the Electrical Engineering Department of UFMG. He is currently a Professor at the Department of Mechanical, Materials and Aerospace Engineering of West Virgina University, USA, where he is the director of the Field and Aerial Robotics (FARO) Laboratory. His research interests include field robotics, robot motion planning, robot perception, and autonomous vehicles development. He is a Senior Member of the IEEE.