Journal of Computational Physics 509 (2024) 113050

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

Efficient and scalable path-planning algorithms for curvature constrained motion in the Hamilton-Jacobi formulation

Christian Parkinson *, Isabelle Boyle

Department of Mathematics, University of Arizona, 617 N. Santa Rita Ave., Tucson, AZ, 85721, USA

ARTICLE INFO

Keywords:
Optimal path-planning
Curvature constrained motion
Hamilton-Jacobi equations
Hopf-Lax formulas
Dynamic programming

ABSTRACT

We present a partial-differential-equation-based optimal path-planning framework for curvature constrained motion, with application to vehicles in 2- and 3-spatial-dimensions. This formulation relies on optimal control theory, dynamic programming, and Hamilton-Jacobi-Bellman equations. We develop efficient and scalable algorithms for solutions of high dimensional Hamilton-Jacobi equations which can solve these types of path-planning problems efficiently, even in high dimensions, while maintaining the Hamilton-Jacobi formulation. Because our method is rooted in optimal control theory and has no black box components, it has solid interpretability, and thus averts the tradeoff between interpretability and efficiency for high-dimensional path-planning problems. We demonstrate our method with several examples.

1. Introduction

In this manuscript, we develop a Hamilton-Jacobi partial differential equation (PDE) based method for optimal trajectory generation, with special application to so-called Dubins vehicles which exhibit curvature constrained motion. Specifically, we study kinematic models for simple cars, airplanes, and submarines.

Curvature constrained motion was first considered by Dubins [1] who considered a simple vehicle which could only move forward. The model was extended by Reeds and Shepp to a car which could move forward and backward [2]. In both cases, the strategy was to decompose paths into straight line segments and arcs of circles and analyze which combinations could be optimal. Later work in this direction was devoted to adding obstacles [3], and developing algorithms which can produce approximately optimal paths which are robust to perturbation [4].

To the authors' knowledge the problem was first analyzed using dynamic programming and PDE by Takei, Tsai and others [5,6]. Besides their work, there is a strong precedent in the literature for control theoretic, PDE-based optimal path-planning in a number of applications [7–14]. Trajectory generation methods which are rooted in PDE have the advantage that they are easy to implement, entirely interpretable, and can provide theoretical guarantees regarding optimality, robustness, and other concerns. This is to distinguish them from sampling and learning based algorithms (for example [15–19]) which often sacrifice interpretability for efficiency. The main drawbacks of the PDE-based methods are the lack of efficiency and scalability. Because these methods typically rely on discretizing a domain and approximating a solution to a Hamilton-Jacobi PDE, they can be inefficient even for relatively low-dimensional problems, and intractable for motion planning problems whose state space is more than three dimensions. While some effort has been made to improve efficiency of grid-based methods through parallelization [20,21], other recent work has been

* Corresponding author.
E-mail address: chparkin@math.arizona.edu (C. Parkinson).

https://doi.org/10.1016/j.jcp.2024.113050
Received 24 April 2023; Received in revised form 15 April 2024; Accepted 24 April 2024

Available online 27 April 2024

devoted to developing grid-free numerical methods based on Hopf-Lax formulas which can approximate solutions to Hamilton-Jacobi equations efficiently even in high-dimensions [22–24].

The goal of this manuscript is to develop algorithms for trajectory generation which are (1) interpretable due to the theoretical underpinnings provided by PDEs and optimal control theory, and the absence of any black box components, and (2) efficient enough to be real-time applicable, even in problems with high-dimensional state spaces. Specifically, we present an optimal path-planning method for simple vehicles which maintains the Hamilton-Jacobi PDE formulation, but does not rely on spatial discretization and is thus efficient and scalable. This requires a reformulation of the standard control theoretic minimal-time path-planning problem analyzed in [5,6,10–14]. Our method represents a significant step toward fully interpretable PDE-based motion-planning algorithms which are real-time applicable. While the specific focus in this manuscript is Dubins' type vehicles, with small modifications, it is likely that our methods could be adapted to more complex and higher-dimensional models of motion as well.

The paper is laid out as follows. In section 2, we present the basic dynamic programming and PDE-based path-planning framework that we use, and introduce models for simple Dubins' type vehicles. In section 3, we discuss the numerical methods which we use to solve the requisite PDEs and generate optimal trajectories, and their specific application to our problem. In section 4, we present results of our simulations. We conclude with a brief discussion of our method and potential future research directions in section 5.

2. Modeling

In this section, we derive the PDE-based path-planning framework which we use. In particular, because we will solve these PDEs by translating them into optimization problems as described in section 3, it will be most convenient if we can avoid a formulation which requires boundary conditions, which would translate into difficult constraints in the optimization. Because of this, we opt for a level-set-type formulation in the vein of [7–9], as opposed to the control theoretic approach of [5,6,10–14]. These approaches are compatible, but different in philosophy. We compare and contrast them in section 2.4.

The level-set method is a general method for modeling contours (or more generally, hypersurfaces of codimension 1) which evolve with prescribed velocity depending on the ambient space and properties inherent to the contour itself [25,26]. The basic strategy is to model the contour as the zero level set of an auxiliary function $u : \mathbb{R}^n \times [0,\infty) \to \mathbb{R}$ and derive a PDE which u satisfies. As u evolves according to the PDE, the zero level set of $u(\cdot,t)$ evolves, affecting the level set flow. A general, first-order level set equation has the form

$$u_t + H(x, \nabla u, t) = 0$$

for some *Hamiltonian* function $H(x,p,t)$ which is homogeneous of degree 1 in the variable p. The level set function u does not need to have any physical meaning, though in many cases (as in ours), level set equations are seen to arise as Hamilton-Jacobi-Bellman equations for feedback control problems where u is a value function.

We demonstrate this with a brief and formal derivation. Given a time-horizon $T > 0$, a starting location $x_0 \in \mathbb{R}^d$, a desired-ending location $x_f \in \mathbb{R}^d$, and a function $f : \mathbb{R}^d \times [0,T] \times \mathbb{R}^m \to \mathbb{R}^d$, we consider trajectories $\boldsymbol{x} : [0,T] \to \mathbb{R}^d$ satisfying

$$\begin{aligned} &\dot{\boldsymbol{x}} = f(\boldsymbol{x}, t, \boldsymbol{\alpha}), \qquad 0 < t \le T, \\ &\boldsymbol{x}(0) = x_0. \end{aligned} \tag{1}$$

Here $\boldsymbol{\alpha}(\cdot)$ is a control map, taking values in some admissible control set $A \subset \mathbb{R}^m$. The goal is to choose $\boldsymbol{\alpha}(\cdot)$ so as to steer the trajectory as close as possible to x_f by time T. Accordingly, we define the cost functional

$$C[\boldsymbol{x}(\cdot)] = \frac{1}{2}\left|\boldsymbol{x}(T) - x_f\right|^2,$$

and we would like to solve the optimization problem

$$\inf_{\boldsymbol{\alpha} \in \mathcal{A}} C[\boldsymbol{x}(\cdot)]$$

where $\mathcal{A} = \{\boldsymbol{\alpha} : [0,T] \to A \ : \ \boldsymbol{\alpha} \text{ measurable}\}$. Other choices of cost function are possible, but so as to serve as some measure of distance, they should be zero for any path which ends at x_f, and increase as $|\boldsymbol{x}(T) - x_f|$ increases.

For $x \in \mathbb{R}^d$ and $t \in [0,T)$, we define the value function

$$u(x,t) = \inf_{\boldsymbol{\alpha} \in \mathcal{A}} C_{x,t}[\boldsymbol{x}(\cdot)], \tag{2}$$

where $C_{x,t}$ denotes the same functional restricted to trajectories $\boldsymbol{x}(\cdot)$ such that $\boldsymbol{x}(t) = x$. This value function denotes the minimum square distance to the desired endpoint that one can achieve if they are sitting at point x at time t. In our case, because there is no running cost along the trajectory, the dynamic programming principle [27] states that the value function is constant along an optimal trajectory. That is, for $\delta > 0$,

$$u(\boldsymbol{x}(t), t) = \inf_{\boldsymbol{\alpha}} \{u(\boldsymbol{x}(t+\delta), t+\delta)\},$$

where the infimum is taken with respect to the values of $\boldsymbol{\alpha}(s)$ for $s \in [t, t+\delta)$. If u is smooth, we rearrange, divide by δ and send $\delta \to 0^+$ to see

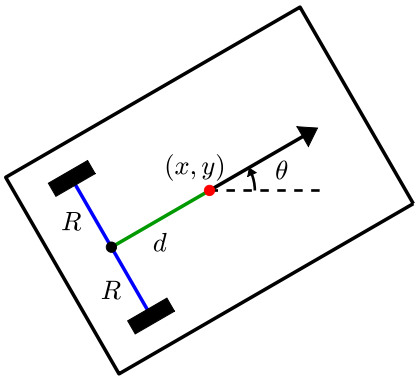

Fig. 1. A simple self-driving car. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

$$\inf_{\alpha\in A}\{u_t + \langle \dot{\boldsymbol{x}}, \nabla u\rangle\} = 0.$$

We can use the equation of motion (1) to replace $\dot{\boldsymbol{x}}$. Further, at time $t = T$, there is no remaining time to travel, so the value is simply the exit cost. Thus we arrive at a terminal-valued Hamilton-Jacobi-Bellman (HJB) equation

$$\begin{aligned} &u_t + \inf_{\alpha\in A}\{\langle f(x,t,\alpha), \nabla u\rangle\} = 0, \\ &u(x,T) = \frac{1}{2}\left|x - x_f\right|^2. \end{aligned} \tag{3}$$

The function $H : \mathbb{R}^d \times \mathbb{R}^d \times [0,T] \to \mathbb{R}$ defined by

$$H(x,p,t) = \inf_{\alpha\in A}\{\langle f(x,t,\alpha), p\rangle\} \tag{4}$$

is the *Hamiltonian*. These computations are formal. There is no reason to believe that u will be smooth, but under very mild conditions on f, u will be the Lipschitz continuous viscosity solution of (3) [28–30]. Assuming the viscosity solution to (3) is known, one resolves the optimal control map via

$$\boldsymbol{\alpha}^*(x,t) = \arg\min_{\alpha\in A}\langle f(x,t,a), \nabla u(x,t)\rangle.$$

Again, pending mild assumptions on f, this optimization problem has a unique solution whenever $\nabla u(x,t)$ exists, which is true for almost every (x,t) when u is Lipschitz continuous. Finally, one can then resolve the optimal trajectory by integrating

$$\begin{aligned} &\dot{\boldsymbol{x}} = f(\boldsymbol{x}, t, \boldsymbol{\alpha}^*(\boldsymbol{x},t)), \qquad 0 < t \le T, \\ &\boldsymbol{x}(0) = x_0, \end{aligned} \tag{5}$$

or using a semi-Lagrangian method as described in [6,11,14].

Because we will solve (3) using Hopf-Lax time formulas (and because it is more comfortable for those familiar with PDE), we make the substitution $t \mapsto T - t$, to arrive at an initial value Hamilton-Jacobi-Bellman equation:

$$\begin{aligned} &u_t - H(x, \nabla u, T-t) = 0, \\ &u(x,0) = \frac{1}{2}\left|x - x_f\right|^2. \end{aligned} \tag{6}$$

For notational convenience, we will still refer to the solution of (6) as u, though it is a time-inverted version of the solution of (3). Also, the Hamiltonians we are interested in below are time-independent (unless moving obstacles are introduced, but these are accounted for separately), so for brevity, we suppress the time-dependence and henceforth write $H = H(x,p)$.

In the ensuing subsections, we use this basic framework to develop optimal path-planning algorithms for Dubins vehicles in 2- and 3-dimensions. This 2-dimensional model for kinematic motion goes back to Dubins [1] and Reeds and Shepp [2]. The 3-dimensional versions still serve as simple models for unmanned airplanes [31,32] and seacrafts [33] today. We note that while our inspiration is self-driving vehicles, curvature constrained motion has other applications such as maneuvering of bevel tipped needles through biological tissue [34].

2.1. Dubins car

We consider a simple rectangular car as pictured in Fig. 1. We let $(x,y) \in \mathbb{R}^2$ denote the center of mass of the car and $\theta \in [0, 2\pi]$ denote the orientation, measured counterclockwise from the horizontal. We refer to (x,y,θ) as the *configuration* of the car. The car

has a rear axle of length $2R$ and distance d from the center of mass to the center of the rear axle. The motion of the car is subject to the nonholonomic constraint

$$\dot{\boldsymbol{y}}\cos\boldsymbol{\theta} - \dot{\boldsymbol{x}}\sin\boldsymbol{\theta} = d\dot{\boldsymbol{\theta}} \tag{7}$$

which specifies that movement occurs tangential to the rear wheels. Motion is also constrained by a maximum angular velocity $|\dot{\theta}| \leq W$ (or equivalently a minimum turning radius $\rho = 1/W$). The kinematics for the car are given by:

$$\begin{aligned}
\dot{\boldsymbol{x}} &= \boldsymbol{v}\cos\boldsymbol{\theta} - \boldsymbol{\omega} W d\sin\boldsymbol{\theta},\\
\dot{\boldsymbol{y}} &= \boldsymbol{v}\sin\boldsymbol{\theta} + \boldsymbol{\omega} W d\cos\boldsymbol{\theta},\\
\dot{\boldsymbol{\theta}} &= \boldsymbol{\omega} W.
\end{aligned} \tag{8}$$

Here, $\boldsymbol{v}(\cdot), \boldsymbol{\omega}(\cdot) \in [-1, 1]$ are normalized control variables representing tangential and angular velocity, respectively.

Given a desired final configuration (x_f, y_f, θ_f) for the car, we can insert these dynamics into the above derivation so that the (time-reversed) optimal remaining distance function $u(x, y, \theta, t)$ for this problem satisfies the Hamilton-Jacobi-Bellman equation

$$u_t - \inf_{v,\omega}\left\{u_x(v\cos\theta - \omega W d\sin\theta) + u_y(v\sin\theta + \omega W d\cos\theta) + \omega W u_\theta\right\} = 0.$$

Rearranging to isolate v, ω, we have

$$u_t - \inf_{v,\omega}\left\{v(u_x\cos\theta + u_y\sin\theta) + \omega W d(-u_x\sin\theta + u_y\cos\theta + \tfrac{1}{d}u_\theta)\right\} = 0.$$

We recall that the control variables v, ω are normalized: $v, \omega \in [-1, 1]$. Thus since the above infimum is linear in v and ω and their dependence is decoupled, the infimum for each control variable must occur at one of the endpoints. Assuming $u(x, y, \theta, t)$ is known, the optimal controls are given by

$$\begin{aligned}
v^*(x, y, \theta, t) &= -\text{sign}(u_x(x, y, \theta, t)\cos\theta + u_y(x, y, \theta, t)\sin\theta),\\
\omega^*(x, y, \theta, t) &= -\text{sign}(-u_x(x, y, \theta, t)\sin\theta + u_y(x, y, \theta, t)\cos\theta + \tfrac{1}{d}u_\theta),
\end{aligned} \tag{9}$$

and

$$\begin{aligned}
&u_t + |u_x\cos\theta + u_y\sin\theta| + W d|-u_x\sin\theta + u_y\cos\theta + \tfrac{1}{d}u_\theta| = 0,\\
&u(x, y, \theta, 0) = \frac{1}{2}|(x, y, \theta) - (x_f, y_f, \theta_f)|^2.
\end{aligned} \tag{10}$$

As written, one can specify a final location *and* orientation that the car must achieve (hence the inclusion of θ_f in the initial condition). If one only cares to specify a final location, this can be omitted, and the initial condition can read $u(x, y, \theta, 0) = \frac{1}{2}\left|(x, y) - (x_f, y_f)\right|^2$. A similar observation applies for the 3-dimensional models below. When modeling the car as a point mass (i.e. setting $d = 0$), (10) simplifies to

$$u_t + |u_x\cos\theta + u_y\sin\theta| + W|u_\theta| = 0. \tag{11}$$

We will deal with this latter formulation (i.e., $d = 0$) so that we are neglecting the actual shape of the vehicle as in [5,6]. Note that, as presented, we have not yet accounted for obstacles. It is argued in [10,11] that it is easier to account for obstacles when one does not simplify the car to a point mass, but the formulation is slightly different in those papers, and we will need to make special considerations for obstacles regardless. We discuss this further in section 2.4. In the next two subsections, we present two higher dimensional generalizations of this model.

2.2. Dubins airplane

We generalize to the case of a simple airplane by introducing a third spatial variable to the dynamics. As a result the autonomous vehicle operates in the coordinates (x, y, z, θ), rather than just (x, y, θ). This is achieved by adding in a fourth kinematic equation, which applies a restriction to the motion in the z direction akin to that imposed on θ. For the sake of this model, the constraint imposed on z is decoupled from the constraint on the planar motion, in a manner similar to that of an airplane so we call this the *Dubins Airplane* model, though, as in the case of the car, we are simplifying the dynamics compared to that of a real airplane. We once again consider the vehicle to be a point mass. This leads to the kinematic equations

$$\begin{aligned}
\dot{\boldsymbol{x}} &= \boldsymbol{v}\cos\boldsymbol{\theta},\\
\dot{\boldsymbol{y}} &= \boldsymbol{v}\sin\boldsymbol{\theta},\\
\dot{\boldsymbol{z}} &= \boldsymbol{\omega_z} W_z,\\
\dot{\boldsymbol{\theta}} &= \boldsymbol{\omega_{xy}} W_{xy},
\end{aligned} \tag{12}$$

where, once again, we enforce $\boldsymbol{v}(\cdot), \boldsymbol{\omega}_{xy}(\cdot), \boldsymbol{\omega}_z(\cdot) \in [-1, 1]$. In (12), $W_{xy} > 0$ is a constraint on the angular velocity in the xy-plane, and $W_z > 0$ is a constraint on the angular velocity in the vertical direction.

Similar reasoning will lead us to the Hamilton-Jacobi-Bellman equation

$$\begin{aligned} &u_t + |u_x \cos\theta + u_y \sin\theta| + W_z|u_z| + W_{xy}|u_\theta| = 0, \\ &u(x, y, z, \theta, 0) = \frac{1}{2}|(x, y, z, \theta) - (x_f, y_f, z_f, \theta_f)|^2. \end{aligned} \tag{13}$$

To truly model an airplane in flight, it makes sense to restrict to unidirectional velocity: $\boldsymbol{v}(\cdot) = 1$. In this case, the equation simplifies to

$$u_t - u_x \cos\theta - u_y \sin\theta + W_z|u_z| + W_{xy}|u_\theta| = 0. \tag{14}$$

As mentioned above, this formulation does not yet account for the presence of obstacles, which will be discussed in section 2.4. One could also account for finer-scale modeling concerns such as minimum cruising velocity, but for our purposes, the formulation is as presented.

2.3. Dubins submarine

As an alternative to Dubins Airplane, we can model the problem in three dimensions by enforcing a total curvature constraint, rather than decoupling the constraints on the planar and vertical motion. We dub this model the *Dubins Submarine*, though again, we are neglecting many of the dynamics of a real submarine. In this case, we let (x, y, z) represent the center of mass of the vehicle. The orientation of the vehicle is now represented by a pair of angles: $\theta \in [0, 2\pi]$ represents the angular orientation on the xy-plane (the azimuthal angle), and $\phi \in [0, \pi]$ is the angle of inclination, with $\phi = 0$ pointing straight up the z-axis and $\phi = \pi$ pointing straight down the z-axis. In this case, the equations of motion are

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{v} \cos\theta \sin\varphi, \\ \dot{\boldsymbol{y}} &= \boldsymbol{v} \sin\theta \sin\varphi, \\ \dot{\boldsymbol{z}} &= \boldsymbol{v} \cos\varphi, \\ \dot{\boldsymbol{\theta}} &= \boldsymbol{\omega}_1 W, \\ \dot{\boldsymbol{\varphi}} &= \boldsymbol{\omega}_2 W, \end{aligned} \tag{15}$$

where $\boldsymbol{v}(\cdot)$ is the normalized control variable representing tangential velocity, $\boldsymbol{\omega}_1(\cdot), \boldsymbol{\omega}_2(\cdot)$ are normalized control variables for angular velocity, and $W > 0$ is a constraint on the curvature of the path. The magnitude of the curvature of a path obeying (15) is given by

$$\kappa(t) := \left| \frac{d}{dt} \frac{(\dot{\boldsymbol{x}}, \dot{\boldsymbol{y}}, \dot{\boldsymbol{z}})}{|(\dot{\boldsymbol{x}}, \dot{\boldsymbol{y}}, \dot{\boldsymbol{z}})|} \right| = W \sqrt{\boldsymbol{\omega}_1^2 \sin^2(\boldsymbol{\varphi}) + \boldsymbol{\omega}_2^2},$$

which leads to a constraint on $\boldsymbol{\omega}_1, \boldsymbol{\omega}_2$ of the form

$$\sqrt{\boldsymbol{\omega}_1^2 \sin^2(\boldsymbol{\varphi}) + \boldsymbol{\omega}_2^2} \leq 1. \tag{16}$$

Going through the same derivation for the Hamilton-Jacobi-Bellman equation (and having made the substitution $t \mapsto T - t$ to reverse time), we arrive at

$$u_t - \inf_{v, \omega_1, \omega_2} \left\{ v(u_x \cos\theta \sin\varphi + u_y \sin\theta \sin\varphi + u_z \cos\varphi) + \omega_1 u_\theta + \omega_2 u_\varphi \right\} = 0.$$

Again, the minimization in v is decoupled from that of (ω_1, ω_2), so it is resolved exactly as before. For the minimization in (ω_1, ω_2), assuming that $\sin\varphi \neq 0$, we write

$$\inf_{\omega_1, \omega_2} \{\omega_1 u_\theta + \omega_2 u_\varphi\} = \inf_{\omega_1, \omega_2} \left\{ (\omega_1 \sin\varphi) \left(\frac{u_\theta}{\sin\varphi} \right) + \omega_2 u_\varphi \right\}.$$

Recall, (ω_1, ω_2) are constrained by (16) so that $(\omega_1 \sin\varphi, \omega_2)$ is in the unit circle. Thus,

$$\inf_{\omega_1, \omega_2} \left\{ (\omega_1 \sin\varphi) \left(\frac{u_\theta}{\sin\varphi} \right) + \omega_2 u_\varphi \right\} = -\sqrt{\frac{u_\theta^2}{\sin^2\varphi} + u_\varphi^2}.$$

Thus the HJB equation for this model is

$$\begin{aligned} &u_t + |u_x \cos\theta \sin\varphi + u_y \sin\theta \sin\varphi + u_z \cos\varphi| + W \sqrt{\frac{u_\theta^2}{\sin^2\varphi} + u_\varphi^2} = 0, \\ &u(x, y, z, \theta, \varphi, 0) = \frac{1}{2}|(x, y, z, \theta, \varphi) - (x_f, y_f, z_f, \theta_f, \varphi_f)|^2, \end{aligned} \tag{17}$$

for $(x, y, z, \theta, \varphi, t) \in \mathbb{R}^3 \times [0, 2\pi] \times (0, \pi) \times [0, T]$. When $\varphi = 0$ or $\varphi = \pi$, the submarine is oriented directly upward or downward respectively. In this case, to change the orientation, one must modify ϕ (i.e., ω_1 can take any value, but it will not affect the orientation; only ω_2 affects the orientation). One could specifically account for this in the above derivation if desired. For computational purposes, it suffices to replace the $\sin^2 \varphi$ in the denominator in (17) with $(\sin^2 \varphi + \varepsilon)$ where ε is only a few orders of magnitude larger than machine precision. This will circuit any division by zero without materially affecting results. For our purposes, we use $\varepsilon = 10^{-10}$.

In the ensuing sections, we discuss the formulation of these models in the presences of impassable obstacles, and then move on to develop numerical methods for approximating these equations and generating optimal trajectories.

2.4. Level set vs. optimal control formulation: the time horizon and obstacles

In all the preceding derivations, we implicitly assume that the vehicles are moving in free space. Here we discuss how one may account for obstacles which impede the vehicles. We also discuss the role of the time horizon T, because both the manner in which we include obstacles and the time horizon arise as a consequence of modeling the problem using level set equations, as opposed to what is perhaps a more natural control-theoretic model. To reiterate, we use level set equations because they are amenable to the available numerical methods. Specifically, using the level set formulation allows us to derive an HJB equation which can resolve time-optimal paths *without* using any boundary conditions. This is desirable because in section 3, we describe how one can solve the HJB equation by reframing it as an optimization problem for which it is difficult to incorporate boundary conditions. However, this decision has ramifications that deserve some discussion.

Reverting back to the general framework from the beginning of section 2, we recall that for our models, the value function $u(x, t)$ encodes the minimum achievable distance to the desired endpoint given that the vehicle is in position x at time t. An alternative approach to path-planning, like that used in [5,6,10–14], is to define the value function $\tau(x, t)$ to be the optimal remaining travel time to the desired endpoint given that the vehicle is in position x at time t. In the latter case, by definition, $\tau(x, t) = +\infty$ whenever there is no admissible path for a car which is at x at time t which can steer the car to the desired ending point before hitting the time horizon T, and thus $\tau(x, t)$ is finite if and only if there is an admissible path to the ending point in the allotted time. When there are no obstacles (or stationary obstacles), once $\tau(x, t)$ becomes finite, it will remain constant. For example, if the optimal travel time from x to x_f is 3 units of time, then $\tau(x, 1) = +\infty$, $\tau(x, 3) = 3$ and $\tau(x, 10) = 3$. In this case, one can essentially eliminate the time horizon, by simply taking T large enough that $\tau(x, T) < +\infty$ for all points x that one cares about. Specifically, with mild assumptions on the dynamics (so that admissible paths are possible from any given point), for any compact spatial domain Ω, there is a time horizon $T^* = T^*(\Omega) > 0$, such that $\tau(x, T^*) < +\infty$ for all $x \in \Omega$, and choosing any $T \geq T^*$ as the time horizon for the problem will yield the same results. In this way, it does not matter that one optimally chooses the time horizon: as long as it is large enough, one can resolve the time-optimal path from x to x_f and this path will be independent of the time horizon T. Moreover, with this modeling choice, whenever $\tau(x, T)$ is differentiable, one can uniquely determine the optimal control values and thus there is a unique time optimal path from x to x_f.

The interpretation of our model is slightly different since $u(x, t)$ denotes some measure of optimal distance to the endpoint. Like $\tau(x, t)$, given stationary obstacles and mild conditions on the dynamics, this function will become constant in finite time: for any compact spatial domain Ω, there is a time horizon $T^* = T^*(\Omega) > 0$ such that $u(x, T^*) = 0$, and then $u(x, t) = 0$ for all $t > T^*$. In essence, this says that given enough time, there are paths which can reach the desired final point. However, in this case, optimality of a trajectory is judged solely in view of "distance to the desired endpoint" so any path that reaches the final endpoint is equally optimal. Because of this, choosing the time horizon optimally becomes important. Specifically, given a point x, define $t_x = \inf\{t > 0 : u(x, t) = 0\}$. If $T > t_x$ (and if the vehicle satisfies a small-time local controllability condition [35]), then there will be infinitely many "optimal" paths from x to x_f since there is more time that needed in order to reach the final point. In this case, we would like the optimal path which requires the minimal time to traverse. In order to resolve the minimal time path beginning from a point x, we need to actually use t_x as the time horizon.

This causes some difficulty because resolving t_x for a given point x requires reformulating the problem in terms of $\tau(x, t)$ as discussed above. Empirically, this affects the numerical methods as well, as we discuss in section 4. Before this, we address one further modeling concern.

One last modeling concern is the inclusion of obstacles. Intuitively, any path the intersects with an obstacle at any time should be considered illegal and assigned infinite cost, so that it is never the optimal path. To model this, assume that a vehicle is navigating a domain $\Omega \subset \mathbb{R}^n$ which, at any time $t > 0$ is disjointly segmented into free space and obstacles, $\Omega = \Omega_{\text{free}}(t) \sqcup \Omega_{\text{obs}}(t)$.

Using the "travel-time" formulation of [5,6,10,11] as discussed above, one assigns infinite cost to paths that intersect with obstacles by setting $\tau(x, t) = +\infty$ for any points $(x, t) \in \Omega \times (0, T]$ such that $x \in \Omega_{\text{obs}}(t)$. This becomes a crucial boundary condition in the HJB equation for the travel time function τ. However, as described in section 3, we hope to resolve the solution to our HJB equations using optimization routines which are much easier to implement when the HJB equations are free of boundary conditions. This is one of the primary motivations for using the level set formulation. In the level set formulation, one incorporates obstacles not by enforcing a boundary condition, but by setting velocity to zero in the obstacles. That is, define $O(x, t)$ to be the indicator function of the free space at time t:

$$O(x, t) = \mathbf{1}_{\Omega_{\text{free}}(t)}(x) = \begin{cases} 0, & x \in \Omega_{\text{obs}}(t), \\ 1, & x \in \Omega_{\text{free}}(t), \end{cases} \qquad (x, t) \in \mathbb{R}^n \times [0, T]. \tag{18}$$

To set velocity to zero in the obstacles, we then simply multiply the Hamiltonians in (11), (14), (17) (or generally in (6)) by $O(x,t)$. For example, the version of (11) that we actually solve is

$$u_t + O(x,y,t)\left[|u_x\cos\theta + u_y\sin\theta| + W|u_\theta|\right] = 0. \tag{19}$$

Having done this, any path which enters an obstacle will be forced to stop, get stuck, and fail to reach the end point, which will never be optimal. In this way, we have incorporated obstacles without using any boundary conditions. This raises other numerical issues (for example, how to efficiently determine whether a given x lies in a obstacle at time t) which we discuss further in the ensuing section. This manner of incorporating obstacles is akin to that of [6–8].

3. Numerical methods

To this point, the standard approach to approximating solutions to HJB equations in applications like this has been to use finite difference schemes such as fast-sweeping schemes [36–38], fast-marching schemes [39,40], and their generalizations [41,42]. These are used, for example, in [5,6,8,10,11]. These methods are easy to implement and can be adapted for high-order accuracy, but because they are grid-based, their computational complexity scales exponentially with the dimension of the domain, meaning that they are only feasible in low dimension. The authors of [20,21] present parallelizable fast-sweeping schemes which can approximate the solutions of 3- and 4-dimensional HJB-type equations in a matter of minutes (or in the case of the Eikonal equation, a matter of seconds). However, because the model for the Dubins' submarine is 5-dimensional, and we would like to develop methods which scale to even higher dimension, we opt for a grid-free scheme.

Recent numerical methods attempt to break the *curse of dimensionality* [27] by resolving the solution to certain Hamilton-Jacobi equations at individual points using variational Hopf-Lax type formulas [22,23]. In particular, given a Hamiltonian $H : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ which is convex in its second argument and C^2-smooth, and an initial data function $g : \mathbb{R}^d \to \mathbb{R}$ which is convex and coercive, the authors of [23] conjecture that the solution to the state-dependent Hamilton-Jacobi equation

$$u_t + H(x,\nabla u) = 0, \qquad u(x,0) = g(x), \tag{20}$$

is given by the generalized Hopf-Lax formula,

$$\begin{aligned} u(x,t) &= \min_{v\in\mathbb{R}^d}\left\{ g(\boldsymbol{x}(0)) + \int_0^t \langle \boldsymbol{p}(s), \nabla_p H(\boldsymbol{x}(s),\boldsymbol{p}(s)) - H(\boldsymbol{x}(s),\boldsymbol{p}(s))\rangle ds \right\} \\ &\text{subject to the Hamiltonian dynamics,} \\ &\dot{\boldsymbol{x}}(s) = \quad \nabla_p H(\boldsymbol{x}(s),\boldsymbol{p}(s)), \qquad \boldsymbol{x}(t) = x, \\ &\dot{\boldsymbol{p}}(s) = -\nabla_x H(\boldsymbol{x}(s),\boldsymbol{p}(s)), \qquad \boldsymbol{p}(t) = v. \end{aligned} \tag{21}$$

While the conjecture requires the Hamiltonian to be C^2, the authors of [23] give specific examples which violate this assumption for which the conjecture still holds, so it may hold even more broadly.

Similarly, beginning from (2), we can derive a conjectured saddle-point formula to approximate the value function. To do so, we discretize the time interval $t = t_0 < t_1 < \cdots < t_N = T$ and let x_j be discrete approximations to $\boldsymbol{x}(t_j)$ for $j = 0,1,\ldots,N$. For simplicity, we will always assume a uniform discretization $t_j = t + j\delta$, $j = 0,1,\ldots,N$ where $\delta = \frac{T-t}{N}$, though this is not strictly necessary. Then we see

$$u(x,t) = \inf_{\boldsymbol{x}(\cdot),\boldsymbol{\alpha}(\cdot)} \left\{ g(x(T)) \,:\, \boldsymbol{x}(t) = x, \dot{\boldsymbol{x}} = f(\boldsymbol{x},s,\boldsymbol{\alpha}), s\in(t,T]\right\}, \tag{22}$$

where $g(x) = \frac{1}{2}\left|x - x_f\right|^2$. Formally, discretizing the constraint using a backward Euler scheme, letting $\alpha_j = \boldsymbol{\alpha}(t_j)$ and $f_j = f(x_j,t_j,\alpha_j)$, and introducing Lagrange multipliers p_j, we have

$$\begin{aligned} u(x,t) &\approx \inf_{x_j,\alpha_j}\left\{ g(x_N) \,:\, x_0 = x, x_j = x_{j-1} + \delta f_j,\ 1\le j\le N\right\} \\ &= \inf_{x_j,\alpha_j}\sup_{p_j}\left\{ g(x_N) + \sum_{j=1}^N \left\langle p_j, \delta f_j + x_{j-1} - x_j\right\rangle \right\} \\ &= \inf_{x_j,\alpha_j}\sup_{p_j}\left\{ g(x_N) + \sum_{j=1}^N \left\langle p_j, x_{j-1} - x_j\right\rangle + \delta\sum_{j=0}^{N-1}\left\langle p_j, f_j\right\rangle \right\}. \end{aligned} \tag{23}$$

From here, we interchange the infimum over α_j with the supremum over p_j and write

Algorithm 1 Splitting Method for Solving (21)

Given a point $(x,t) \in \mathbb{R}^d \times (0,T]$, a Hamiltonian H, an initial data function g, a time-discretization count N, a max iteration count K, an error tolerance TOL, and relaxation parameters $\sigma, \tau, \kappa > 0$, we resolve the minimization problem (21) as follows.

Set $x_N^1 = x$, $p_0^1 = 0$ and $\delta = t/N$. Initialize $x_0^1, x_1^1, \dots, x_{N-1}^1, p_1^1, \dots, p_N^1$ randomly, and set $z_j^1 = x_j^1$ for all $j = 0, 1, \dots, N$.
for $k = 1$ to K **do**
  Set $p_0^{k+1} = 0$
  for $j = 1$ to N **do**
    $p_j^{k+1} = \arg\min_{\tilde{p}} \{\delta\sigma H(x_j^k, \tilde{p}) + \frac{1}{2}\left|\tilde{p} - (p_j^k + \sigma(z_j^k - z_{j-1}^k))\right|_2^2\}$
  end for
  $x_0^{k+1} = \arg\min_{\tilde{x}} \{\tau g(\tilde{x}) + \frac{1}{2}\left|\tilde{x} - (x_0^k + \tau p_1^{k+1})\right|_2^2\}$ (note: $p_0^{k+1} = 0$)
  for $j = 1$ to $N-1$ **do**
    $x_j^{k+1} = \arg\min_{\tilde{x}} \{-\delta\tau H(\tilde{x}, p_j^{k+1}) + \frac{1}{2}\left|\tilde{x} - (x_j^k - \tau(p_j^{k+1} - p_{j+1}^{k+1}))\right|_2^2\}$
  end for
  Set $x_N^{k+1} = x$
  for $j = 0$ to N **do**
    $z_j^{k+1} = x_j^{k+1} + \kappa(x_j^{k+1} - x_j^k)$
  end for
  change $= \max\{\|x^{k+1} - x^k\|, \|p^{k+1} - p^k\|\}$
  if change $<$ TOL **then**
    break
  end if
end for
$u = g(x_0) + \sum_{j=1}^N \langle p_j, x_j - x_{j-1}\rangle - \delta H(x_j, p_j)$
return u; the value of the solution of (20) at the point (x,t)

$$\begin{aligned} u(x,t) &\approx \inf_{x_j} \sup_{p_j} \left\{ g(x_N) + \sum_{j=1}^N \left\langle p_j, x_{j-1} - x_j \right\rangle + \delta \sum_{j=0}^{N-1} \inf_{\alpha_j} \{ \left\langle p_j, f_j \right\rangle \} \right\} \\ &= \inf_{x_j} \sup_{p_j} \left\{ g(x_N) + \sum_{j=1}^N \left\langle p_j, x_{j-1} - x_j \right\rangle + \delta \sum_{j=0}^{N-1} H(x_j, p_j) \} \right\} \end{aligned} \tag{24}$$

Finally, as in (6), making the time-reversing substitution $t \mapsto T - t$ yields

$$u(x,t) \approx \inf_{x_j} \sup_{p_j} \left\{ g(x_0) + \sum_{j=1}^N \left\langle p_j, x_j - x_{j-1} \right\rangle - \delta \sum_{j=0}^{N-1} H(x_j, p_j) \} \right\}. \tag{25}$$

In general, swapping the order of optimization does not yield the same value. However, performing this step provides a connection between this optimization problem and the Hamilton-Jacobi-Bellman equation via the Hamiltonian. In the spirit of the conjectured Hopf-Lax formula (21), the authors of [24] conjecture (and provide solid empirical evidence) that having swapped the order of the optimization, the resulting saddle-point problem still provides an approximation of the solution of the HJB equation. Our results corroborate this. In fact, [24] specifically considers Eikonal-type equations whose Hamiltonians $H(x,p) = V(x)|p|$ share key properties with ours, such as convexity and 1-homogeneity in the second variable and smoothness almost everywhere, so there is precedent to expect this saddle point problem can indeed represent our solutions.

An alternating minimization technique in the spirit of the primal-dual hybrid gradient method [43–45] is proposed by [24] to solve this saddle-point problem. For completeness of our exposition, we reprint this in Algorithm 1 (note: this is a reprint of Algorithm 5 in [24], modified slightly to fit our equations). As this algorithm resolves the values of u (the solution of (20)), it also resolves discrete approximations of the optimal trajectory $\boldsymbol{x}(s)$, and the optimal costate trajectory $\boldsymbol{p}(s)$, which can be seen as a proxy for ∇u along the optimal trajectory.

Assuming a saddle point exists and that the minima are resolved exactly, convergence of the algorithm to a saddle point is guaranteed when $\sigma\tau \leq 0.25$ and $\kappa \in [0,1]$ [24,43]. In our particular implementation, we take $\sigma = 0.5$, $\tau = 0.5$, and $\kappa = 1$. The norm used to determine convergence is not terribly important. If the state space is d-dimensional, so that at each time-step $j = 0, 1 \dots, N$ and iteration k, we have $x_j^k = ((x_1)_j^k, (x_2)_j^k, \dots, (x_d)_j^k)$, we use the norm

$$\|x^{k+1} - x^k\| = \sup_{1 \leq i \leq d, 0 \leq j \leq N} \left| (x_i)_j^{k+1} - (x_i)_j^k \right|$$

and similarly for p. Thus we halt the iteration when no coordinate of either x or p has changed by more than the prescribed TOL value.

Note that at each iteration in Algorithm 1, there are approximately $2N$ optimization problems

$$p_j^{k+1} = \arg\min_{\tilde{p}} \{\delta\sigma H(x_j^k, \tilde{p}) + \frac{1}{2}\left|\tilde{p} - (p_j^k + \sigma(z_j^k - z_{j-1}^k))\right|_2^2\}, \tag{26}$$

$$x_0^{k+1} = \arg\min_{\tilde{x}} \{\tau g(\tilde{x}) + \frac{1}{2}\left|\tilde{x} - (x_0^k + \tau p_1^k))\right|_2^2\}, \tag{27}$$

$$x_j^{k+1} = \arg\min_{\tilde{x}} \{-\delta\tau H(\tilde{x}, p_j^{k+1}) + \frac{1}{2}\left|\tilde{x} - (x_j^k - \tau(p_j^{k+1} - p_{j+1}^{k+1}))\right|_2^2\}, \tag{28}$$

where N is the number of discrete time steps along the path. These are vector valued quantities, with the subscript denoting the time step along the path and the superscript denoting the iteration number. We allow a maximum of $K = 100000$ in our simulations, though with baseline parameter values, the routine usually converged to within a tolerance of $\text{TOL} = 10^{-3}$ within 10000 iterations and never reached the maximal iteration count (we discuss this more specifically in section 4). Even so, this is a fairly large computational burden, so when possible, it behooves one to resolve the optimization problems in Algorithm 1 analytically. When this is not possible, one can use gradient descent or some other comparably simple optimization method. In our applications, the minimization problems in the costate variables p can be resolved exactly, while the minimization problems in the state variables x will need to be approximated in some cases. Empirically, it was observed in [24] (and corroborated in our simulations) that one only needs to very crudely approximate these minimizers; for example, using only a few steps of gradient descent at each iteration. In this manner, the approximation may be very poor at early iterations, but becomes better as the iteration count increases. We describe the specifics of how we resolve the minimization problems from Algorithm 1 in the next subsections.

3.1. Implementation of Algorithm 1 for the Dubins car

Here we describe the implementation of Algorithm 1 for the Dubins car. In (11), we see that the Hamiltonian is

$$H(x, p) = |p_1 \cos(x_3) + p_2 \sin(x_3)| + W|p_3|, \tag{29}$$

where (x_1, x_2, x_3) represent (x, y, θ) respectively and (p_1, p_2, p_3) are proxies for (u_x, u_y, u_θ) respectively.

An iteration in Algorithm 1 begins by resolving the minimization problem (26), which can be done analytically. For completeness, we include the formal derivation of the minimizer here. For this Hamiltonian, the dependence of H on (p_1, p_2) is decoupled from the dependence on p_3, so we can treat these as two separate problems. In what follows, all indexing notation is adapted to MATLAB indexing conventions so that $\tilde{p}_{1:2}$ indicates the first two components of the vector $\tilde{p}$, for example.

To simplify notation, we set

$$\gamma = (\cos((x_3)_j^k), \sin((x_3)_j^k)), \quad \beta = p_j^k + \sigma\left(z_j^k - z_{j-1}^k\right),$$

so that (26) is can be written

$$p_j^{k+1} = \arg\min_{\tilde{p}} \left\{\delta\sigma\left|\gamma^T\tilde{p}_{1:2}\right| + W\left|\tilde{p}_3\right| + \frac{1}{2}\|\tilde{p} - \beta\|_2^2\right\}.$$

Note that, when resolving p_j^{k+1}, γ and β are known quantities which depend on j and k, but we suppress this dependence for ease of notation. Taking the gradient of the function being minimized and setting to zero, we see that the minimizer $\tilde{p}$ satisfies

$$\delta\sigma\frac{\gamma^T\tilde{p}_{1:2}}{\left|\gamma^T\tilde{p}_{1:2}\right|}\gamma + \tilde{p}_{1:2} - \beta_{1:2} = 0, \tag{30}$$

so that

$$\left(\frac{\delta\sigma}{\left|\gamma^T\tilde{p}_{1:2}\right|}\gamma\gamma^T + I\right)\tilde{p}_{1:2} = \beta_{1:2}.$$

Now in order to solve for $\tilde{p}_{1:2}$, we project all vectors along γ and the orthogonal direction to γ by writing,

$$\tilde{p}_{1:2} = r\gamma + s(\beta_{1:2} - (\gamma^T\beta_{1:2})\gamma), \tag{31}$$

for some $r, s \in \mathbb{R}$, and

$$\beta_{1:2} = (\gamma^T\beta_{1:2})\gamma + (\beta_{1:2} - (\gamma^T\beta_{1:2})\gamma).$$

Inserting these in (30) and using $\gamma^T\gamma = 1$ gives

$$\frac{\delta\sigma r}{|r|}\gamma + r\gamma + s(\beta_{1:2} - (\gamma^T\beta_{1:2})\gamma) = (\gamma^T\beta_{1:2})\gamma + (\beta_{1:2} - (\gamma^T\beta_{1:2})\gamma),$$

whereupon we immediately have $s = 1$. Then

$$\left(\frac{\delta\sigma}{|r|} + 1\right)r = \gamma^T\beta_{1:2}$$

shows that r shares a sign with $\gamma^T\beta_{1:2}$, so we can write $r = a(\gamma^T\beta_{1:2})$ for some $a > 0$ and arrive at

$$\left(\frac{\delta\sigma}{a\left|\gamma^T\beta_{1:2}\right|} + 1\right)a(\gamma^T\beta_{1:2}) = \gamma^T\beta_{1:2} \quad \Longrightarrow \quad a = 1 - \frac{\delta\sigma}{\left|\gamma^T\beta_{1:2}\right|}.$$

If this formula yields a negative result, this is a reflection of the fact that the true minimizer $\tilde{p}_{1:2}$ is orthogonal to γ, whereupon this derivation is invalid, since the function being minimized is not differentiable at the minimizer, but this can be accounted for by simply setting $a = 0$. Thus we have

$$a = \max\left\{1 - \frac{\delta\sigma}{|\gamma^T \beta_{1:2}|}, 0\right\}.$$

Finally, plugging this back into (31) gives the final update rule

$$(p_{1:2})_j^{k+1} = \left[\max\left\{0, 1 - \frac{\delta\sigma}{|\gamma^T \beta_{1:2}|}\right\} - 1\right](\gamma^T \beta_{1:2})\gamma + \beta_{1:2}. \tag{32}$$

One can resolve the minimization for p_3 in a similar manner and arrive at

$$(p_3)_j^{k+1} = \max\left\{0, 1 - \frac{\delta\sigma W}{|\beta_3|}\right\}\beta_3. \tag{33}$$

Next we resolve the $k+1$ iterate for the path x. First, we solve (27) with the initial data function $g(x) = \frac{1}{2}\left|x - x_f\right|^2$. Note that these are vector quantities; in a slight abuse of notation we are letting $x = (x_1, x_2, x_3) = (x, y, \theta)$. Then (27) can be written

$$x_0^{k+1} = \arg\min_{\tilde{x}}\left\{\frac{\tau}{2}|\tilde{x} - x_f|^2 + \frac{1}{2}|\tilde{x} - (x_0^k + \tau p_1^{k+1})|^2\right\}.$$

One easily finds that this has minimizer

$$x_0^{k+1} = \frac{x_0^k + \tau(x_f + p_1^{k+1})}{1 + \tau}. \tag{34}$$

To update x_j^{k+1} for $j = 1, \ldots, N-1$, we note that our Hamiltonian $H(x, p)$ given in (29) does not depend on $x_{1:2}$. Because of this, it is trivial to resolve the first and second coordinate in the minimization (28):

$$(x_{1:2})_j^{k+1} = (x_{1:2})_j^k - \tau\left((p_{1:2})_j^{k+1} - (p_{1:2})_{j+1}^{k+1}\right) \tag{35}$$

By contrast, one cannot resolve the third coordinate $(x_3)_j^{k+1}$ of the minimizer in (28) analytically, as the solution to the minimization problem is given in terms of a transcendental equation. Thus we approximate using gradient descent. That is, we set $\theta^* = (x_3)_j^{k+1}$ and perform a few iterations of

$$\theta^* = \theta^* - \eta h'(\theta^*) \tag{36}$$

where $h : \mathbb{R} \to \mathbb{R}$ is defined

$$\begin{aligned} h(\theta) = -\delta\tau|(p_1)_j^{k+1}\cos(\theta) + (p_2)_j^{k+1}\sin(\theta)| \\ + \frac{1}{2}\left(\theta - ((x_3)_j^k - \tau((p_3)_j^{k+1} - (p_3)_{j+1}^{k+1}))\right)^2. \end{aligned} \tag{37}$$

We then assign $(x_3)_j^{k+1} = \theta^*$. Here $\eta > 0$ is the gradient descent rate. For our simulations, we choose $\eta = 0.15$, and performed three steps of gradient descent at each iteration, though the method also worked with other choices.

Finally, we update the z values:

$$z_j^{k+1} = x_j^{k+1} + \kappa(x_j^{k+1} - x_j^k).$$

3.2. Implementation of Algorithm 1 for the Dubins airplane

For the Dubins airplane, we use the Hamiltonian (14) which can be written

$$H(x, p) = -p_1\cos(x_3) - p_2\sin(x_3) + W_z|p_3| + W_{xy}|p_4|.$$

Here our notation is $x = (x_1, x_2, x_3, x_4) = (x, y, z, \theta)$ and $p = (p_1, p_2, p_3, p_4) = (u_x, u_y, u_z, u_\theta)$. The update rules for this Hamiltonian are virtually unchanged from those of section 3.1, since the minimization for $p_{1:2}, p_3, p_4$ can be resolved individually.

In fact, the minimization for $p_{1:2}$ is simpler, since there are no absolute values on the first two terms in this Hamiltonian. Define

$$\gamma = (\cos((x_4)_j^k), \sin((x_4)_j^k)), \quad \beta = p_j^k + \sigma\left(z_j^k - z_{j-1}^k\right).$$

Note that these are analogous to the definitions of γ and β in section 3.1. Then the update rule for $(p_{1:2})_j^{k+1}$ is

$$(p_{1:2})_j^{k+1} = \beta_{1:2} + \delta\sigma\gamma.$$

The updates for the components p_3 and p_4 have the same structure as before, but with their respective constraints on the angular velocities, W_z for z and W_{xy} for θ. That is,

$$(p_3)_j^{k+1} = \max\left\{0, 1 - \frac{\delta\sigma W_z}{|\beta_3|}\right\}\beta_3,$$
$$(p_4)_j^{k+1} = \max\left\{0, 1 - \frac{\delta\sigma W_{xy}}{|\beta_4|}\right\}\beta_4.$$

The updates for the x vector are also similar to those in section 3.1. The update for x_0^{k+1} remains formally the same, though all vectors are four dimensional. The update for the spatial coordinates $x_{1:3}$ is modified only in that in includes the $x_3 = z$ coordinate:

$$(x_{1:3})_j^{k+1} = (x_{1:3})_j^k - \tau\left((p_{1:3})_j^{k+1} - (p_{1:3})_{j+1}^{k+1}\right). \tag{38}$$

The update for $(x_4)_j^{k+1}$ is slightly different (again: simpler). We perform some fixed number of steps of the gradient descent

$$\theta^* = \theta^* - \eta h(\theta^*) \tag{39}$$

where in this case $h : \mathbb{R} \to \mathbb{R}$ is defined

$$\begin{aligned} h(\theta) = \delta\tau\left((p_1)_j^{k+1}\cos(\theta) + (p_2)_j^{k+1}\sin(\theta)\right) \\ + \frac{1}{2}\left(\theta - ((x_4)_j^k - \tau((p_4)_j^{k+1} - (p_4)_{j+1}^{k+1}))\right)^2 \end{aligned} \tag{40}$$

and then assign $(x^4)_j^{k+1} = \theta^*$. The z update is the same as in section 3.1.

3.3. Implementation of Algorithm 1 for the Dubins submarine

The Hamiltonian for the Dubins submarine in (17) can be written

$$H(x,p) = \left|p_1\cos(x_4)\sin(x_5) + p_1\sin(x_4)\sin(x_5) + p_3\cos(x_5)\right| + W\sqrt{A(x_5)p_{4:5}}$$

where the matrix A is given by

$$A(x_5) = \begin{bmatrix} \frac{1}{\sin^2(x_5)} & 0 \\ 0 & 1 \end{bmatrix}.$$

Here our state vector is $x = (x_1, x_2, x_3, x_4, x_5) = (x, y, z, \theta, \varphi)$.

For ease of notation, when resolving (26)-(28) for this Hamiltonian at a time step j on iteration $k+1$, we define

$$\gamma = (\cos((x_4)_j^k)\sin((x_5)_j^k), \sin((x_4)_j^k)\sin((x_5)_j^k), \cos((x_5)_j^k),$$
$$\beta = p_j^k + \sigma(z_j^k - z_{j-1}^k),$$
$$\nu = x_j^k - \tau(p_j^{k+1} - p_{j+1}^{k+1}).$$

Again, we can resolve (26) mostly analytically, by noting that the dependence of $H(x,p)$ on $p_{1:3}$ is decoupled from the dependence on $p_{4:5}$. The update for $p_{1:3}$ is very similarly to the update for $p_{1:2}$ in section 3.1. We find the rule

$$(p_{1:3})_j^{k+1} = M(\gamma^T\beta_{1:3})\gamma + (\beta_{1:3})$$

where the constant M is given by

$$M = \max\left(0, 1 - \frac{\delta\sigma}{|\gamma^T\beta|}\right) - 1.$$

The updates for $p_{4:5}$ are a bit more complicated. Specifically, we need to resolve

$$(p_{4:5})_j^{k+1} = \arg\min_{\tilde{p}}\{\delta\sigma\left|A((x_5)_j^k)\tilde{p}\right| + \frac{1}{2}|\tilde{p} - \beta_{4:5}|^2\}.$$

This is an evaluation of the proximal operator of the function $f(\tilde{p}) = |A\tilde{p}|$. Using methods similar to those in section 3.1, one can resolve this almost analytically. The solution is

$$(p_{4:5})_j^{k+1} = \begin{cases} 0, & a^* \le 0, \\ \left(I + \frac{\delta\sigma W}{a^*}A((x_5)_j^k)^2\right)^{-1}\beta_{4:5}, & \alpha^* > 0 \end{cases} \tag{41}$$

where I is the 2×2 identity matrix and α^* is the unique root of the function

$$g(a) = \frac{\frac{1}{\sin^4((x_5)_j^k)}\beta_4^2}{\left(\frac{\delta\sigma W}{\sin^4((x_5)_j^k)} + a\right)^2} + \frac{\beta_5^2}{(\delta\sigma W + a)^2} - 1. \tag{42}$$

This function is decreasing, approaches $+\infty$ as $a \searrow -\delta\sigma W$, and approaches -1 as $a \nearrow +\infty$, so it has a unique root $a^* \in [-\sigma\delta W, \infty)$. We find this root a^* using a bisection method. First, we check the values $g(-\delta\sigma W + 100\ell)$, for $\ell = 1, 2, 3, \ldots$ to determine the interval of length 100 which contains the root, and then apply the standard bisection method to resolve the root to within 10^{-8}.

For the x vector, our Hamiltonian is again independent of $x_{1:3}$, while $x_{4:5}$ will need to be approximated with gradient descent. Specifically,

$$(x_{1:3})_j^{k+1} = v_{1:3}, \tag{43}$$

and for $x_{4:5}$, we perform a few iterations of

$$(\theta^*, \varphi^*) = (\theta^*, \varphi^*) - \eta\nabla h(\theta^*, \varphi^*) \tag{44}$$

where

$$h(\theta, \varphi) = -\delta\tau\Bigg(\left|(p_1)_j^{k+1}\cos(\theta)\sin(\varphi) + (p_2)_j^{k+1}\sin(\theta)\sin(\varphi) + (p_3)_j^{k+1}\cos(\varphi)\right| + W\sqrt{\frac{(p_4)_j^{k+1}}{\sin^2(\varphi)} + (p_5)_j^{k+1}}\Bigg) + \frac{1}{2}((\theta - v_4)^2 + (\varphi - v_5)^2), \tag{45}$$

and then assign $(x_{4:5})_j^{k+1} = (\theta^*, \varphi^*)$.

As mentioned before, anywhere one sees a $\sin^2(\varphi)$ in a denominator, we replace it with $\sin^2(\varphi) + \varepsilon$ where $\varepsilon = 10^{-10}$, to avoid division by zero. Once again, the z updates are the same.

3.4. Adding obstacles computationally

One final matter to address is how to computationally account for obstacles. As mentioned in section 2.4, we can account for obstacles without using boundary conditions by modifying the Hamilton-Jacobi equation by multiplying the Hamiltonian by the indicator function of the free space. Thus we are actually solving

$$u_t + O(x,t)H(x, \nabla u) = 0, \quad u(x,0) = g(x), \tag{46}$$

where $O(x,t) = 0$ when $x \in \Omega_{\text{obs}}(t)$ and $O(x,t) = 1$ otherwise.

However, this raises the question of how to efficiently determine when $x \in \Omega_{\text{obs}}(t)$ for a given point x and time t. To make this possible, we restrict ourselves to the case of obstacles which are disjoint collections of balls. Thus at any time $t > 0$, we have

$$\Omega_{\text{obs}}(t) = \bigcup_{\ell=1}^{L} B(x_\ell(t), r_\ell(t)).$$

In this case, it is computationally efficient to check if x is in a obstacle at time t by checking its distance to each of the centers $x_\ell(t)$. [Note, there is another small abuse of notation here: the centers of the obstacles will be *spatial* coordinates only, whereas our state vector has spatial and angular coordinates.]

Of course, in application, it will almost never be the case that the set of obstacles is a finite collection of balls. In this case, we run the following greedy algorithm to iteratively fill the obstacle with disjoint balls of the largest possible radius.

(0) Given a set of obstacles $\Omega_0 = \Omega_{\text{obs}}(t)$ and a minimal radius $r_{\min}$, set $\ell = 1$ and do the following
(1) Compute the signed distance function $d_{\ell-1}(x)$ to $\partial\Omega_{\ell-1}$ using the level set method [46], fast marching method [47], parallel redistancing method [48,49], or another method of your choice. We choose the distance which is *positive* inside $\Omega_{\ell-1}$ and negative outside the set.
(2) Set $x_\ell = \arg\max_x d_{\ell-1}(x)$, $r_\ell = d_{\ell-1}(x_\ell)$, and $B_\ell = B(x_\ell, r_\ell)$.
(3) Set $\Omega_\ell = \Omega_{\ell-1} \setminus B_\ell$.
(4) If $r_\ell < r_{\min}$, break the loop. Otherwise, increment ℓ and return to (1).

Doing this results in a collection of disjoint balls which approximate the shape $\Omega_{\text{obs}}(t)$ and have radii larger than the prespecified minimum radius $r_{\min}$. This is demonstrated in Fig. 2. The $r_{\min}$ we use is chosen on an *ad hoc* basis. In cases where the obstacles have "long, thin" parts, one may need to choose $r_{\min}$ very small, while for smoother, more regular obstacles, a larger choice of $r_{\min}$ may suffice. In general, covering of surfaces by balls is a problem of interest in computational geometry [50,51], and theory regarding recursive subdivision algorithms for circle packing is an active area of research [52]. Our algorithm is admittedly quite simple, but is sufficient for our purposes.

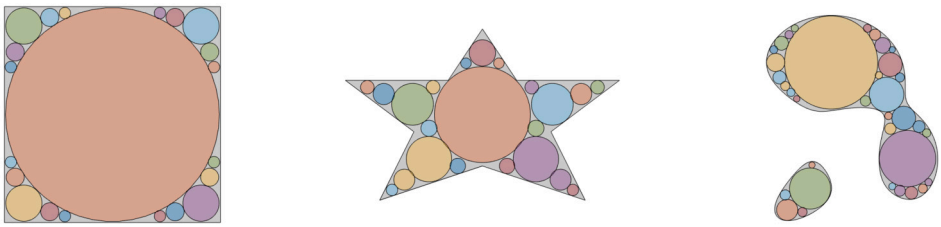

Fig. 2. Three obstacles approximated by disjoint circles.

Note that as described, this algorithm is somewhat computationally burdensome, but it is done in preprocessing. As an alternative, if one wishes to do this in real-time (and possibly do away with the need to approximate the obstacle by circles), one could one may be able to embed the algorithms from [48,49] into Algorithm 1, since they also compute the distance function based on Hopf-Lax type formulas. For our purposes, we use the algorithm described above and approximate obstacles with circles.

As one final note, we point out how adding the obstacle function $O(x,t)$ to the Hamilton-Jacobi equation as in (46) affects the update rules in each of the cases above. When replacing the Hamiltonian $H(x,p)$ with $O(x_s,t)H(x,p)$, where x_s denotes the spatial variables in x, the update rules for any of the costate variables do not change in any significant way: the indicator function of the obstacles is simply brought along as a multiplier and appears anywhere where $\delta\sigma$ appears. For example, the update rules for the car given by (32) and (33) become

$$(p_{1:2})_j^{k+1} = \left[\max\left\{0, 1 - \frac{O((x_{1:2})_j^k, t_j)\delta\sigma}{|\gamma^T\beta_{1:2}|}\right\} - 1\right](\gamma^T\beta_{1:2})\gamma + \beta_{1:2},$$
$$(p_3)_j^{k+1} = \max\left\{0, 1 - \frac{O((x_{1:2})_j^k, t_j)\delta\sigma W}{|\beta_3|}\right\}\beta_3,$$

which are the exact same as before aside from the inclusion of $O(x_s,t)$ evaluated at the current spatial coordinates $(x_{1:2})_j^k$ and time step t_j. The updates for the airplane or submarine are modified in analogous ways.

Likewise, because the obstacle function $O(x_s,t)$ depends only on the spatial variables in the state vector, the updates for the angular variables (described by equations (36), (37) for the car, by equations (39), (40) for the airplane, and by equations (44), (45) for the submarine) remain virtually unchanged, except once again for the inclusion of the obstacle function as a multiplier wherever $\delta\tau$ appears. For example, for the car, we simply need to change the function h in (37) to

$$h(\theta) = -\delta\tau O((x_{1:2})_j^k, t_j)|(p_1)_j^{k+1}\cos(\theta) + (p_2)_j^{k+1}\sin(\theta)| + \frac{1}{2}\left(\theta - ((x_3)_j^k - \tau((p_3)_j^{k+1} - (p_3)_{j+1}^{k+1}))\right)^2,$$

which is the exact same as before aside from the inclusion of $O(x_s,t)$. Analogous modifications are made for the plane and submarine.

The significant change comes in the updates of the spatial components which are given for the car, airplane, and submarine by equations (35), (38), and (43) respectively. The minimization problems for these updates can no longer be resolved analytically, so they approximate the solutions using gradient descent, as was done with the angular variables before. In each case, we need to resolve

$$(x_s)_j^{k+1} = \arg\min_{\overline{x}}\left\{-\delta\tau O(\overline{x}_s,t)H(x_a,p) + \frac{1}{2}\|\overline{x} - \nu\|^2\right\}$$

where $\nu = (x_s)_j^k - \tau((p_s)_j^{k+1} - (p_s)_{j+1}^{k+1})$. To reiterate, we are using x_s to denote the spatial variables of x (and p_s to denote the corresponding co-state variables), and we introduce x_a to denote the angular variables. Note that in our three models above, the Hamiltonian depends only on the angular variables x_a. To approximate the solution of the minimization problem, we use the gradient descent

$$x_s^* = x_s^* - \eta\nabla h(x_s^*)$$

where h is defined

$$h(x_s) = -\delta\tau O(x_s,t_j)H((x_a)_j^k, p_j^{k+1}) + \frac{1}{2}|x_s - \nu|^2$$

and then set $(x_s)_j^{k+1} = x_s^*$. In doing this, we will need to evaluate $\nabla O(\cdot,t)$. Recall, this function is the indicator function of the free space, which is discontinuous across the boundary of the obstacles. Accordingly, we approximate the indicator function of the free space by

$$O(x_s,t) \approx \frac{1}{2} + \frac{1}{2}\tanh(-100d(x_s,t))$$

where $d(x_s, t)$ is the signed distance function to the boundary of the obstacles at time t (positive inside the obstacles). This definition gives a smooth function which is approximately zero inside the obstacles and approximately 1 in the free space. Here the 100 in the definition is an arbitrary large number. We can quickly evaluate the distance function when we are approximating the obstacles by disjoint circles. This also gives an efficient method for computing the gradient of the distance function when x_s is outside the obstacles: it is the unit vector pointing from x_s to the center $x_\ell(t)$ of nearest ball to x_s in the collection of balls which approximates the obstacles.

Note that the updates for x_0^{k+1}—which are given by equation (34) for all three models—do not change because the Hamiltonian does not appear in these updates.

4. Simulations & discussion

In this section, we present the results of several simulations which demonstrate the efficacy of our models. In all simulations, we choose somewhat arbitrary and synthetic data, which we list for the individual simulations. The simulations were performed on a laptop computer with an AMD Ryzen 7 7735U processor running at a maximum of 4.75 GHz, and 16GB of RAM. Besides anything described above, no further efforts were made to optimize the algorithms, though certain parts are parallelizable, and the resolution of the minimizers used to update the state variables could likely be done more efficiently. Even so, our algorithms are quite efficient as reported below.

In all cases we fix the time step $\delta = 0.1$ and discretize the paths into $N = t/\delta$ steps, where t is the time chosen for the simulation as seen in Algorithm 1. Whenever it is necessary to use gradient descent to perform the updates described in sections 3.1-3.3, we perform 3 steps of gradient descent with a descent rate of $\eta = 0.15$. We also fix $\sigma = 0.5, \tau = 0.5$ and $\kappa = 1$. In each case, we continue the iteration until the maximum change in any coordinate of x or p is less than $\text{TOL} = 10^{-3}$. This tolerance may appear fairly large, but there was no appreciable difference in the resolved optimal trajectories when this tolerance was decreased to 10^{-8}. In each case, we use a maximal iteration count of $K = 100000$ but this maximal count was never reached when using the baseline parameter values. When we report clock times and iteration counts for the simulations below, these are averaged over 50 trials. Average clock time is then rounded to the nearest tenth of a second and average iteration count is rounded to the nearest integer.

Our first two simulations, the results of which are shown in Fig. 3 and Fig. 4, show a car beginning in the configuration $(x_0, y_0, \theta_0) = (-1.5, 1.5, \pi/2)$ and navigating around obstacles to the point $(x_f, y_f, \theta_f) = (2, 2, 3\pi/2)$ which are in the bottom-left and top-right (respectively) of the domain used to plot the results. In both cases, the maximum angular velocity is $W = 2$. The obstacles in both figures have the same shape, but in Fig. 3 the obstacles remain stationary, whereas in Fig. 4, the obstacles rotation clockwise around the origin at a constant rate of 1 radian per unit time. In the former figure, the time required to reach the endpoint is $T = 8$, whereas, when the obstacles rotate out of the way in the latter figure, the car can reach the endpoint by time $T = 6$. As mentioned in section 2.4, the time-horizon T that is chosen for the problem does actually matter here. Theoretically, for any T chosen large enough, there is an optimal path from the initial point to the end point. However, empirically if T is chosen too large the algorithm will require longer to converge. As chosen for each of these simulations, the time horizons of $T = 8$ and $T = 6$, respectively, are very close to the best possible travel time. For the moving obstacles example (Fig. 4), the algorithm was able to resolve optimal paths in an average 1.6 seconds using an average of 1748 iterations. If T is chosen too small, so that no path requiring time less than T to traverse can reach the endpoint, then the algorithm will often fail to converge, or converge to a path which is not meaningful. In these examples if the tolerance for convergence is decreased to $\text{TOL} = 10^{-8}$, the optimal trajectories do not appreciably change, but the clock time increases to roughly 15 seconds, requiring on the order of 10000-15000 iterations for convergence.

Our next simulation has a Dubins airplane circling down for a landing, which is displayed in Fig. 5. The airplane begins in the configuration $(x_0, y_0, z_0, \theta_0) = (0, 0, \frac{1}{2}, 0)$ and must end at $(x_f, y_f, z_f, \theta_f) = (0, 0, 0, 0)$. In this case, the maximum angular velocity in the xy-plane is $W_{xy} = 2.5$ and the maximum angular velocity in the z-direction is $W_z = 0.5$. In order to make its descent, the airplane flies in a perfect figure eight in the xy-plane, while descending in the z-direction. In Fig. 5, the 3D view is displayed on the left of each panel and the projection down to the xy-plane is displayed on the right of each panel. This simulation required an average of 0.4 seconds of clock time to resolve the optimal path, doing so in an average of 2506 iterations. The updates for the plane are a bit simpler than those for the car (since the plane can only move forward) which explains why more iterations can be performed in less clock time. For this example, there is not much penalty for deceasing the convergence tolerance. Indeed, decreasing the tolerance all the way to $\text{TOL} = 10^{-8}$, the algorithm still usually resolves the optimal path in an average of 1.2 seconds and 7233 iterations (though the "worst" simulation required 5.4 seconds and 32268 iterations). An interesting note is that the Dubins airplane does not have small time local controllability (STLC), meaning roughly that the plane cannot necessarily reach a point which is with distance ε of its current configuration in time $O(\varepsilon)$. The value function for a control problem without STLC can be discontinuous, which necessitates special consideration when designing grid based numerical schemes for these problems as discussed in [6]. Because our method only ever deals with the Hamiltonian and initial data function (and never with approximations to derivatives of the value function), our method is agnostic to whether the dynamics satisfy an STLC condition.

We next consider a Dubins submarine. Here we take the initial configuration to be $(x_0, y_0, z_0, \theta_0, \varphi_0) = (-1.8, -1.8, 0, \pi/4, \pi/2)$ and let the submarine navigate through and around obstacles to $(x_f, y_f, z_f, \theta_f, \varphi_f) = (1.3, 1.5, -1.5, 0, \pi/2)$. In this case, the maximum angular velocity is $W = 2$. The result is displayed in Fig. 6, where the "third person" view is displayed on the left of each panel and the corresponding "first person" view is displayed on the right. In Fig. 6, the red bubbles are obstacles. This simulation required an average of 5.0 seconds of clock time to resolve the optimal trajectory in an average of 1936 iterations. Here the iterations are more expensive due to the resolution of $p_{4:5}$ by the bisection method described in equations (41), (42). Once again, if we decrease the

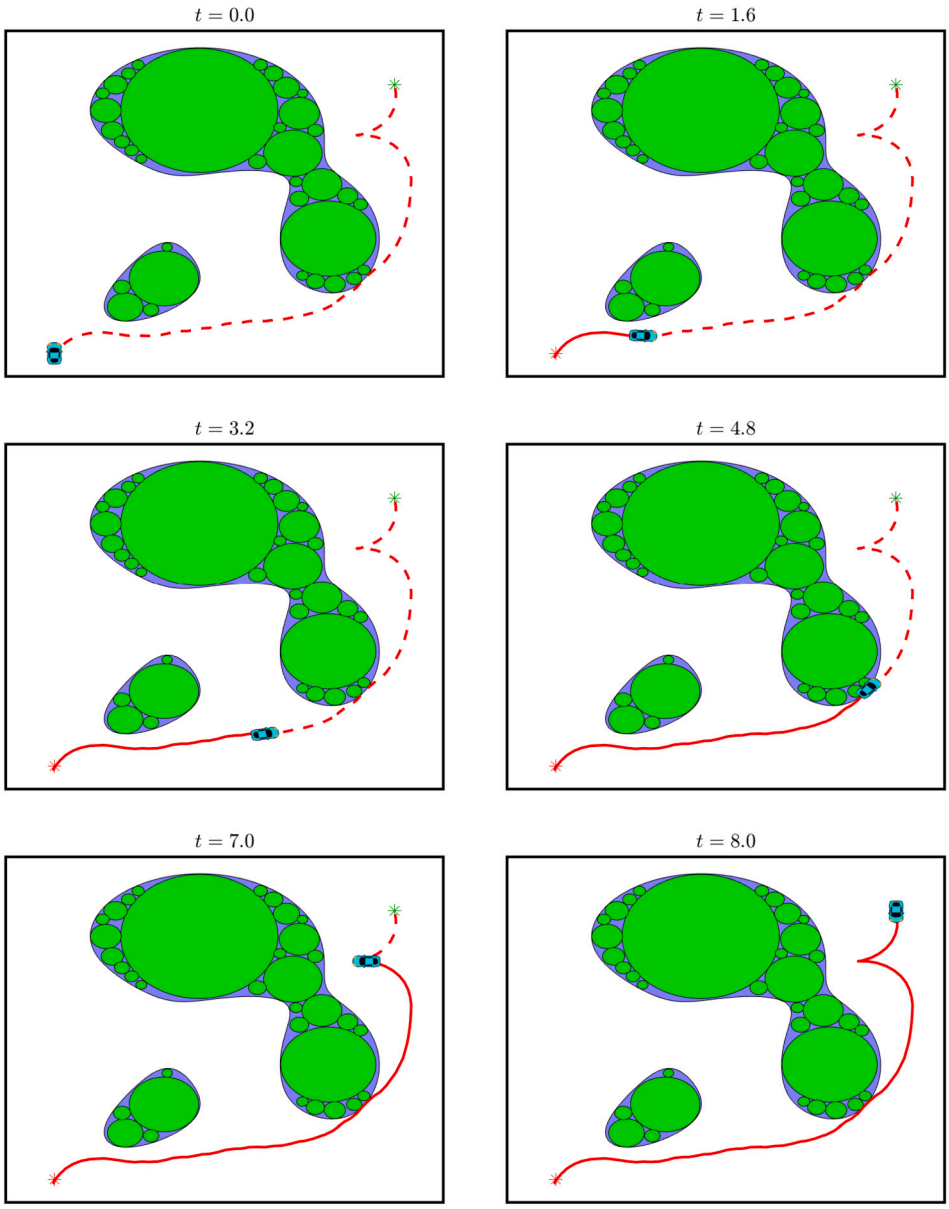

Fig. 3. A car navigating around obstacles to end point in the top right of each frame. The obstacles are the blue regions, and the collection of green circles is their computational representation.

convergence tolerance to TOL $= 10^{-8}$, the path generated is not appreciably different, but requires roughly 28 seconds of clock time and on the order of 8000 iterations to resolve.

Finally, we want to empirically study the affect that varying the ambient parameters has on the convergence of the algorithm. To do so, we focus on the example of the car with moving obstacles in Fig. 4, and vary the time step δ, the time horizon T, and the PDHG parameters σ, τ, κ. Once again, the iteration counts are averaged over 50 trials, and the average is rounded to the nearest integer. The results are in Fig. 7. In each table in the figure, aside from the parameter being highlighted, all other parameters are held at their baseline values of $\delta = 0.1, T = 6, \sigma = \tau = 0.5, \kappa = 1$.

Looking at the table, there is a reasonably clear trend that decreasing δ increases the number of iterations requires. This is likely because with smaller δ there are simply more points that need to be resolved (and more randomness in the initialization which needs to be overcome). Using this same reasoning, one may expect that increasing T also increases the number of iterations required. However, this does not hold. When decreasing T to 4, there is now no path which can even come close to reaching the final point. In cases like this, we consistently found that the algorithm would fail to converge within the maximum iteration count of 100000 (and showed no apparent signs of nearing convergence). When T is increased, the iteration count significantly falls. Unfortunately, choosing such T is still undesirable. When T is chosen too large, since any path which reaches the endpoint is an optimal path,

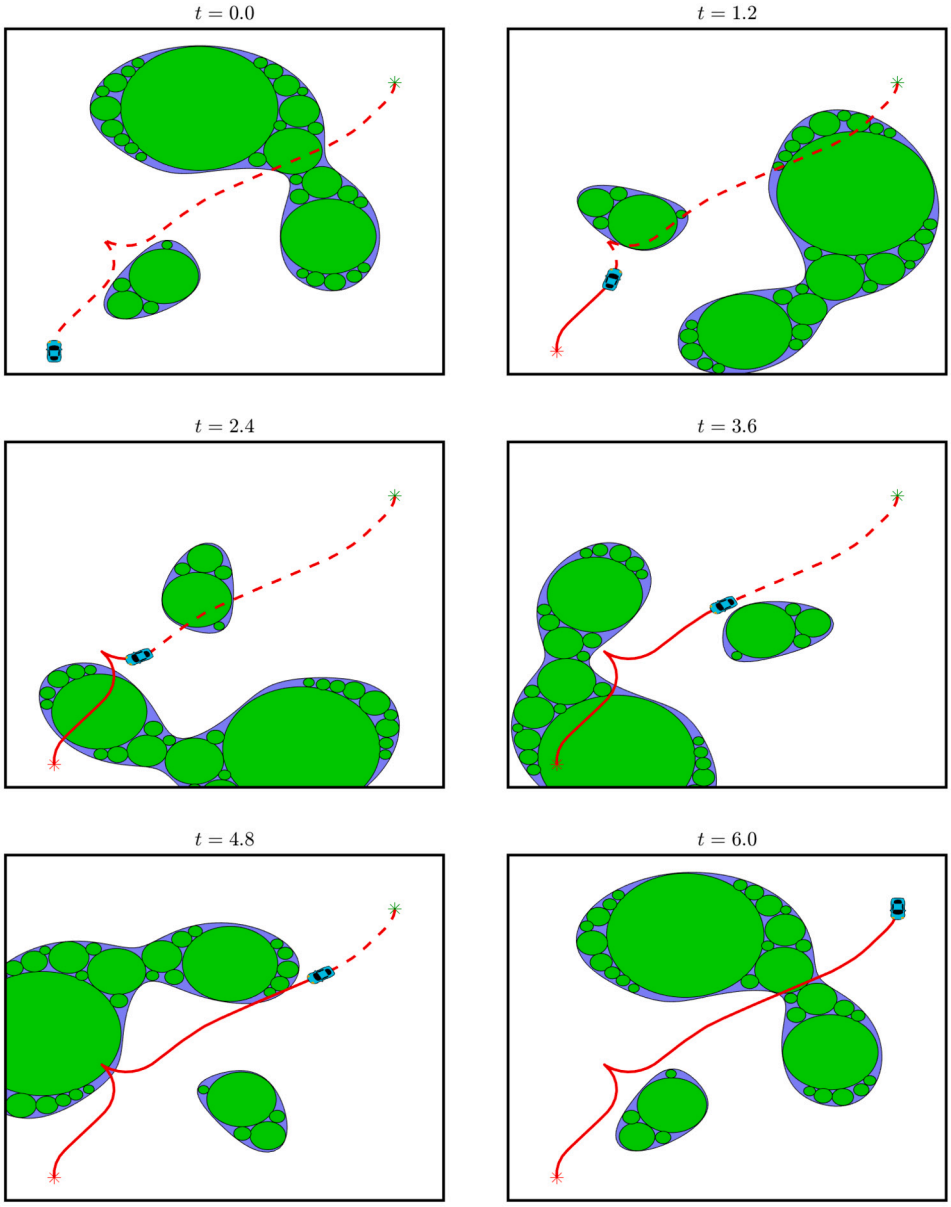

Fig. 4. A car navigating around the same obstacles as in Fig. 4, but in this case the obstacles begin rotating clockwise about the center of the domain.

there are infinitely many optimal paths (assuming small-time local controllability). Thus the algorithm has an easier time finding an optimal path simply because there are more optimal paths. But observing the selected path, the car will usually dawdle for a bit, wasting some time before arriving at the endpoint at the time horizon T. Paths like this are likely not what one would wish to resolve, though in many cases, one could easily intuit the "correct" optimal path from one of these paths by removing segments of the path which one deems as purposeless motion. One final note regarding either decreasing δ or increasing T is that either of these changes increases the cost per iteration by increasing the number of time steps. So for example, when δ is halved, there is a two-fold increase in computation because there are twice as many points, but there is also an increase in computation due to the increased iteration count. Because of this, to maximize efficiency, it is important to choose δ as large as is allowable for ones particular application.

The PDHG parameters are bound by $\sigma\tau \leq 0.25$ and $\kappa \in [0, 1]$. Thus, from the baseline values, neither of σ or τ can be increased without decreasing the other, and κ can only be decreased. Looking at the tables, there are clear trends which indicate that decreasing σ or κ increases the number of iterations required for convergence. For τ, there isn't an abundantly clear trend, though it is possible that decreasing τ could provide a marginal benefit when compared with the baseline value. Another possible benefit of decreasing τ is that it would allow one to increase σ further, which could decrease the iteration count further. Using this observation, we tested further combinations of larger σ with smaller τ (keeping other parameters at baseline) and from our simulations, the best case scenario was $\sigma = 0.75, \tau = 0.25$, with an average iteration count of 1442 iterations before convergence.

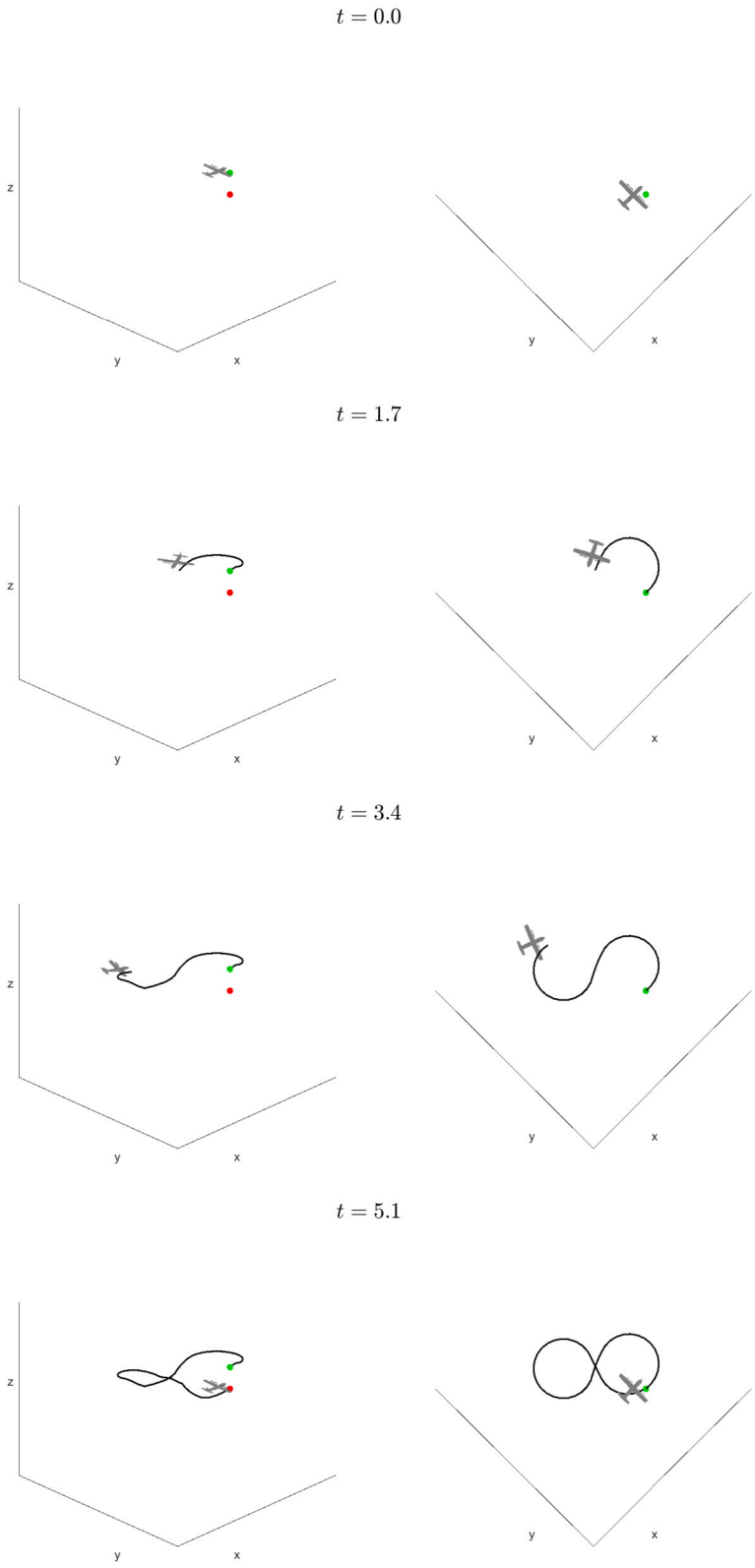

Fig. 5. A Dubins airplane circling down for a landing. The 3D view is displayed on the left of each panel, and the projection down to the xy-axis is displayed on the right.

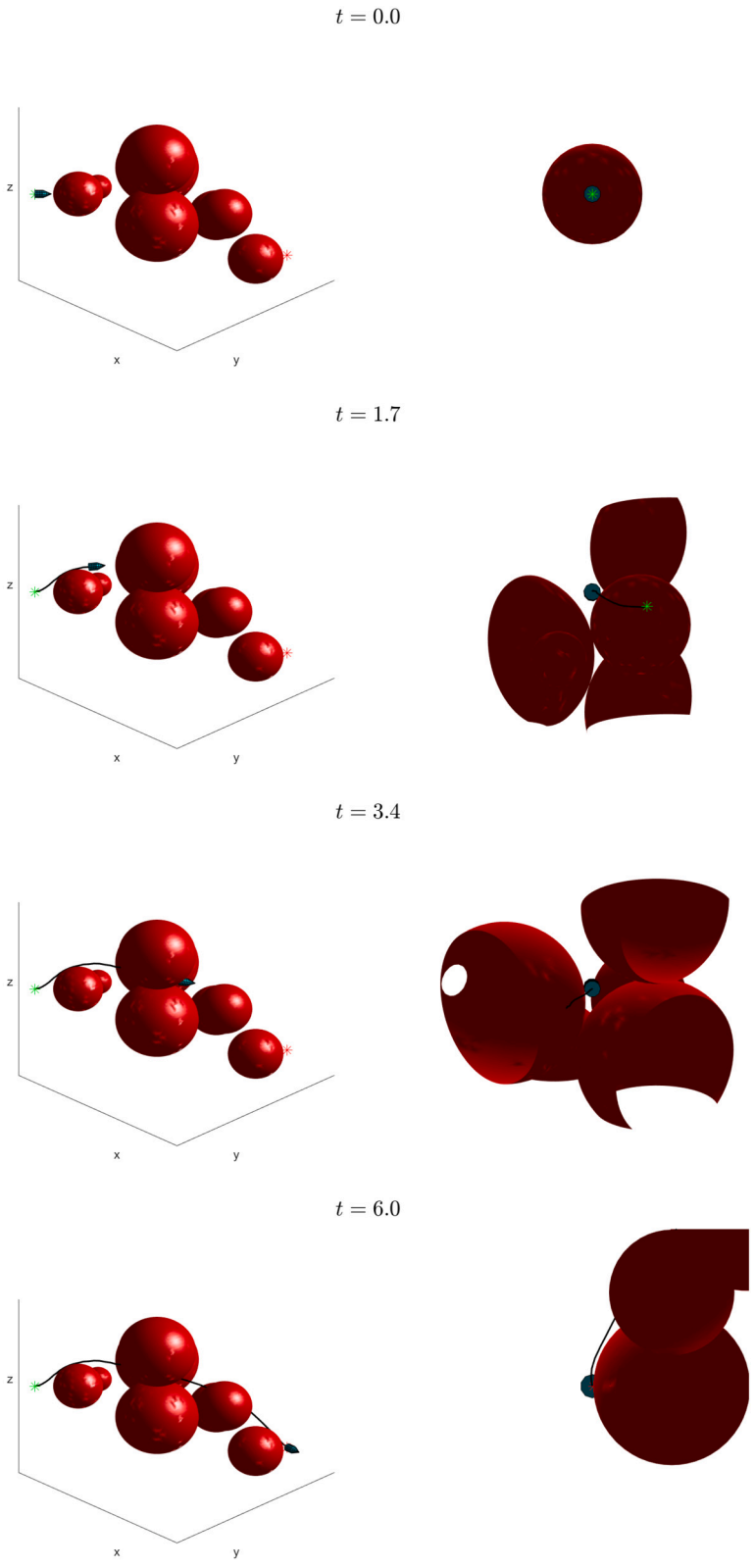

Fig. 6. A Dubins submarine navigating through and around obstacles to reach its desired destination. The "third-person" view is displayed on the left of each panel, while the corresponding "first-person" view is displayed on the right. The red bubbles are obstacles.

δ	Iters.
0.1	1748
0.05	1777
0.025	2578
0.0125	4317

T	Iters.
4	100000+
6	1748
10	864
14	874

σ	Iters.
0.5	1748
0.25	2340
0.125	3405
0.0625	4382

τ	Iters.
0.5	1748
0.25	1660
0.125	1763
0.0625	1630

κ	Iters.
1	1748
0.75	2173
0.5	3618
0.25	85065*

Fig. 7. Tables detailing how the number of iterations until convergence depends on the different parameters for the example in Fig. 4. Each iteration count is averaged over 50 trials, and in each table all parameters aside from the one being highlighted are held at baseline values. For $T = 4$, no trial resulted in convergence within the maximum iteration count of 100000. The asterisk for iteration count when $\kappa = 0.25$ is included because in that case, 18 of the 50 trials failed to converge within the maximum iteration count of 100000 so the value is artificially deflated.

5. Conclusion & future work

In this paper, we developed an algorithm for optimal path-planning for curvature constrained motion which includes kinematic models for simple cars, airplanes, and submarines. Our method relied on a level-set, PDE-based formulation of the optimal path-planning problem, wherein the value function is not the optimal travel time (as in more control theoretic models), but optimal distance to the desired endpoint. This allowed us to solve the problem using new numerical methods which resolve the solutions to Hamilton-Jacobi equations via optimization problems. We discussed the ramifications of this modeling decision, and described in detail the implementation of our method. Finally, we demonstrated our method on some synthetic examples. In these examples, we are able to resolve optimal trajectories for cars, planes, and submarines in a matter of seconds. This allows one to maintain the PDE and control based methodology (and thus maintain interpretability) without sacrificing efficiency or scalability.

An immediate avenue of future work is to look into manners in which convergence of Algorithm 1 could be accelerated using more sophisticated optimization methods, Nesterov-type acceleration, or better approximation of the proximal operator. Here basic gradient descent was chosen for ease of implementation and exposition, but this could very likely be improved to make the algorithm even faster and more robust. Next, while this represents a step toward real-time PDE-based path-planning algorithms, there are still difficulties to overcome. As presented, our method can compute optimal paths in a fully-known environment. In realistic scenarios, the exact configuration of obstacles is likely unknown and obstacles may move unpredictably. Accordingly, it would be desirable if the vehicles had only local information regarding terrain and obstacles which is updated along the trajectory so that they can react and recompute paths in semi-real-time. In doing so, one would need to abandon the hope of finding globally optimal trajectories, because only local information would be known. However, this also presents the difficulty of correctly choosing the new time-horizon upon adding new information in the Hamiltonian and recomputing the optimal path, so it may require a significant change in the modeling. In the same vein, it could prove interesting to adapt and apply our method to problems with other realistic concerns such as energy-efficient path-planning or rider comfort via jerk bounds or control regularization. Finally, we would like to adapt this method to a multi-agent control or many-player differential games scenario.

CRediT authorship contribution statement

Christian Parkinson: Writing – original draft, Writing – review & editing. **Isabelle Boyle:** Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgements

The authors would like to think Robert Ferrando for many discussions regarding this work. The authors were supported in part by NSF DMS-1937229 through the Data Driven Discovery Research Training Group at the University of Arizona.

References

[1] L.E. Dubins, On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents, Am. J. Math. 79 (3) (1957) 497–516.
[2] J. Reeds, L. Shepp, Optimal paths for a car that goes both forwards and backwards, Pac. J. Math. 145 (2) (1990) 367–393.
[3] J. Barraquand, J.-C. Latombe, Nonholonomic multibody mobile robots: controllability and motion planning in the presence of obstacles, Algorithmica 10 (2–4) (1993) 121.
[4] P.K. Agarwal, H. Wang, Approximation algorithms for curvature-constrained shortest paths, SIAM J. Comput. 30 (6) (2001) 1739–1772.
[5] R. Takei, R. Tsai, H. Shen, Y. Landa, A practical path-planning algorithm for a simple car: a Hamilton-Jacobi approach, in: Proceedings of the 2010 American Control Conference, 2010, pp. 6175–6180.
[6] R. Takei, R. Tsai, Optimal trajectories of curvature constrained motion in the Hamilton-Jacobi formulation, J. Sci. Comput. 54 (2) (2013) 622–644.
[7] D.J. Arnold, D. Fernandez, R. Jia, C. Parkinson, D. Tonne, Y. Yaniv, A.L. Bertozzi, S.J. Osher, Modeling environmental crime in protected areas using the level set method, SIAM J. Appl. Math. 79 (3) (2019) 802–821.
[8] C. Parkinson, D. Arnold, A.L. Bertozzi, Y.T. Chow, S. Osher, Optimal human navigation in steep terrain: a Hamilton–Jacobi–Bellman approach, Commun. Math. Sci. 17 (1) (2019) 227–242.
[9] C. Parkinson, D. Arnold, A. Bertozzi, S. Osher, A model for optimal human navigation with stochastic effects, SIAM J. Appl. Math. 80 (4) (2020) 1862–1881.
[10] C. Parkinson, A.L. Bertozzi, S.J. Osher, A Hamilton-Jacobi formulation for time-optimal paths of rectangular nonholonomic vehicles, in: 2020 59th IEEE Conference on Decision and Control (CDC), IEEE, 2020, pp. 4073–4078.
[11] C. Parkinson, M. Ceccia, Time-optimal paths for simple cars with moving obstacles in the Hamilton-Jacobi formulation, in: 2022 American Control Conference (ACC), IEEE, 2022, pp. 2944–2949.
[12] B. Chen, K. Peng, C. Parkinson, A.L. Bertozzi, T.L. Slough, J. Urpelainen, Modeling illegal logging in Brazil, Res. Math. Sci. 8 (2) (2021) 1–21.
[13] M. Gee, A. Vladimirsky, Optimal path-planning with random breakdowns, IEEE Control Syst. Lett. 6 (2021) 1658–1663.
[14] E. Cartee, L. Lai, Q. Song, A. Vladimirsky, Time-dependent surveillance-evasion games, in: 2019 IEEE 58th Conference on Decision and Control (CDC), IEEE, 2019, pp. 7128–7133.
[15] A. Shukla, E. Singla, P. Wahi, B. Dasgupta, A direct variational method for planning monotonically optimal paths for redundant manipulators in constrained workspaces, Robot. Auton. Syst. 61 (2) (2013) 209–220.
[16] F. Zhang, C. Wang, C. Cheng, D. Yang, G. Pan, Reinforcement learning path planning method with error estimation, Energies 15 (1) (2022).
[17] K. Wan, D. Wu, B. Li, X. Gao, Z. Hu, D. Chen, Me-maddpg: an efficient learning-based motion planning method for multiple agents in complex environments, Int. J. Intell. Syst. 37 (3) (2022) 2393–2427.
[18] R. Deng, Q. Zhang, R. Gao, M. Li, P. Liang, X. Gao, A trajectory tracking control algorithm of nonholonomic wheeled mobile robot, in: 2021 6th IEEE International Conference on Advanced Robotics and Mechatronics (ICARM), 2021, pp. 823–828.
[19] J.J. Johnson, M.C. Yip, Chance-constrained motion planning using modeled distance- to-collision functions, in: 2021 IEEE 17th International Conference on Automation Science and Engineering (CASE), 2021, pp. 1582–1589.
[20] M. Detrixhe, F. Gibou, C. Min, A parallel fast sweeping method for the Eikonal equation, J. Comput. Phys. 237 (2013) 46–55.
[21] M. Detrixhe, F. Gibou, Hybrid massively parallel fast sweeping method for static Hamilton–Jacobi equations, J. Comput. Phys. 322 (2016) 199–223.
[22] J. Darbon, S. Osher, Algorithms for overcoming the curse of dimensionality for certain Hamilton–Jacobi equations arising in control theory and elsewhere, Res. Math. Sci. 3 (1) (2016) 19.
[23] Y.T. Chow, J. Darbon, S. Osher, W. Yin, Algorithm for overcoming the curse of dimensionality for state-dependent Hamilton-Jacobi equations, J. Comput. Phys. 387 (2019) 376–409.
[24] A.T. Lin, Y.T. Chow, S.J. Osher, A splitting method for overcoming the curse of dimensionality in Hamilton–Jacobi equations arising from nonlinear optimal control and differential games with applications to trajectory generation, Commun. Math. Sci. 16 (7) (1 2018).
[25] S. Osher, J.A. Sethian, Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations, J. Comput. Phys. 79 (1) (1988) 12–49.
[26] F. Gibou, R. Fedkiw, S. Osher, A review of level-set methods and some recent applications, J. Comput. Phys. 353 (2018) 82–109.
[27] R. Bellman, Dynamic programming, Science 153 (3731) (1966) 34–37.
[28] M.G. Crandall, P.-L. Lions, Viscosity solutions of Hamilton-Jacobi equations, Trans. Am. Math. Soc. 277 (1) (1983) 1–42.
[29] M.G. Crandall, H. Ishii, P.-L. Lions, User's guide to viscosity solutions of second order partial differential equations, Bull. Am. Math. Soc. 27 (1) (1992) 1–67.
[30] M. Bardi, I.C. Dolcetta, et al., Optimal Control and Viscosity Solutions of Hamilton-Jacobi-Bellman Equations, vol. 12, Springer, 1997.
[31] H. Chitsaz, S.M. LaValle, Time-optimal paths for a Dubins airplane, in: 2007 46th IEEE Conference on Decision and Control, IEEE, 2007, pp. 2379–2384.
[32] H.J. Choi, Time-optimal paths for a Dubins car and Dubins airplane with a unidirectional turning constraint, Phd thesis, University of Michigan, 2014.
[33] W. Cai, M. Zhang, Y.R. Zheng, Task assignment and path planning for multiple autonomous underwater vehicles using 3D Dubins curves, Sensors 17 (7) (2017) 1607.
[34] Y. Duan, S. Patil, J. Schulman, K. Goldberg, P. Abbeel, Planning locally optimal, curvature-constrained trajectories in 3d using sequential convex optimization, in: 2014 IEEE International Conference on Robotics and Automation (ICRA), 2014, pp. 5889–5895.
[35] S. Jafarpour, On small-time local controllability, SIAM J. Control Optim. 58 (1) (2020) 425–446.
[36] C. Parkinson, A rotating-grid upwind fast sweeping scheme for a class of Hamilton-Jacobi equations, J. Sci. Comput. 88 (1) (2021) 13.
[37] C.-Y. Kao, S. Osher, Y.-H. Tsai, Fast sweeping methods for static Hamilton–Jacobi equations, SIAM J. Numer. Anal. 42 (6) (2005) 2612–2632.
[38] Y.-T. Zhang, H.-K. Zhao, J. Qian, High order fast sweeping methods for static Hamilton–Jacobi equations, J. Sci. Comput. 29 (2006) 25–56.
[39] J.N. Tsitsiklis, Efficient algorithms for globally optimal trajectories, IEEE Trans. Autom. Control 40 (9) (1995) 1528–1538.
[40] J.A. Sethian, Fast marching methods, SIAM Rev. 41 (2) (1999) 199–235.
[41] K. Alton, I.M. Mitchell, Fast marching methods for stationary Hamilton–Jacobi equations with axis-aligned anisotropy, SIAM J. Numer. Anal. 47 (1) (2009) 363–385.
[42] J.A. Sethian, A. Vladimirsky, Ordered upwind methods for static Hamilton–Jacobi equations: theory and algorithms, SIAM J. Numer. Anal. 41 (1) (2003) 325–363.
[43] A. Chambolle, T. Pock, A first-order primal-dual algorithm for convex problems with applications to imaging, J. Math. Imaging Vis. 40 (2011) 120–145.
[44] M. Zhu, T. Chan, An efficient primal-dual hybrid gradient algorithm for total variation image restoration, Ucla Cam Rep. 34 (2008) 8–34.
[45] E. Esser, X. Zhang, T.F. Chan, A general framework for a class of first order primal-dual algorithms for convex optimization in imaging science, SIAM J. Imaging Sci. 3 (4) (2010) 1015–1046.
[46] S. Osher, R.P. Fedkiw, Level Set Methods and Dynamic Implicit Surfaces, vol. 1, Springer New York, 2005.
[47] J.A. Sethian, Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry Fluid Mechanics, Computer Vision, and Materials Science, vol. 3, Cambridge University Press, 1999.
[48] B. Lee, J. Darbon, S. Osher, M. Kang, Revisiting the redistancing problem using the Hopf–Lax formula, J. Comput. Phys. 330 (2017) 268–281.
[49] M. Royston, A. Pradhana, B. Lee, Y.T. Chow, W. Yin, J. Teran, S. Osher, Parallel redistancing using the Hopf–Lax formula, J. Comput. Phys. 365 (2018) 7–17.

[50] N. Amenta, S. Choi, R.K. Kolluri, The power crust, unions of balls, and the medial axis transform, in: Combinatorial Curves and Surfaces, Comput. Geom. 19 (2) (2001) 127–153.
[51] F. Cazals, T. Dreyfus, S. Sachdeva, N. Shah, Greedy geometric algorithms for collection of balls, with applications to geometric approximation and molecular coarse-graining, Comput. Graph. Forum 33 (6) (2014) 1–17.
[52] Y. Luo, Y. Zhang, Circle packings, renormalizations and subdivision rules, arXiv preprint, arXiv:2308.13151, 2023.