LevelSetPy: A GPU-Accelerated Package for Hyperbolic Hamilton-Jacobi Partial Differential Equations' Solubility

LEKAN MOLU, Microsoft Research, USA

This article is devoted to several themes surrounding the scalable numerical solutions to hyperbolic HJ PDEs viz., their implicit representation on co-dimension one domains; dynamics evolution with levelsets; upwinding spatial derivatives; total variation diminishing Runge-Kutta integration schemes via method of lines; and their applications to the theory of reachable sets and safety-critical systems analyses. The HJ evolution equation and its level set dynamics that we chiefly consider are increasingly finding interest in multiple research domains including (a) analyzing safety-critical problems in reinforcement learning, and broadly in robotics and automation; (b) control engineering; (c) computing flows in transport problems; (d) aerospace and defense engineering domains; (e) geometric optics; (f) volume of fluid analysis; and (g) image processing inter alia. We briefly review HJ theory and its viscosity approximations, describe a hierarchy of library components, and provide rigorously-tested representative numerical examples in reachability differential games, transport analysis, and time-optimal control problems. This GPU software package allows for easy portability to modern libraries for the numerical analyses of the Bellman and Isaacs equations – the mainstays of modern machine learning today. Furthermore, we provide a CPU implementation in python that is significantly faster than existing alternatives. Let us enquire.

CCS Concepts: • **Software and its engineering → Software libraries and repositories**; • **Applied computing** → *Physical sciences and engineering*; • **Mathematics of computing** → *Solvers*.

Additional Key Words and Phrases: partial differential equations, level sets, reachability theory

ACM Reference Format:
Lekan Molu. 20XX. LevelSetPy: A GPU-Accelerated Package for Hyperbolic Hamilton-Jacobi Partial Differential Equations' Solubility. *J. ACM* 00, 0, Article 000 (20XX), 40 pages. https://doi.org/XXXXXXX.XXXXXXX

1 OVERVIEW

The *reliability*[1] of the complex algorithms (in AI, modern cyberphysical systems etc) that we design has become paramount given the dangers that may evolve if nominally envisioned system performance falter. Even so, the need for scalable and faster numerical algorithms in software for *verification* [14] and *validation* [13] has become timely given the emerging growth of complexity in the systems that we design and build. That which entails generating evidence that a system, or any its components satisfy all specified requirements and functional and allocated baselines is termed verification [14]. Validation entails providing unreprovable evidence that system capabilities comply with an end-user's performance requirements and satisfy its intended operational environment's specifications [13]. We focus on the numerical tools for safety assurance (ascertaining the freedom of a system from harm) in a verification sense in this article.

[1]Reliability is taken to mean guaranteed consistency in system performance over time.

Author's address: Lekan Molu, lekanmolu@microsoft.com, Microsoft Research, 300 Lafayette Street, New York, NY, USA, 10012.

0004-5411/20XX/0-ART000 $15.00
https://doi.org/XXXXXXX.XXXXXXX

The foremost open-source verification software for engineering applications based on Hamilton-Jacobi equations [19, 30] and level set methods [39, 44] is the CPU-based MATLAB® [35] level sets toolbox, before computing via graphical processing units (GPU) became the era of pervasive. Since then, there has been a lot of improvement in computer hardware design, architecture, and code-acceleration. This paper principally describes a GPU-accelerated software package – written entirely in python and accelerated on graphical processing units (GPUs) – for numerically resolving generalized discontinuous solutions to Cauchy-type (or time-dependent) Hamilton-Jacobi (HJ) hyperbolic partial differential equations (PDE's) that arise in many problem contexts including (multi-agent) reinforcement learning, robotics, control theory, differential games, as well as flow and transport phenomena *inter alia*.

Accompanying the package are implicit calculus operations on dynamic codimension-one interfaces embedded on surfaces in $\mathbb{R}^n$, and numerical (spatial and temporal) discretization schemes for hyperbolic partial differential equations. Furthermore, we describe explicit integration schemes including Lax-Friedrichs, Courant-Friedrichs-Lewy (CFL) integration TVD-RK conditioning schemes for HJ Hamiltonians of the form $\boldsymbol{H}(\boldsymbol{x}, \boldsymbol{p})$, where $\boldsymbol{x}$ is the state and $\boldsymbol{p}$ is the co-state. Finally, extensions to reachability analyses for continuous and hybrid systems, formulated as optimal control or game theory problems using viscosity solutions to HJ PDE's is described. While our emphasis is on the resolution of safe sets in a reachability context for verification settings, the applications of the software package herewith presented extend beyond control engineering applications.

The GPU package, implemented in CuPy [38], is available on the author's github repository: `LevelSetPy`. Extensions to other python GPU programming language is straightforward (as detailed in the CuPy `interoperability document`). The CPU implementation (in Python) can be found at `on the cpu-numpy tree of LevelSetPy`. In addition, installation instructions are available on the github repository. In all, we try to follow the Python Enhancement Proposals (PEP) 8 style guide[2] as much as possible but in order not to break readability with respect to the original MATLAB®code, we have tried to edge on the side of consistency within the previous project.

2 BACKGROUND AND MOTIVATION

Our chief interest is the evolution form of the HJ equation

$$\begin{aligned} \boldsymbol{v}_t(x,t) + \boldsymbol{H}(t; x, \nabla_x \boldsymbol{v}) &= 0 \text{ in } \Omega \times (0,T] \\ \boldsymbol{v}(x,t) &= \boldsymbol{g}, \text{ on } \partial\Omega \times \{t = T\}, \boldsymbol{v}(x,0) = \boldsymbol{v}_0(x) \text{ in } \Omega \end{aligned} \tag{1}$$

or the convection equation

$$\begin{aligned} \boldsymbol{v}_t + \sum_{i=0}^{N} f_i(u)_{x_i} &= 0, \quad \text{for } t > 0, \boldsymbol{x} \in \mathbb{R}^n, \\ \boldsymbol{v}(\boldsymbol{x},0) &= \boldsymbol{v}_0(\boldsymbol{x}), \ \ \boldsymbol{x} \in \mathbb{R}^n \end{aligned} \tag{2}$$

where Ω is an open set in $\mathbb{R}^n$; $\boldsymbol{x}$ is the state; $\boldsymbol{v}_t$ denotes the partial derivative(s) of the solution $\boldsymbol{v}$ with respect to time t; the Hamiltonian $\boldsymbol{H} : (0,T] \times \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ and f are continuous; $\boldsymbol{g}$, and $\boldsymbol{v}_0$ are bounded and uniformly continuous (BUC) functions in $\mathbb{R}^n$; and $\nabla_x \boldsymbol{v}$ is the spatial gradient of $\boldsymbol{v}$. It is assumed that $\boldsymbol{g}$ and $\boldsymbol{v}_0$ are given.

Solving problems described by (1) under appropriate boundary and/or initial conditions using the method of characteristics is limiting as a result of crossing characteristics [10]. In the same vein, global analysis is virtually impossible owing to the lack of existence and uniqueness of solutions $\boldsymbol{v} \in C^1(\Omega) \times (0,T]$ even if $\boldsymbol{H}$ and $\boldsymbol{g}$ are smooth [10].

[2]Python PEP 8 style guide: peps.python.org/pep-0008/

The method of "vanishing viscosity", based on the idea of traversing the limit as $\delta \to 0$ in the hyperbolic equation (1) (where a parameter $\delta > 0$ "endows" the problem in a viscosity sense as in gas dynamics [25]), allows generalized (discontinuous) solutions [17] whereupon if $\boldsymbol{v} \in W^{1,\infty}_{loc}(\Omega) \times (0,T]$ and $\boldsymbol{H} \in W^{1,\infty}_{loc}(\Omega)$, one can lay claim to strong notions of general existence, stability, and uniqueness to BUC solutions $\boldsymbol{v}^{\delta}$ of the (approximate) viscous Cauchy-type HJ equation

$$\begin{aligned} \boldsymbol{v}^{\delta}_t + \boldsymbol{H}(t; \boldsymbol{x}, \nabla_x \boldsymbol{v}^{\delta}) - \delta \Delta \boldsymbol{v}^{\delta} = 0 \text{ in } \Omega \times (0,T] \\ \boldsymbol{v}^{\delta}(x,t) = \boldsymbol{g}, \text{ on } \partial\Omega \times \{t = T\}, \boldsymbol{v}^{\delta}(\boldsymbol{x},0) = \boldsymbol{v}_0(\boldsymbol{x}) \text{ in } \Omega \end{aligned} \tag{3}$$

in the class $\mathrm{BUC}(\Omega \times [0,T]) \cap C^{2,1}(\Omega \times (0,T])$ i.e. continuous second-order spatial and first order time derivatives for all time $T < \infty$. Crandall and Lions [11] showed that $|\boldsymbol{v}^{\delta}(\boldsymbol{x},t) - \boldsymbol{v}(\boldsymbol{x},t)| \leq k\sqrt{\delta}$ for $\boldsymbol{x} \in \Omega$ and $t > 0$. For most of this article, we are concerned with *generalized* viscosity solutions of the manner described by (3).

Popular means of computing the solution $\boldsymbol{v}^{\delta}$ of (3) leverage hyperbolic conservation laws in generating *propagation of surfaces under curvature* (PSC) [40]. These PSC algorithms *implicitly* embed $(n-1)$ dimensional *interfaces* (or isocontours) between two or more separable regions in spatial dimensions belonging in $\mathbb{R}^n$. The interface's speed is typically set as a function of the dynamical system's curvature, and the front gets passively advected by a coupled flow.

This implicit representation has found solution in a wide array of flow/motion of many physical phenomena such as gas dynamics, global illumination problems, and problems arising in the evolution of the trajectories of dynamical systems in a reachable control context [34, 36, 37]. These schemes work despite these phenomena developing discontinuities, corners, or cusps as dynamics evolve owing to the rich theory of viscosity solutions to HJ problems [9, 10, 17, 31].

Mitchell [37] connected techniques used in level set methods to reachability analysis in optimal control, essentially showing that the zero-level set of the differential zero-sum two-person game in an Hamilton-Jacobi-Isaacs (HJI) setting [17, 26] constitutes the safe set of a reachability problem. Reachability concerns evaluating the *decidability* of a dynamical system's trajectories' evolution throughout a state space. Decidable reachable systems are those where one can compute all states that can be reached from an initial condition in *a finite number of steps*. For inf-sup or sup-inf optimal control problems [32], the Hamiltonian is related to the *backward* reachable set of a dynamical system. The essential fabric of this techical communique revolves around reachability problems defined on co-dimension-one surfaces.

The well-known `LevelSet Toolbox` [35] is the consolidated `MATLAB®` package that contains the gridding methods, boundary conditions, time and spatial derivatives, integrators and helper functions. While Mitchell motivated the execution of the toolkit in `MATLAB®` for the expressiveness the language provides, modern data manipulation and scripting libraries often render the original package non-portable across distributed hardware since it lacks other programming language interoperability, particularly python and its associated scientific computing libraries such as `Numpy`, `Scipy`, `PyTorch` and their variants.

In this regard, we revisit the major algorithms necessary for implicit surface representation of HJ PDEs, write the spatial, temporal, and monotone difference schemes in Python, accelerated onto GPUs via CuPy [38] and present representative numerical examples.

2.1 Contributions

Given the prevalence of HJ PDEs in application areas encompassing robotics and learning-enabled (control) systems, the need for easy portability and extensibility to other popular software packages has necessitated our rewriting the toolbox in a single python package, leveraging GPU computations,

and covering the essential algorithms of Crandall [12], Osher and Fedkiw [39], and Mitchell [35]. Our contributions are as follows:

(1) we describe the details of the LevelSetPy software, starting with the common implicit surfaces that are used as initial conditions to represent $v(x, t)$ leveraging signed distance functions (SDF) and the associated boolean operations and calculus toolboxes for their *dynamics* endowment;
(2) we describe the upwinding spatial derivative schemes, temporal discretization via method of lines schemes based on (approximate) total variation diminishing (TVD) Runge-Kutta (RK), and stabilizing Lax-Friedrichs schemes for multidimensional monotone Hamiltonians of HJ equations or scalar conservation laws;
(3) we conclude this article with three representative examples, viz., (i) the barrier surface for two adversarial rockets traveling on a plane; (ii) the time-to-reach time-optimal control problem for a generic double integrator system; and (iii) analyze the barrier surface for a flock of birds traversing a phase space.

This article provides a description of the underlying theory and implementation of these hyperbolic PDE 's. First, in section 3, differential games in the context of dynamic programming and reachability theory are introduced. Second, in section 4, we describe the geometry of (and Boolean operations on) implicit function representations of continuous-time value functions described by (1) using Cartesian grids. Third, spatial derivatives to scalar conservation laws are described in section 5, and temporal discretization schemes for these conservation laws follow thereafter. Fourth, in section 7 we describe real-world problems and make them amenable to HJ PDE's and the construction of safe sets or tubes within a verification geometrical reasoning framework: we present numerical results on the motivated examples to demonstrate the efficacy of our software package on diverse representative problems that cover time-to-reach optimal control, and HJI differential games in reachability settings . Sixth, we conclude the paper in section 8. Additional examples, `jupyter` notebooks, and representative problems are provided in the online package.

3 DIFFERENTIAL GAMES AND REACHABILITY

In this section, we briefly describe reachability theory within the context of the HJ-Isaacs (HJI) equation [26]. Reachable sets in the context of two person games is introduced. We then establish the "viscuous" PDE to the terminal HJI PDE .

For a state $x \in \Omega$, fix: $T > t \geq 0$. Suppose that the controls for opposing players P and E in a two-player differential game are given by measurable functions

$$u : [t, T] \to \mathcal{U}, \qquad w : [t, T] \to \mathcal{W} \tag{4}$$

where $\mathcal{U} \in \mathbb{R}^m$ and $\mathcal{W} \subset \mathbb{R}^p$ are compact sets. Let us consider the differential equation,

$$\dot{x}(\tau) = f(\tau, x(\tau), u(t), w(t)), t \leq \tau \leq T, \quad x(t) = x, \tag{5}$$

where $f(\tau, \cdot, \cdot, \cdot)$ is uniformly continuous and $x(\cdot)$ is the unique solution of system (5). Suppose that we associate with the dynamical system (5) the payoff functional

$$\Phi(t; x, u, w) = \int_t^T l(\tau, x(\tau), u(\tau), w(\tau)) d\tau + g(x(T)), \tag{6}$$

where $g : \mathbb{R}^n \to \mathbb{R}$ satisfies

$$|g(x)| \leq k_1, \qquad |g(x) - g(\hat{x})| \leq k_1 |x - \hat{x}| \tag{7a}$$

and $l : [0, T] \times \mathbb{R}^n \times \mathcal{U} \times \mathcal{W} \to \mathbb{R}$ is the BUC stage cost. We say T is the *terminal time* (it may be infinity!) and the integral, when it does not depend on the control laws, is the *performance index*.

The evader is maximizing and pursuer is minimizing the payoff (6). In reachability analysis, the cost-to-go $\boldsymbol{l}(\cdot)$ is taken as zero so that the payoff functional (6) becomes

$$\Phi(t; \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) = \boldsymbol{g}(\boldsymbol{x}(T)). \tag{8}$$

3.1 HJI's Lower Value and its Viscosity Solution

Define

$$\bar{\mathcal{U}} \equiv \{\boldsymbol{u} : [t, T] \to \mathcal{U}\} \text{ and } \bar{\mathcal{W}} \equiv \{\boldsymbol{w} : [t, T] \to \mathcal{W}\} \tag{9}$$

as all control sets for players $\boldsymbol{P}$ and $\boldsymbol{E}$. For controls which agree almost everywhere (a.e.), we are interested in the pursuer's mapping strategy (starting at t) i.e.

$$\beta : \bar{\mathcal{U}}(t) \to \bar{\mathcal{W}}(t) \tag{10}$$

provided for each $t \leq \tau \leq T$ and $\boldsymbol{u}, \hat{\boldsymbol{u}} \in \bar{\mathcal{U}}(t)$; then $\boldsymbol{u}(\bar{t}) = \hat{\boldsymbol{u}}(\bar{t})$ a.e. on $t \leq \bar{t} \leq \tau$ implies $\beta[\boldsymbol{u}](\bar{t}) = \beta[\hat{\boldsymbol{u}}](\bar{t})$ a.e. on $t \leq \bar{t} \leq \tau$. The differential game's *lower value* for a solution $\boldsymbol{x}(t)$ of (5) for a $\boldsymbol{u}(t)$ and $\boldsymbol{w}(t) = \beta[\boldsymbol{u}](\cdot)$ is [18]

$$\begin{aligned} \boldsymbol{v}(\boldsymbol{x}, t) &= \inf_{\beta \in \mathcal{B}(t)} \sup_{\boldsymbol{u} \in \mathcal{U}(t)} \Phi(t, \boldsymbol{x}, \boldsymbol{u}, \beta[\boldsymbol{u}]) \\ &= \inf_{\beta \in \mathcal{B}(t)} \sup_{\boldsymbol{u} \in \mathcal{U}(t)} \boldsymbol{g}\left(\boldsymbol{x}(T)\right). \end{aligned} \tag{11}$$

We say $\boldsymbol{v}(\boldsymbol{x}, t)$ is the lower value function of the differential game. This value function is used in backward reachability analysis, the chief subject of our discourse.

LEMMA 1. *The lower value* $\boldsymbol{v}(\boldsymbol{x}, t)$ *in* (11) *is the viscosity solution to the lower Hamilton-Jacobi Isaac's equation* (1) *and it has with it the lower Hamiltonian,*

$$\boldsymbol{H}(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}, p) = \max_{\boldsymbol{u} \in \mathcal{U}} \min_{\boldsymbol{w} \in \mathcal{V}} \langle f(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}), p \rangle. \tag{12}$$

where p *is the co-state, i.e. the spatial derivative of* $\boldsymbol{v}$ *w.r.t* $\boldsymbol{x}$ *and* $\langle \cdot, \cdot \rangle$ *is the dot product operator between two variables.*

PROOF. This proof is given in [18]. □

3.2 Optimal Control, HJI PDE , and Reachability Theory

That which concerns point sets for a dynamical system's behavioral evolution (or trajectory) throughout a phase space[3] and such that the sets satisfy input or state constraints in a least restrictive sense and under a worst-possible disturbance [32] describes reachability. For computational and decidability reasons[4], we restrict our computation scheme over a finite time period. This is a classic reachability theory problem.

Reachability analysis is concerned with finding the set of forward (resp. backward) states that are reachable from a set of initial (resp. target) state sets, *up to a final time* under the worst possible disturbance. This verification problem usually consists in finding a *reachable states set* that lie along the trajectory of the solution to a first order nonlinear P.D.E. Typically, this solution originates from some initial phase $(\boldsymbol{x}_0, t_0)$ up to a state, $\boldsymbol{x}(T)$, at a specified final time T. Once computed, the optimal value function provides a safety certificate and controller defined by the spatial gradients of the value function.

[3]To avoid the cumbersome phrase "the state $\boldsymbol{x}$ at time t", we will associate the pair $(\boldsymbol{x}, t)$ with the *phase* of the system for a state $\boldsymbol{x}$ at time t. We associate the Cartesian product of Ω and the space $T = \mathbb{R}^1$ of all time values as the *phase space* of $\Omega \times T$.
[4]We say a reachability problem is decidable if we can compute all the states that can be reached from an initial condition in a finite number of steps.

In addition to producing a value function, the optimality principle of dynamic programming on the underlying HJI PDE can be used to generate a minimum time-to-reach (TTR) function. This function maps initial conditions to the minimum time horizon required to reach the target set. This can be computed by "stacking" the zero level sets of the value function as it propagates backwards in time [1, 3, 36].

Backward reachable sets (within a continuous system framework) are those subsets of a phase space whereupon trajectories initialized in some subset of the phase space can reach a so-called target set, $\mathcal{L}(x)$. Mitchell et. al' [37] essential contribution was that viscosity solution of the Cauchy-type HJI PDE (1) or (12) is tantamount to an implicit surface representation of the continuous-time backward reachable set. This formulation is amenable to modern large-scale problems because level set methods can be used in computing the target set as the viscosity solution is numerically continuous everywhere and well-posed. In backward reachability analysis, the ordering of time indices is reversed so that instead of having $t \in [0, T]$, we shall henceforth have $t \in [-T, 0]$.

3.3 The Backward Reachable (Target) Set

Suppose that the goal for an agent moving over a phase space is to reach a region at the end of a time horizon. We could leverage the terminal cost $g(\cdot)$ in (12) and impose constraints

$$|g(0; \boldsymbol{x})| \leq k, \ |g(0; \boldsymbol{x}) - g(t; \hat{\boldsymbol{x}}) \ | \leq k|\boldsymbol{x} - \hat{\boldsymbol{x}} \ | \tag{13}$$

for constant k and all $-T \leq t \leq 0$, $\hat{\boldsymbol{x}}, \ \boldsymbol{x} \in \mathbb{R}^n$. The set

$$\mathcal{L}_0 = \{\boldsymbol{x} \in \bar{\Omega} \,|\, g(0; \boldsymbol{x}) \leq 0\}, \tag{14}$$

is called the *target set* (otherwise referred to as the backward reachable set) in the phase space $\Omega \times \mathbb{R}$ (proof in [37]). This target set can represent the failure set (to avoid) or a goal set (to reach) in the state space. Note that the target set, $\mathcal{L}_0$, is a closed subset of $\mathbb{R}^n$ and is in the closure of Ω. Typically $\mathcal{L}_0$ is user-defined, and in level set methods, $g(x)$ is a signed distance function – negative inside the target set and positive elsewhere.

3.4 The Backward Reachable Tube

It is often desirable to consider all conditions under which trajectories of the system may enter a user-defined target set. This could be desirable in goal-regions of the phase space (safe sets) or undesirable configurations (unsafe sets).

For the safety problem setup in (11), we can define the corresponding *robustly controlled backward reachable tube* [36] as the closure of the open set

$$\begin{aligned}\mathcal{L}([\tau, 0], \mathcal{L}_0) = \{\boldsymbol{x} \in \Omega \,|\, \exists\, \beta \in \bar{\mathcal{W}}(t) \,\forall\, \boldsymbol{u} \in \mathcal{U}(t), \exists \\ \tau \in [-T, 0], \boldsymbol{\xi}(\bar{t}) \in \mathcal{L}_0\}.\end{aligned} \tag{15}$$

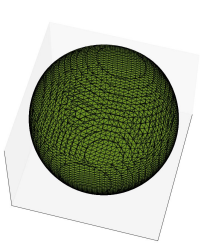
Read: The set of states from which the strategies of $\boldsymbol{P}$ and for all controls of $\boldsymbol{E}$ imply that we *reach and remain in the target set* in the interval $[-T, 0]$. Following Lemma 2 of [37], the states in the reachable set admit the following properties w.r.t the value function $\boldsymbol{v}$:

$$\boldsymbol{x}(t) \in \mathcal{L}(\cdot) \implies \boldsymbol{v}(\boldsymbol{x}, t) \leq 0, \tag{16a}$$

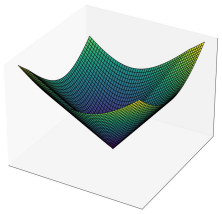
$$\boldsymbol{v}(\boldsymbol{x}, t) \leq 0 \implies \boldsymbol{x}(t) \in \mathcal{L}(\cdot). \tag{16b}$$

Player $\boldsymbol{P}$ is minimizing (the game's termination time c.f. (14)), seeking to drive system trajectories into the unsafe set; and player $\boldsymbol{E}$ is maximizing (the game's termination time) i.e. is seeking to avoid the unsafe set[5].

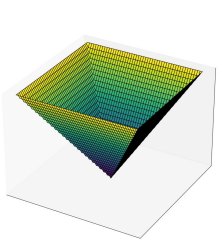
[5]For the goal-satisfaction (or *liveness*) problem setups, the strategies are reversed and the backward reachable tube are the states from which the evader $\boldsymbol{E}$ can successfully reach the target set despite worst-case efforts of the pursuer $\boldsymbol{P}$.



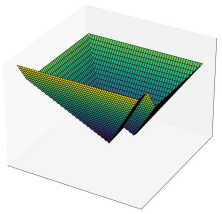
(a) Sphere



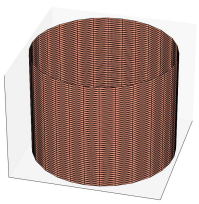
(b) Union of two spheres



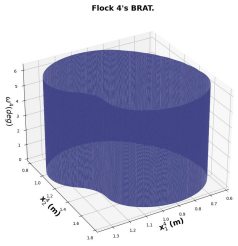
(c) Rectangle



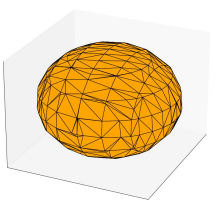
(d) Union of 2 rectangles



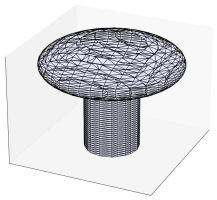
(e) Cylinder



(f) Union of six cylinders



(g) 3D Ellipsoid



(h) Sphere+Cylinder intersect.

Fig. 1. Examples of implicitly constructed zero levelsets of interfaces of geometric primitives along with Boolean operations on 2D and 3D Cartesian grids. Zero level set of **(a)** a sphere on a 3D grid; **(b)** union of two 3D spheres implicitly constructed on a 2D grid; **(c)** a rectangle on a 2D grid; **(d)** the union of rectangles on a 2D grid; **(e)** a cylinder on a 3D grid; **(f)** the union of multiple cylinders on a 3D grid; **(g)** an ellipsoid on a 3D grid; **(h)** Intersection of a sphere and a cylinder on a 3D grid.

4 GEOMETRY OF IMPLICIT SURFACES AND LAYOUTS

In this section, we discuss how implicit surface functions are constructed, stored on local memory and how they are transferred to GPUs. Throughout, links to `api's`, `routines`, and `subroutines` are highlighted in `blue text` (with a working hyperlink) and we use code snippets in Python to illustrate API calls when it's convenient.

At issue are co-dimension one implicitly defined surfaces on $\mathbb{R}^n$ which represent the interface of a flow, or function e.g. $f(x)$. These interfaces are more often than not the isocontours of some function. This representation is attractive since it requires less number of points to represent a function than explicit forms. Relating to the problems of chief interest in this article, the zero isocontour (or the zero levelset) of a reachability optimal control problem is equivalent to the safety set or backward reachable tube; and for a differential game, it is the *boundary of the useable part* of the *barrier surface* between the *capture* and *escape zones* for all trajectories that emanate from a system.

By implicitly representing functions $f(x)$ for points x e.g. in $\mathbb{R}^n$, we do not explicitly describe but rather imply the existence of f. This representation is attractive since it requires less number of points to represent a function than explicit forms. Let us describe the representation of data we employ in what follows.

4.1 Grids Layout

Fundamental to implicit surface representations are Cartesian grids in our library. Packages that implement 'grid' data structures are in the folder `Grids`. Grid $\mathbf{g}$ is `created` by specifying minimum, $\mathbf{g}_{min}$, and maximum axes bounds, $\mathbf{g}_{max}$, along every Cartesian coordinate axes n (see lines 3 and 4) of Listing 1; a desired number of discrete points N is passed to the grid data structure – specifying the

number of grid nodes and the grid spacing in each dimension as (line 5) listed in Listing 1. On line 5, a grid data structure is constructed and all input parameters to the API are checked for consistency.

```
1 from math import pi
2 import numpy as np
3 gmin = np.array((-5, -5, -pi)) // lower corner
4 gmax = np.array((5, 5, pi)) // upper corner
5 N = 41*ones(3, 1) // number of grid nodes
6 pdim = 3; // periodic boundary condition, dim 3
7 g = createGrid(gmin, gmax, N, pdim)
```

Listing 1. Creating a three-dimensional grid.

A grid `data structure`, **g**, (implemented in Listing 1) has the following fields: (i) discretized nodes of the state(s) x in (3), denoted as 1-D vectors $\mathbf{g}.vs$; (ii) given the 1-D vectors $\mathbf{g}.vs$, an n-dimensional array of coordinates over n-dimensional grids is computed with matrix-based indexing; this generates a mesh for all state nodal points on the grid $\mathbf{g}.xs$ as a list across all the dimensions of the grid; (iii) grid dimension $\mathbf{g}.dim$, denoting the number of Cartesian axes needed for representing the state x[6]; (iv) boundary conditions of the relavant HJ equation to be solved are grafted in by populating the corresponding grid dimension with ghost cells (to be introduced shortly).

4.2 Implicit Surface Representations: Levelsets

For an implicit surface representation of a geometric function, we treat the coordinates as functional arguments instead of functional values using a fixed level set of continuous function $v : \mathbb{R}^n \to \mathbb{R}$. We use signed distance functions to represent moving fronts throughout. When the signed distance function is not numerically possible, we describe where the implicit surface representations are smeared out in every routines' `documentation`.

The query points for moving interfaces are grid point sets of the computational domain described by implicit geometric primitives such as spheres, cylinders, ellipsoids and even polyhedrons such as icosahedrons. All of these are contained in the folder `InitialConditions` on our projects page.

Suppose that the zero levelset of an implicit surface $v(x,t)$ is defined as $\Gamma = \{x : v(x) = 0\}$ on a grid $G \in \mathbb{R}^n$, where n denotes the number of dimensions. Our representation of Γ on G generalizes a row-major layout. An example representation of an ellipsoid on a three-dimensional grid is illustrated in Listing 2.

```
1 e = (g.xs[0])**2 // ellipsoid nodal points
2 e += 4.0*(g.xs[1])**2
3 if g.dim==3:
4   data += (9.0*(grid.xs[2])**2)
5 e -= radius // radius=major axis of ellipsoid
```

Listing 2. An `ellipsoid` as a signed distance function.

4.3 Calculus on Implicit Function Representations

Geometrical operations on implicitly defined functions carries through in the package as follows.

Suppose that $v_1(x)$ and $v_2(x)$ are two signed distance representations, then the union of the interior of the two functions is simply $\min(v_1(x), v_2(x))$ (example illustrations in Fig. 1 a, c, d and h). The intersection of the interior of two signed distance functions is generated by $\max(v_1(x), v_2(x))$ (example illustrations in Fig. 1). The complement of a function is found by negating its signed distance function i.e. $-v(x)$. The resultant function as a result of the subtraction of the interior of one

[6]This parameter is useful when computing signed distance to every nodal point on the state space in the implicit representation of v

signed distance function v_2 from the another one, say, v_1 is defined $\max(v_1(x), -v_2(x))$. All of these are implemented in the module `shapeOps`.

- For example, if we desire to unite two separate interfaces defined along the nodal points of a grid constructed on $\mathbb{R}^n$, say a sphere and a rectangle, we would carry out an element-wise minimum of every point belonging to the two or more interfaces. This is what we do in `shapeUnion`.
- If we desire to intersect the two interfaces, we would carry out an element-wise maximum of every nodal point belonging to the two or more interfaces. We implement this in the routine `shapeIntersect`.
- Very often, we want to just subtract an interface' geometry from that of another. In this case, we negate the nodal points belong to the interface which we want to subtract, and then follow this operation with an element-wise maximum of all shapes. We implement this operation in `shapeDifference`.
- Another common operation on implicit interfaces is the need to find the complement set of an interface. In this instance, we simply negate the original function. This is in `shapeComplement`.

5 SPATIAL DISCRETIZATION: UPWINDING

Monotone solutions to the levelset equation, as introduced by Crandall and Lions [11] are attractive but they are at most first-order accurate and tend to be dissipative for most practical applications. In this section, we discuss higher-order upwinding schemes that mimic high-order essentially non-socillatory (ENO) schemes for computing the spatial derivatives v_x for the numerical viscosity solutions to levelset PDE's of the *Eulerian form* (introduced in (17)). Codebases for procedures herewith described are in the folder `SpatialDerivatives`.

Using the Eulerian form of the levelset equation,

$$v_t + \boldsymbol{F} \cdot \nabla v = 0 \tag{17}$$

where $\boldsymbol{F}$ is the speed functionI, the implicit function itself (section 4) is used both to denote and to evolve the interface. Suppose that the interface speed $\boldsymbol{F}$ is a three-vector $[f_x, f_y, f_z]$ on a three-dimensional Cartesian grid, expanding (17) the evolution of the implicit function on the zero levelset yields the Eulerian form

$$v_t + f_x v_x + f_y v_y + f_z v_z = 0 \tag{18}$$

of the interface evolution given that the interface encapsulates the implicit representation v. In our implementations, we define v throughout the computational domain Ω. However, narrow band methods [43] that only contain the interface can be implemented as well so that memory is saved while the front is being tracked. -

5.1 Upwinding

Let us first define the following differencing schemes

$$D^- v = \frac{\partial v}{\partial x} \approx \frac{v_{i+1} - v_i}{\Delta x}, D^+ v \approx \frac{v_i - v_{i-1}}{\Delta x}. \tag{19}$$

Suppose that v and its speed $\boldsymbol{F}$ are defined over a domain Ω (this is the Cartesian grid in our representation). Using the forward Euler method, the levelset equation (18) becomes

$$\frac{v^{n+1} - v^n}{\Delta t} + f_x^n v_x^n + f_y^n v_y^n + f_z^n v_z^n = 0. \tag{20}$$

Now, suppose that we are on a one-dimensional surface and around a grid point i, then given that f^n may be spatially varying the equation in the foregoing evaluates to

$$\frac{\boldsymbol{v}_i^{n+1} - \boldsymbol{v}_i^n}{\Delta t} + f_i^n (\boldsymbol{v}_x)_i^n = 0 \tag{21}$$

where $(\boldsymbol{v}_x)_i$ denotes the spatial derivative of $\boldsymbol{v}$ w.r.t x at the point i.

5.2 First-order accurate discretization

If $f_i > 0$, the values of $\boldsymbol{v}$ are traversing from left to right so that in order to update $\boldsymbol{v}$ at the end of the next time step, we must look to the left (going by the method of characteristics [39, §3.1]) and vice versa if $f_i < 0$. We therefore follow the standard upwinding method by using (19): we approximate $\boldsymbol{v}_x$ with $D^-\boldsymbol{v}$ whenever $f_i > 0$ and we approximate $\boldsymbol{v}_x$ with $D^+\boldsymbol{v}$ whenever $f_i < 0$. No approximation is needed when $f_i = 0$ since $f_i(\boldsymbol{v}_x)_i)$ vanishes. This discretization scheme is accurate within $O(\Delta x)$ given the first order accurate approximations $D^-\boldsymbol{v}$ and $D^+\boldsymbol{v}$. We have followed the naming convention in [35] and in our `SpatialDerivatives` folder, we name this function `upwindFirstFirst`.

5.3 ENO Polynomial Interpolation of Solutions

Using a divided difference table, essentially non-oscillatory (ENO) polynomial interpolation of the discretization [42] of the levelset equation are known to generate improved numerical approximations to $D^-\boldsymbol{v}$ and $D^+\boldsymbol{v}$.

Suppose that we choose a uniform mesh discretization Δx. Define the zeroth divided differences of $\boldsymbol{v}$ at the grid nodes i as

$$D_i^0 \boldsymbol{v} = \boldsymbol{v}_i, \tag{22}$$

and the first, second, and third order divided differences of $\boldsymbol{v}$ as the midway between grid nodes i.e.

$$D_{i+1/2}^1 \boldsymbol{v} = \frac{D_{i+1}^0 \boldsymbol{v} - D_i^0 \boldsymbol{v}}{\Delta x},\ D_i^2 \boldsymbol{v} = \frac{D_{i+1/2}^1 \boldsymbol{v} - D_{i-1/2}^1 \boldsymbol{v}}{2\Delta x},\ D_{i+1/2}^3 \boldsymbol{v} = \frac{D_{i+1}^2 \boldsymbol{v} - D_i^2 \boldsymbol{v}}{3\Delta x}. \tag{23}$$

Then, an essentially non-oscillating polynomial of the form

$$\boldsymbol{v}(x) = Q_0(x) + Q_1(x)!Q_2(x) + Q_3(x) \tag{24}$$

can be constructed. In this light, the backward and forward spatial derivatives of $\boldsymbol{v}$ w.r.t x at grid node i is found in terms of the derivatives of the coefficients $Q_i(x)$ in the foregoing i.e.

$$\boldsymbol{v}_x(x_i) = Q_1'(x_i) + Q_2'(x_i) + Q_3'(x_i). \tag{25}$$

Define $k = i - 1$ and $k = i$ for $\boldsymbol{v}_x^-$ and $\boldsymbol{v}_x^+$ respectively. Then the first order accurate polynomial interpolation is essentially

$$Q_1(x) = (D_{k+1/2}^1 \boldsymbol{v})(x - x_i),\ Q_1'(x_i) = D_{k+1/2}^1 \boldsymbol{v} \tag{26}$$

i.e. first-order upwinding.

We follow [39]'s recommendation in avoiding interpolating near large oscillations in gradients. Therefore, we choose a constant c such that

$$c = \begin{cases} D_k^2 \boldsymbol{v} & \text{if } |D_k^2 \boldsymbol{v}| \le |D_{k+1}^2 \boldsymbol{v}| \\ D_{k+1}^2 \boldsymbol{v} & \text{otherwise} \end{cases} \tag{27}$$

so that

$$Q_2(x) = c(x - x_k)(x - x_{k+1}), \quad Q_2'(x_i) = c(2i - 2k - 1)\Delta x \tag{28a}$$

is the second-order accurate upwinding solution for the polynomial interpolation. This is implemented as `upwindFirstENO2` in the `SpatialDerivatives` folder.

To obtain a third-order accurate solution, we choose $c^\star$ as follows

$$c^\star = \begin{cases} D^3_{k^\star+1/2} & \text{if } |D^3_{k^\star+1/2}\boldsymbol{v}| \le |D^3_{k^\star+3/2}\boldsymbol{v}| \\ D^3_{k^\star+3/2}\boldsymbol{v} & \text{if } |D^3_{k^\star+1/2}\boldsymbol{v}| > |D^3_{k^\star+3/2}\boldsymbol{v}|. \end{cases} \tag{29}$$

Whence, we have

$$Q_3(x) = c^\star(x - x_{k^\star})(x - x_{k^\star} + 1)(x - x_{k^\star} + 2) \tag{30a}$$

$$Q_3'(x_i) = c^\star(3(i - k^\star)^2 - 6(i - k^\star) + 2)(\Delta x)^2 \tag{30b}$$

for the third-order accurate correction to the approximated upwinding scheme (24). This is implemented as a routine in `upwindFirstENO3aHelper` and called as `upwindFirstENO3` in the `SpatialDerivatives` folder.

5.4 HJ Weighted Essentially Nonoscillatory Solutions

Here, we focus on weighted ENO (WENO) schemes with the same stencil as the third-order ENO scheme but with accuracy reaching as high as fifth-order in the smooth parts of the solution. Results here presented closely follow the presentation of Jiang and Peng in [28]. These WENO schemes approximate spatial derivatives at integer grid points as opposed to at half-integer grid values as we did in the ENO schemes in the previous section.

The third-order accurate ENO scheme essentially employs one of three substencils on a grid, namely $\{i-3, i-2, \cdots, i\}$, $\{i-2, i-1, \cdots, i+1\}$, and $\{i-1, \cdots, i+3\}$ on the stencils range $\{i-3, i-2, \cdots, i+3\}$ in calculating spatial derivatives for $\boldsymbol{v}$.

Suppose that the spatial derivative $\boldsymbol{v}_x$ is to be found using the left-leaning substencil: $\{i-3, i-2, \cdots, i\}$, then the third-order ENO scheme chooses one from

$$\boldsymbol{v}^{-,0}_{x,i} = \frac{1}{3}D^+\boldsymbol{v}_{i-3} - \frac{7}{6}D^+\boldsymbol{v}_{i-2} + \frac{11}{6}D^+\boldsymbol{v}_{i-1} \tag{31a}$$

$$\boldsymbol{v}^{-,1}_{x,i} = -\frac{1}{6}D^+\boldsymbol{v}_{i-2} + \frac{5}{6}D^+\boldsymbol{v}_{i-1} + \frac{1}{3}D^+\boldsymbol{v}_i \tag{31b}$$

$$\boldsymbol{v}^{-,2}_{x,i} = -\frac{1}{3}D^+\boldsymbol{v}_{i-1} + \frac{5}{6}D^+\boldsymbol{v}_i - \frac{1}{6}D^+\boldsymbol{v}_{i+1} \tag{31c}$$

where $\boldsymbol{v}^{-,p}_{x,i}$ denotes the third-order p'th substencil to $\boldsymbol{v}_x(x_i)$ for $p = 0, 1, 2$. The WENO approximation to $\boldsymbol{v}_x(x_i)$ leverages a convex weighted average of the three substencils so that

$$\boldsymbol{v}^-_{x,i} = w_0\boldsymbol{v}^{-,0}_{x,i} + w_1\boldsymbol{v}^{-,1}_{x,i} + w_2\boldsymbol{v}^{-,2}_{x,i}. \tag{32}$$

In smooth regions of the phase space, $w_0 = 0.1$, $w_1 = 0.6$, and $w_2 = 0.3$ yield the optimally accurate fifth order WENO approximation, we have for $\boldsymbol{v}^-_{x,i}$

$$\frac{1}{30}D^+\boldsymbol{v}_{i-3} - \frac{13}{60}D^+\boldsymbol{v}_{i-2} + \frac{47}{60}D^+\boldsymbol{v}_{i-1} + \frac{9}{20}D^+\boldsymbol{v}_i - \frac{1}{20}D^+\boldsymbol{v}_{i+1} \tag{33}$$

the fifth-order approximation $\boldsymbol{v}_x(x_i)$ and provides the smallest truncation error on a six-point stencil.

To account for weights in non-smooth regions, however, the smoothness of the stencils (31) can be estimated as recommended in [39, §3.4] so that if

$$\alpha_1 = 0.1/(\sigma_1 + \epsilon)^2,\ \alpha_2 = 0.6/(\sigma_2 + \epsilon)^2,\ \alpha_3 = 0.1/(\sigma_3 + \epsilon)^2 \tag{34}$$

for

$$\sigma_1 = \frac{13}{12}(D^+\boldsymbol{v}_{i-3} - 2D^+\boldsymbol{v}_{i-2} + D^+\boldsymbol{v}_{i-1})^2 + \frac{1}{4}(D^+\boldsymbol{v}_{i-3} - 4D^+\boldsymbol{v}_{i-2} + 3D^+\boldsymbol{v}_{i-1})^2, \tag{35a}$$

$$\sigma_2 = \frac{13}{12}(D^+\boldsymbol{v}_{i-2} - 2D^+\boldsymbol{v}_{i-3} + D^+\boldsymbol{v}_i)^2 + \frac{1}{4}(D^+\boldsymbol{v}_{i-2} - D^+\boldsymbol{v}_i)^2, \tag{35b}$$

$$\sigma_3 = \frac{13}{12}(D^+\boldsymbol{v}_{i-1} - 2D^+\boldsymbol{v}_i + D^+\boldsymbol{v}_{i+1})^2 + \frac{1}{4}(3D^+\boldsymbol{v}_{i-1} - 4D^+\boldsymbol{v}_i + D^+\boldsymbol{v}_{i+1})^2, \tag{35c}$$

and

$$\epsilon = 10^{-6}\max\{D^+\boldsymbol{v}_{i-3}, D^+\boldsymbol{v}_{i-2}, D^+\boldsymbol{v}_{i-1}, D^+\boldsymbol{v}_i D^+\boldsymbol{v}_{i+1}\} + 10^{-99} \tag{36}$$

then, we may define the weights for the WENO scheme as

$$w_1 = \alpha_1/\sum_{i=1}^{3}\alpha_i,\ \ w_2 = \alpha_2/\sum_{i=1}^{3}\alpha_i,\ \ w_3 = \alpha_3/\sum_{i=1}^{3}\alpha_i. \tag{37}$$

which well approximates the optimal weights $w_0 = 0.1$, $w_1 = 0.6$ and $w_2 = 0.3$ for decently smooth σ_k that can be dominated ny ϵ. This is implemented as a routine in `upwindFirstWENO5a` and called as `upwindFirstWENO5`.

5.5 Lax-Friedrichs Monotone Difference Schemes

We now describe a convergent monotone difference spatial approximation scheme for scalar conservation laws of the form

$$\begin{aligned} \boldsymbol{v}_t + \sum_{i=1}^{N} f_i(\boldsymbol{v})_{x_i} = 0 \text{ for } t > 0, \boldsymbol{x} = (x_1, \cdots, x_N) \in \mathbb{R}^n \\ \boldsymbol{v}(\boldsymbol{x}, 0) = \boldsymbol{v}_0(\boldsymbol{x}),\ \text{ for } \boldsymbol{x} \in \mathbb{R}^n \end{aligned} \tag{38}$$

Suppose that $N = 1$, let us define $\lambda_x = \Delta t/\Delta x$, $\Delta_x^+ = \boldsymbol{v}_{j+1} - v_j$, and $\Delta_x^- = \boldsymbol{v}_j - \boldsymbol{v}_{j-1}$. Then at the nth time step, the Lax-Friedrichs scheme is [12]

$$\boldsymbol{v}_j^{n+1} = \boldsymbol{v}_j^n - \frac{\lambda_x}{2}\Delta_x^0 f(\boldsymbol{v}_j^n) + \frac{1}{2}\Delta_x^+\Delta_x^-\boldsymbol{v}_j^n. \tag{39}$$

Furthermore, if we define the flux on the state space as

$$g(\boldsymbol{v}_j, \boldsymbol{v}_{j-1}) = \frac{f(\boldsymbol{v}_j) + f(\boldsymbol{v}_{j-1})}{2} - \frac{1}{2}\lambda_x(\boldsymbol{v}_j - \boldsymbol{v}_{j-1}), \tag{40}$$

we may write

$$\boldsymbol{v}_j^{n+1} = \boldsymbol{v}_j^n - \lambda_x^+(\boldsymbol{v}_j, \boldsymbol{v}_{j-1}). \tag{41}$$

The Lax-Friedrichs scheme is monotone on the interval $[a, b]$ if the CFL condition

$$\lambda_x \max_{a\le \boldsymbol{v}\le b}|f'(\boldsymbol{v})| \le 1 \tag{42}$$

for $(a, b) > 0$ and the upwind differencing scheme for a nondecreasing f is

$$\boldsymbol{v}_j^{n+1} = \boldsymbol{v}_j^n - \lambda_x\Delta_x^+ f(\boldsymbol{v}_{j-1}^n). \tag{43}$$

For a nonincreasing f, we have

$$\boldsymbol{v}_j^{n+1} = \boldsymbol{v}_j^n - \lambda_x\Delta_x^+ f(\boldsymbol{v}_j^n). \tag{44}$$

Our Lax-Friedrichs implementation is in the `link`.

6 TEMPORAL DISCRETIZATION: METHOD OF LINES

Here, we describe further improvements on the numerical derivatives of HJ equations by further improving the fifth order accurate HJ WENO schemes presented in section 5. We adopt the *method of lines* (MOL) used in converting the time-dependent PDE 's to ODE's.

In the MOL, the spatial derivatives in the PDE are replaced with algebraic approximations (in our case, one of the ENO schemes earlier presented) so that spatial derivatives no longer explicitly depend on spatial independent variables. Whence, only time, the initial value variable, is left so that we end up with a system of ordinary differential equations (ODEs) that closely approximate the original PDE.

Our presentation here follows the total variation diminishing (TVD) Runge Kutta (RK) schemes with Courant-Friedrichs-Lewy (CFL) conditioning imposed for stability as presented in [41] and implemented in `MATLAB®` in [35].

6.1 Higher-Order TVD-RK Time Discretizations

To adopt the method of lines, the N-dimensional levelset representation of v is first rolled into a 1-D vector and an adaptive integration step size, Δt, is chosen to guarantee stability following the recommendation in [45]. The forward Euler algorithm thus becomes

$$v(x, t + \Delta t) = v(x, t) + \Delta t \Upsilon(x, v(x, t)) \tag{45}$$

where Υ is now the function to be integrated.

A standard MOL can then be applied for the integration similar to ODEs (we have followed [35]'s code layout to provide consistency for MATLAB users). Since the details of the implementation are described in [35], we here describe the function arguments and describe call signatures.

We implement TVD-RK MOL schemes up to third-order accurate forward Euler integration schemes and the calling signature is as described in Listing 3.

```
odeCFLx(schemeFunc, tspan, y0, options, schemeData)
```

Listing 3. CFL-constrained method of lines routines.

where x could be one of 1, 2, or 3 to indicate first-order accurate, second-order accurate, or third-order accurate TVD-RK scheme. `schemeFunc` is typically a one of the Lax-Friedrichs approximation routines (implemented as `termLaxFriedrichs`) in the folder `ExplicitIntegration/Term` that approximates the HJ equation based on dissipation functions (to be shortly introduced).

The first-order accurate TVD (it is total variation bounded [TVB] actually) together with the spatial discretization used for the PDE is equivalent to the forward Euler method. We implement this as `odeCFL1`.

The second-order accurate TVD-RK scheme follows the RK scheme by evolving the Euler step to $t^n + \Delta t$,

$$\frac{v^{n+1} - v^n}{\Delta t} + F^n \cdot \nabla v^n = 0. \tag{46}$$

A following Euler step to $t^n + 2\Delta t$ follows such that

$$\frac{v^{n+2} - v^{n+1}}{\Delta t} + F^{n+1} \cdot \nabla v^{n+1} = 0 \tag{47}$$

before a convex combination of the initial value function and the result of the preceding Euler steps is taken in the following averaging step

$$v^{n+1} = \frac{1}{2}\{v^n + v^{n+2}\}. \tag{48}$$

The equation in the foregoing produces the second-order accurate TVD approximation to v at $t^n + \Delta t$, implemented as `odeCFL2`.

With the third-order accurate TVD-RK scheme, the first two advancements in forward Euler schemes are computed but with a different averaging scheme

$$v^{n+1/2} = \frac{1}{4}\{3v^n + v^{n+2}\} \tag{49}$$

which averages the previous two solutions at $t^n + \frac{1}{2}\Delta t$. The third Euler advancement step to $t^n + \frac{3}{2}\Delta t$ is

$$\frac{v^{n+\frac{3}{2}} - v^{n+\frac{1}{2}}}{\Delta t} + F^{n+\frac{1}{2}} \cdot \nabla v^{n+\frac{1}{2}} = 0 \tag{50}$$

together with the averaging scheme

$$v^{n+1} = \frac{1}{3}\{v^n + 2v^{n+\frac{3}{2}}\} \tag{51}$$

to produce a third-order accurate approximation to v at time $t^n + \Delta t$, implemented as `odeCFL3`.

7 EXAMPLES AND NUMERICAL EXPERIMENTS

In this section, we will present problems motivated by real-world scenarios and amend them to HJ PDE forms where their numerical solutions can be resolved with our `LevelSetPy` toolbox. The problems that we consider belong in `transport`, `differential games`, and `time-to-reach` problem classes. The library has been tested on numerous problems; however, for the sake of brevity we will only report a few results.

For the *differential games*, we do not necessarily analyze a single game, but rather a *collection/-family of games*, $\Upsilon = \{\Gamma_1, \cdots, \Gamma_g\}$. Each game within a differential game may be characterized as a pursuit-evasion *game*, Γ. Such a game terminates when *capture* occurs, that is the distance between players falls below a predetermined threshold. Each player in a game shall constitute either a pursuer ($\boldsymbol{P}$) or an evader ($\boldsymbol{E}$). Let the cursory reader not interpret $\boldsymbol{P}$ or $\boldsymbol{E}$ as controlling a single agent. In our various numerical experiments, we are poised with one or several pursuers (enemies) or evaders (peaceful citizens). However, when $\boldsymbol{P}$ or $\boldsymbol{E}$ governs the behavior of but one agent, these symbols will denote the agent itself. The nucleolus of our illustrative examples is to geometrically (approximately) ascertain the separation between the CZ and EZ surfaces i.e. the *barrier hypersurface*, where starting points exist for which escape occurs, capture occurs, and for which the outcome is neutral.

To address our desiderata, we must settle upon how best should $\boldsymbol{P}$ pursue $\boldsymbol{E}$. Here, at every time instant, $\boldsymbol{P}$ possesses knowledge of his own and that of $\boldsymbol{E}$'s position so that $\boldsymbol{P}$ knows how to regulate its various controlling variables with respect to $\boldsymbol{E}$'s motion in an optimal fashion. This follows the mathematical layout in section 3. The task is to assay the *game of kind* for the envelope of the capturable states i.e. we are not so much as seeking a game's outcome as we are seeking the conditions under which capture can occur. This introduces the *barrier* hypersurface which separates, in the initial conditions space, the hypersurface of capture from those of escape. In this *game of kind* postulation, all optimal strategies are not unique, but rather are a *legion*. Ergo, we are concerned with the set of initial positions on the vectogram where the capture zone (CZ) exists i.e. where game termination occurs; and the nature of escape zones (EZ) i.e. zones where termination or escape does not occur – after playing the differential game.

The rest of this section introduces different representative examples where real-world problems are adopted and amended to HJ PDE forms and whose solution we seek to numerically recover. Space here has limited us to four illustrative examples: First, we present two rockets in a pursuit-evasion game where the goal is for the evader to guard a territory and the pursuer's goal is to penetrate the

boundary of the territory being guarded. Second, we describe a double integral dynamical system: the double integral plant is a simplified abstraction of many real-world force-control system e.g. those that obey Newton's second law of motion or the torque-inertia dynamics of a body with rotary dynamics. We provide the numerical enumeration of the solution to the analytical time to reach problem. Third, we describe a collective behavior system in natural starlings and we provide a mathematical abstraction that allows the computation of the collision-avoidance *safety envelope* that may then be used in e.g. runtime assurance (RTA) safety-critical controller [24]. Fourth, we compare the resolution of the (approximately) capturable sets in Dubins' game of two identical vehicles with the levelsets toolbox's solution. RTA controllers act intelligently as a safety system between a real-world controller and the system to be controlled by providing a state monitoring scheme useful in intervening in the real-world where vulnerabilities to danger is a constant factor to be mitigated against. All the examples presented in this section can be found in the `Examples` folder of our online library.

7.1 Two Rockets in a Pursuit-Evasion Terminal Value Differential Game

We adopt the rocket launch problem of Dreyfus [15] and amend it to a differential game between two identical rockets, $\boldsymbol{P}$ and $\boldsymbol{E}$, on an (x, z) cross-section of a Cartesian plane. We set out to compute the *useable part* of the boundary of the *approximate* terminal surface of a predefined target set over a time horizon (i.e. the target tube). The useable part entails the regions of the state-space for which the min-max operation over either strategy of $\boldsymbol{P}$ and $\boldsymbol{E}$ is below 0. The boundary of the useable part (BUP) constitute where the variational HJI PDE is exactly zero.

The BUP, *target tube*, or in modern parlance backward reachable tube (BRT) shall be implicitly constructed with our LevelSetPy library as the zero-level set of an implicitly defined function over the entire state space. At the zero level set, resolving the *kinematic equation* of the rockets in relative coordinates helps us understand the nature of the barrier hypersurface. Specifically, the target tube is a terminal surface that enunciates the set of initial starting points for which termination (capture or **C**), no termination (escape or **E**) of a game does occur, or analyzing the barrier separating **C** or **E** after playing the differential game.

A single rocket's motion is dictated by the following system of differential equations (under Dreyfus' assumptions):

$$\dot{x}_1 = x_3, \qquad x_1(t_0) = 0; \tag{52a}$$
$$\dot{x}_2 = x_4, \qquad x_2(t_0) = 0; \tag{52b}$$
$$\dot{x}_3 = a\cos u, \qquad x_3(t_0) = 0; \tag{52c}$$
$$\dot{x}_4 = a\sin u - g, \qquad x_4(t_0) = 0; \tag{52d}$$

where, (x_1, x_2) are respectively the horizontal and vertical range of the rocket (in feet); (x_3, x_4) are respectively the horizontal and vertical velocities of the rocket (in feet per second); and a and g are respectively the acceleration and gravitational accelerations (in feet per square second).

We now make the problem amenable to a two-player differential game. Let rockets $\boldsymbol{P}$ and $\boldsymbol{E}$ share identical dynamics in a general sense. The coordinates of $\boldsymbol{P}$ are freely chosen; however, the coordinates of $\boldsymbol{E}$ are chosen a distance $\boldsymbol{\phi}$ away from (x, z) at the origin of the plane (as illustrated in Fig. 2) so that the $\boldsymbol{PE}$ vector's inclination measured counterclockwise from the $\boldsymbol{x}$ axis is θ.

Being a free endpoint problem, let the states of $\boldsymbol{P}$ and $\boldsymbol{E}$ be denoted by $(\boldsymbol{x}_p, \boldsymbol{x}_e)$. Furthermore, let the rockets be driven by their thrusts, denoted by (u_p, u_e) for $\boldsymbol{P}$ and $\boldsymbol{E}$ respectively (see Figure 2). Fix the range of the rockets so that what is left of the motion of either $\boldsymbol{P}$ or $\boldsymbol{E}$'s is restricted to orientation on the (x, z) plane as illustrated in Fig. 2. Whence, the relevant k*inematic equations* (KE)

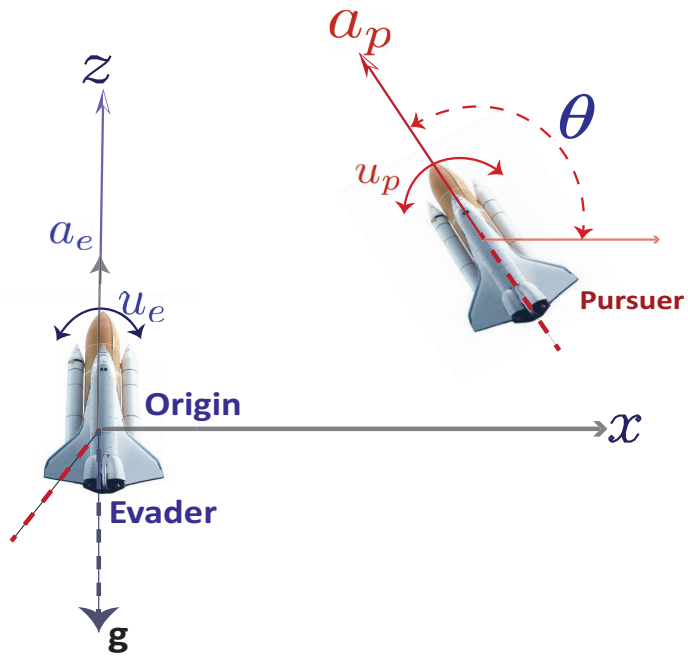


Fig. 2. Motion of two rockets on a Cartesian xz-plane with a thrust inclination in relative coordinates given by $\theta := u_p - u_e$.

from equation (52) are

$$\dot{x}_{2e} = x_{4e}; \qquad \dot{x}_{2p} = x_{4p}, \tag{53a}$$

$$\dot{x}_{4e} = a \sin u_e - g; \qquad \dot{x}_{4p} = a \sin u_p - g \tag{53b}$$

where a and g are respectively the acceleration and gravitational accelerations (in feet per square second) [7].

We want to determine the outcome of a simulated game between the two agents over a time interval. In the process of this protracted simulation, the nature of the barrier surface (henceforth called the backward reachable tube [37] or BRT[8]) will change.

Our desideratum is determining if capture can be achieved at all in a "yes-or-no" fashion. Therefore, we pose the game over a finite range over outcomes so that the game at hand assumes Isaac's [26] description of *a game of kind*. $\boldsymbol{P}$ can achieve as much proximity to a given *target set* as much as possible while $\boldsymbol{E}$ is set up to protect the *target set*. For example, one may take $\boldsymbol{P}$ as seeking to penetrate a (closed) territory (called *target*) under guard by player $\boldsymbol{E}$; and $\boldsymbol{P}$'s goal may be to maximize the time of play so as to penetrate the barrier surface of the target. $\boldsymbol{E}$ seeks to protect a given target's surface. As long as $\boldsymbol{E}$ remains within this *backward reachable tube* (or BRT), $\boldsymbol{P}$ cannot cause damage or exercise an action of deleterious consequence on, say, the territory being guarded by $\boldsymbol{E}$.

Setting up $\boldsymbol{E}$ to maximize the payoff quantity (55) with the largest possible margin or at least frustrate the efforts of $\boldsymbol{P}$ with minimal collateral damage while the pursuer minimizes the payoff quantity constitutes a terminal value *optimal* differential game: there is no optimal pursuit without

[7]We set $a = 1ft/sec^2$ and $g = 32ft/sec^2$ in our simulation.
[8]It is called backward because the game is simulated backward in time.

an optimal evasion since $\boldsymbol{P}$ and $\boldsymbol{E}$ are both executing motions as they see fit within the problem parameters.

Therefore, we rewrite (52) with $\boldsymbol{P}$'s motion relative to $\boldsymbol{E}$'s along the $(\boldsymbol{x}, z)$ plane so that the relative orientation as shown in Fig. 2 is $\theta = u_p - u_e$ – this shall serve as the control input. Following the conventions in Fig. 2, the game's relative equations of motion in *reduced space* [26, §2.2] i.e. is $\boldsymbol{x} = (x, z, \theta)$ where $\theta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right)$ and $(x, z) \in \mathbb{R}^2$ are

$$\dot{\boldsymbol{x}} = \begin{cases} \dot{x} &= a_p \cos\theta + u_e x, \\ \dot{z} &= a_p \sin\theta + a_e + u_e x - g, \\ \dot{\theta} &= u_p - u_e. \end{cases} \tag{54}$$

The boundary of the *usable part* of the origin-centered circle of radius $\boldsymbol{\phi}$ (we have set $\boldsymbol{\phi} = 1.5$ feet in our evaluations) is $\|\boldsymbol{PE}\|_2$ so that

$$\boldsymbol{\phi}^2 = x^2 + z^2. \tag{55}$$

All capture points are specified by the variational HJ PDE [37]:

$$\frac{\partial \boldsymbol{\phi}}{\partial t}(\boldsymbol{x}, t) + \min\left[0, \boldsymbol{H}(\boldsymbol{x}, \frac{\partial \boldsymbol{\phi}(\boldsymbol{x}, t)}{\partial \boldsymbol{x}})\right] \leq 0, \tag{56}$$

with Hamiltonian given by

$$\boldsymbol{H}(\boldsymbol{x}, p) = -\max_{u_e \in [\underline{u}_e, \bar{u}_e]} \min_{u_p \in [\underline{u}_p, \bar{u}_p].} \begin{bmatrix} p_1 & p_2 & p_3 \end{bmatrix} \begin{bmatrix} a_p \cos\theta + u_e x \\ a_p \sin\theta + a_e + u_p x - g \\ u_p - u_e \end{bmatrix} \tag{57}$$

Here, the co-states p is defined with a strict corresponding property, and $[\underline{u}_e, \bar{u}_e]$ denotes the extremals that the evader must choose as input in response to the extremal controls that the pursuer plays i.e. $[\underline{u}_p, \bar{u}_p]$.

We must consider the possibilities of behavior by either agent in an all-encompassing fashion in order to know what an outcome may be in the future should either agent execute different controls. Rather than resort to analytical *geometric reasoning*, we may analyze this *game* via a principled numerical simulation. This is what we present next. From (57), set $\underline{u}_e = \underline{u}_p = \underline{u} \triangleq -1$ and $\bar{u}_p = \bar{u}_e = \bar{u} \triangleq +1$ so that

$$\begin{aligned} \boldsymbol{H}(\boldsymbol{x}, p) &= -\max_{u_e \in [\underline{u}_e, \bar{u}_e]} \min_{u_p \in [\underline{u}_p, \bar{u}_p]} \left[p_1(a_p \cos\theta + u_e x) + p_2(a_p \sin\theta + a_e + u_p x - g) + p_3(u_p - u_e)\right], \\ &= -a_p p_1 \cos\theta - a p_2 \sin\theta - a p_2 + g p_2 - \max_{u_e \in [\underline{u}_e, \bar{u}_e]} \min_{u_p \in [\underline{u}_p, \bar{u}_p]} \left(p_1 u_e + p_2 u_p x + p_3(u_p - u_e)\right), \\ &= -a_p p_1 \cos\theta - a p_2 \sin\theta - a p_2 + g p_2 - \bar{u}|p_1 x + p_3| + \underline{u}|p_2 x + p_3|, \\ &\triangleq -a p_1 \cos\theta - p_2(g - a - a\sin\theta) - \bar{u}|p_1 x + p_3| + \underline{u}|p_2 x + p_3|, \end{aligned} \tag{58}$$

where the last line in (58) follows from setting $a_e = a_p \triangleq a$.

For the target set being guarded by $\boldsymbol{E}$, we choose an implicit representation with a cylindrical mesh on a three-dimensional grid as our representation. The grid's nodes are uniformly spaced apart at a resolution of 100 points per dimension over the interval $[-64, 64]$. In numerically solving for the Hamiltonian (58), a TVD-RK discretization scheme [42] based on fluxes is used in choosing smooth nonoscillatory results as described in §6. Denote by (x, y, z) a generic point in $\mathbb{R}^3$ so that given mesh sizes Δx, Δy, Δz, $\Delta t > 0$, letters u, v, w represent functions on the x, y, z lattice:

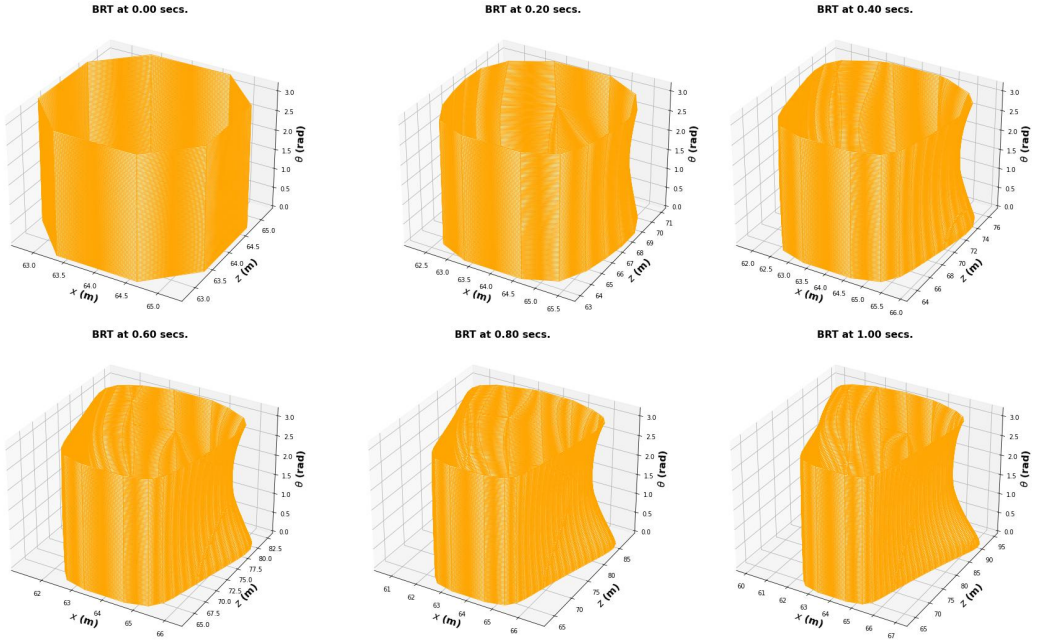


Fig. 3. *(Left to Right)*: Backward reachable tubes (capture surfaces) for the rocket system (cf. Fig. 2) optimized for the paths of slowest-quickest descent in equation (57) at various time steps during the differential game. In all, the BRTs were computed using the method outlined in [9, 36, 39]. We set $a_e = a_p = 1ft/sec^2$ and $g = 32ft/sec^2$ as in Dreyfus' original example.

$\Delta = \{(x_i, y_j, z_k) : i, j, k \in \mathbb{Z}\}$. We define the numerical monotone flux $\hat{\boldsymbol{H}}(\boldsymbol{x}, p)$, of $\boldsymbol{H}(\boldsymbol{x}, p)$ as

$$\begin{aligned}\hat{\boldsymbol{H}}(u^+, u^-, v^+, v^-, w^+, w^-) &= \boldsymbol{H}\left(\frac{u^+ + u^-}{2}, \frac{v^+ + v^-}{2}, \frac{w^+ + w^-}{2}\right), \\ &= -\frac{1}{2}\left[\alpha_x^{(i)_j}\left(u^+ - u^-\right) + \alpha_y^{(i)_j}\left(v^+ - v^-\right) + \alpha_z^{(i)_j}\left(w^+ - w^-\right)\right], \end{aligned} \tag{59}$$

where

$$\alpha_x = \max_{\substack{a\le u\le b\\ c\le v\le d\\ e\le w\le f}} |\boldsymbol{H}_u(\cdot)|,\ \alpha_y = \max_{\substack{a\le u\le b\\ c\le v\le d\\ e\le w\le f}} |\boldsymbol{H}_v(\cdot)|, \quad \text{and } \alpha_z = \max_{\substack{a\le u\le b\\ c\le v\le d\\ e\le w\le f}} |\boldsymbol{H}_w(\cdot)| \tag{60}$$

are the dissipation coefficients, controlling the level of numerical viscosity in order to realize a stable solution that is physically realistic [11]. Here, the subscripts of $\boldsymbol{H}$ are the partial derivatives w.r.t the subscript variable, and the flux, $\hat{\boldsymbol{H}}(\cdot)$ is monotone for $a \le u^{\pm} \le b, c \le v^{\pm} \le d, e \le w^{\pm} \le f$. It is easy to very from (58) that

$$\alpha_x = |a\cos\theta| + u|x|,\ \alpha_y = |g - a - a\sin\theta| + \underline{u}|x|, \quad \text{and } \alpha_z = |\bar{u}| + |\underline{u}. \tag{61}$$

```
finite_diff_data = {"innerFunc": termLaxFriedrichs,
  "innerData": {"grid": g, "hamFunc": rocket_rel.ham,
  "partialFunc": rocket_rel.dissipation,
  "dissFunc": artificialDissipationGLF,
  "CoStateCalc": upwindFirstENO2},
  "positive": True} // direction of approx. growth
```

Listing 4. HJ ENO2 computational scheme for the rockets.

The Hamiltonian, upwinding scheme, flux dissipation method, and the overapproximation parameter for the essentially non-oscillatory polynomial interpolatory data used in geometrically reasoning about the *target set* is set up as seen in Listing 4. The data structure `finite_diff_data` contains all the routines needed for adding dynamics to the original implicit surface representation of $\boldsymbol{v}(\boldsymbol{x}, t)$. The monotone spatial upwinding scheme used (here `termLaxFriedrichs` described in §5.5) is passed into the `innerFunc` query field. The explicit form of the Hamiltonian (see (58)) is passed to the `hamFunc` query field and the grid described in the foregoing is passed to the `grid` field. We adopted a second-order accurate upwinding scheme together with the `Lax-Friedrichs` approximator. To indicate that we intend to overapproximate the value function, we specify a `True` parameter for the `positive` query field.

Safety is engendered by having the evader respond optimally to the pursuer at various times during the game. We are thus interested in the entire safety set over the time interval of play (i.e. the safety tube). The backward reachable tube (BRT) [37], under the control strategies of $\boldsymbol{P}$ or $\boldsymbol{E}$, is a part of the phase space that constitutes $\Omega \times T$. We would like the BRT to cover as much of the entire phase space as possible. Thus, we *overapproximate* it. Using our GPU-accelerated levelset toolbox, we compute the *overapproximated* BRT of the game over a time span of $[-2.5, 0]$ seconds over 11 global optimization time steps. The BRTs at representative time steps in the optimization procedure is depicted in Fig. 3.

The initial value function (leftmost inset of Fig. 3) is represented as a (closed) dynamic implicit surface over all point sets in the state space (using a signed distance function) for a coordinate-aligned cylinder whose vertical axes runs parallel to the orientation of the rockets depicted in Fig. 2. This closed and bounded assumption of the target set is a prerequisite of the backward reachable analysis (see [37]). It allows us to include all limiting velocities The two middle capture surfaces indicate the evolution of the capture surface (here the zero levelset) of the target set upon the optimal response of the evader to the pursuer. We reach convergence at the eleventh global optimization timestep (rightmost inset of Fig. 3).

Reachability theory thus affords us an ability to numerically reason about the behavior of these two rockets aforetime in a principled manner. To do this, we have passed relevant parameters to the package as shown in Listing 4 and run a CFL constrained optimization scheme (as in Listing 3) for a finite number of global optimization timesteps. It is global because internally, there are other local spatial and temporal finite differencing scheme that occurs “under the hood" (see 5 and the corresponding codes described).

7.2 Time Optimal Control: The Double Integral Plant

Here, we analyze a time-optimal control problem to determine what admissible control[9] can “transport" the system under consideration to a desired “origin" in the shortest possible time. We consider the double integral plant [5, 46] as an illustrative example of our objective, which is to compute the points in the state space that can reach the origin in *finite-time* under the influence of a time-optimal controller.

[9]A control law is admissible when its range belongs in the admissible input set where it is bounded.

We shall leverage standard necessary conditions from the principle of optimality [4] to obtain a time-optimal feedback control design; introduce the notion of isochrones and switching surfaces; and discuss the analytic and approximate solutions (with our library) to the time-optimal control problem for a double integrator. We shall conclude the section by comparing the analytic and the overapproximated numerical solution (using the LevelSetPy toolbox) to the time to reach the origin problem.

7.2.1 Dynamics and Problem Setup. The double integrator is controllable, so that open-loop strategies may be employed in driving specific states to the origin in finite time [46]. The plant has the following second-order dynamics

$$\ddot{\boldsymbol{x}}(t) = \boldsymbol{u}(t) \tag{62}$$

and admits bounded control signals $\mid \boldsymbol{u}(t) \mid \leq 1$ for all time t. After a change of variables,we have the following system of first-order differential equations

$$\begin{aligned} \dot{\boldsymbol{x}}_1(t) &= \boldsymbol{x}_2(t), \\ \dot{\boldsymbol{x}}_2(t) &= \boldsymbol{u}(t), \quad \mid \boldsymbol{u}(t) \mid \leq 1. \end{aligned} \tag{63}$$

The *reachability problem that we consider is to address the question of what states can reach a certain point (here, the origin) in a transient manner*. That is, we would like to find point sets on the state space, at a particular time step, such that we can bring the system to the equilibrium, $(0, 0)$.

7.2.2 Time-optimal control scheme. This is an $\boldsymbol{H}$-minimal control problem whereupon we must find the control law that minimizes the Hamiltonian

$$\boldsymbol{H}(\boldsymbol{x}, p) = p_1 \dot{\boldsymbol{x}}_1 + p_2 \dot{\boldsymbol{x}}_2. \tag{64}$$

The necessary optimality condition stipulates that the minimizing control law be

$$\boldsymbol{u}(t) = -\operatorname{sign}\left(p_2(t)\right) \triangleq \pm 1. \tag{65}$$

For the co-states in question, suppose that their initial values (for constants k_1 and k_2) are $p_1(t_0) = k_1$ and $p_2(t_0) = k_2$, only four candidates can serve as time-optimal control sequences i.e. $\{[+1], [-1], [+1, -1], [-1, +1]\}$. On a finite time interval, $t \in [t_0, t_f]$, the time-optimal $\boldsymbol{u}(t)$ is a constant $k \equiv \pm 1$ so that for initial conditions $\boldsymbol{x}_1(t_0) = \xi_1$ and $\boldsymbol{x}_2(t_0) = \xi_2$, it can be verified that the state trajectories obey the relation

$$\boldsymbol{x}_1(t) = \xi_1 + \frac{1}{2} k \left(\boldsymbol{x}_2^2 - \xi_2^2\right), \text{ for } t = k\left(\boldsymbol{x}_2(t) - \xi_2\right). \tag{66}$$

The trajectories of (66) traced out over a finite time horizon $t = [-1, 1]$ with *piecewise constant control laws*, $u = \pm 1$ on a state space and under the control laws $\boldsymbol{u}(t) = \pm 1$ is depicted in Fig. 4. Curves with arrows that point upwards denote trajectories under the control law $\boldsymbol{u} = +1$; call these trajectories $\boldsymbol{\gamma}_+$; while the trajectories marked by dashed arrows pointing downward on the curves were executed under $\boldsymbol{u} = -1$; call these trajectories $\boldsymbol{\gamma}_-$.

7.2.3 Analytic Time to Reach the Origin. The *time to reach the origin* $(0, 0)$ from any other pair (x_1, x_2) on the state plane of Fig. 4 *in the shortest possible time* is our approximation problem. This minimum time admits an analytical solution given by [1]

$$t^\star(\boldsymbol{x}_1, \boldsymbol{x}_2) = \begin{cases} \boldsymbol{x}_2 + \sqrt{4\boldsymbol{x}_1 + 2\boldsymbol{x}_2^2} & \text{if } \boldsymbol{x}_1 > \frac{1}{2}\boldsymbol{x}_2|\boldsymbol{x}_2| \\ -\boldsymbol{x}_2 + \sqrt{-4\boldsymbol{x}_1 + 2\boldsymbol{x}_2^2} & \text{if } \boldsymbol{x}_1 < -\frac{1}{2}\boldsymbol{x}_2|\boldsymbol{x}_2| \\ |\boldsymbol{x}_2| & \text{if } \boldsymbol{x}_1 = \frac{1}{2}\boldsymbol{x}_2|\boldsymbol{x}_2|. \end{cases} \tag{67}$$

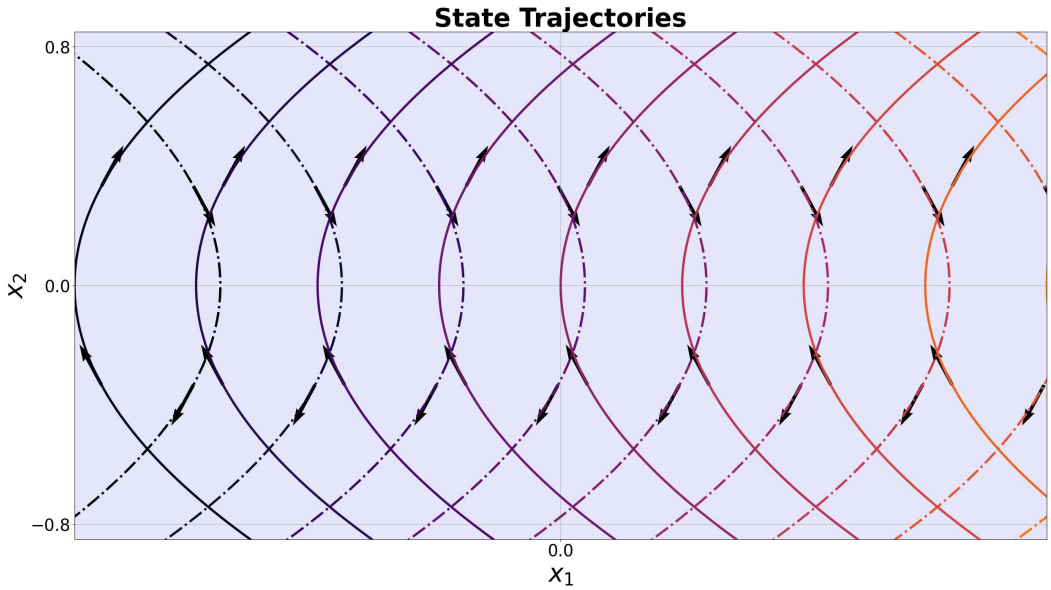


Fig. 4. State trajectories of the double integral plant. The solid curves are trajectories generated for $\boldsymbol{u} = +1$ while the dashed curves are trajectories for $\boldsymbol{u} = -1$.

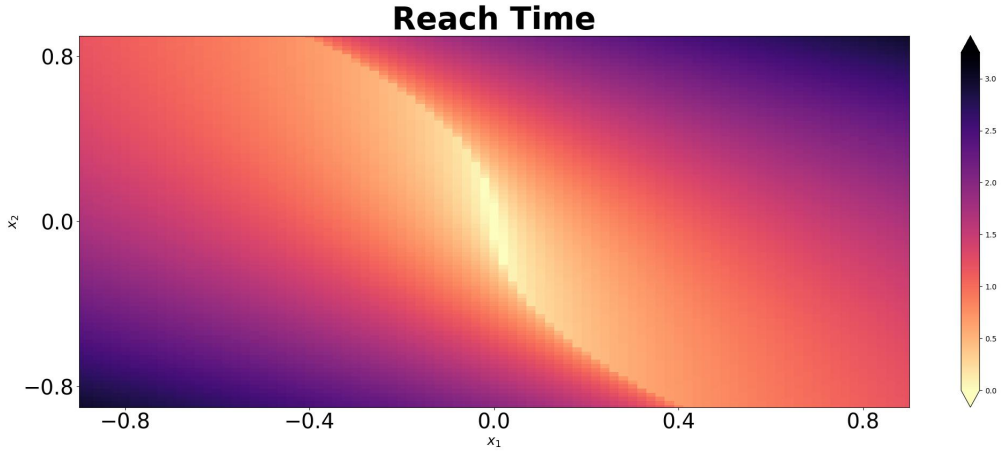
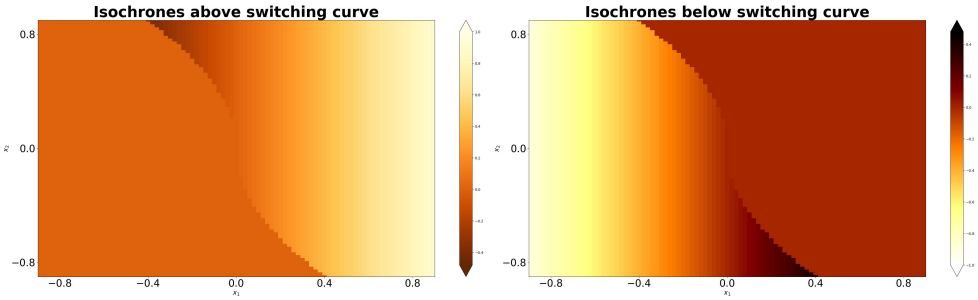


Fig. 5. Analytical time to reach the origin on the state grid, $(\mathbb{R} \times \mathbb{R})$; the switching curve, $\boldsymbol{\gamma} = \boldsymbol{\gamma}_- \cup \boldsymbol{\gamma}_+$, passes through states on $(0, 0)$.

The geometry (phase portrait) of (67) is shown in Fig. 5. Let us define $\boldsymbol{\gamma}_+$ as the locus of all points (x_1, x_2) which can be forced to the origin by $u = +1$ i.e.

$$\boldsymbol{\gamma}_+ = \{(x_1, x_2) : x_1 = \frac{1}{2}x_2^2;\ x_2 \leq 0\} \tag{68}$$



(a) Isochrones above $\boldsymbol{\gamma}$ *(b)* Isochrones below $\boldsymbol{\gamma}$

Fig. 6. (a) Isochrones for states above the switching curve, (b) states below the switching curve.

and let $\boldsymbol{\gamma}_-$ be the locus of all points (x_1, x_2) which can be forced to the origin by $u = -1$ i.e.

$$\boldsymbol{\gamma}_- = \{(x_1, x_2) : x_1 = \frac{1}{2}x_2^2;\ x_2 \geq 0\}. \tag{69}$$

The confluence of the locus of points on $\boldsymbol{\gamma}_+$ and $\boldsymbol{\gamma}_-$ is the *switching curve*, depicted in bright orange in Fig. 5, is

$$\boldsymbol{\gamma} \triangleq \boldsymbol{\gamma}_+ \cup \boldsymbol{\gamma}_- = \left\{(x_1, x_2) : x_1 = \frac{1}{2}x_2|x_2|\right\}. \tag{70}$$

The *unique* time-optimal control law, $\boldsymbol{u}^\star$, that solves this problem can be determined to be

$$\begin{aligned} \boldsymbol{u}^\star &= \boldsymbol{u}^\star(x_1, x_2) = +1 \quad \forall\, (x_1, x_2) \in \boldsymbol{\gamma}_+ \cup \mathbb{R}_+, \\ \boldsymbol{u}^\star &= \boldsymbol{u}^\star(x_1, x_2) = -1 \quad \forall\, (x_1, x_2) \in \boldsymbol{\gamma}_- \cup \mathbb{R}_-, \\ \boldsymbol{u}^\star &= -\text{sgn}\,\{x_2\} \quad \forall\, (x_1, x_2) \in \boldsymbol{\gamma}. \end{aligned} \tag{71}$$

The minimum cost for this problem is equivalent to the minimum time for states (x_1, x_2) to reach the origin $(0, 0)$. This is given as

$$\Phi^\star(x, t) \triangleq t^\star(x_1, x_2) \tag{72}$$

with the associated terminal value

$$-\frac{\partial \Phi^\star(x,t)}{\partial t} = H\left(t, x, \frac{\partial \Phi^\star(x,t)}{\partial t}, u\right)\Big|_{\substack{x=x^\star \\ u=u^\star}} \text{ with } H(t; x, u, p_1, p_2) = x_2(t)p_1(t) + u(t)p_2(t) \tag{73}$$

and

$$p_1 = \frac{\partial t^\star}{\partial x_1},\ \ p_2 = \frac{\partial t^\star}{\partial x_2}. \tag{74}$$

The HJ equation is given by

$$\begin{aligned} &\frac{\partial \Phi^\star}{\partial t} + x_2\frac{\partial \Phi^\star}{\partial x_1} - \frac{\partial \Phi^\star}{\partial x_2} = 0 && \text{if } x_1 > -\frac{1}{2}x_2|x_2|, \\ &\frac{\partial \Phi^\star}{\partial t} + x_2\frac{\partial \Phi^\star}{\partial x_1} + \frac{\partial \Phi^\star}{\partial x_2} = 0 && \text{if } x_1 < -\frac{1}{2}x_2|x_2|, \\ &\frac{\partial \Phi^\star}{\partial t} + x_2\frac{\partial \Phi^\star}{\partial x_1} - \text{sgn}\{x_2\}\frac{\partial \Phi^\star}{\partial x_2} = 0 && \text{if } x_1 = -\frac{1}{2}x_2|x_2|. \end{aligned} \tag{75}$$

The set of states (x_1, x_2) that can be forced to reach the origin in the same minimum time $t^\star \equiv \Phi^\star$ are the system's *isochrones* which are illustrated in Fig. 6. A point (x_1, x_2) on the state grid belongs

to the set of states $S(t^\star)$ from which it can be forced to the origin $(0, 0)$ in the same minimum time $t^\star$. We call the set $S(t^\star)$ the minimum isochrone. These are the isochrones of the system – akin to the isochrone map used in geography, hydrology, and transportation planning for depicting areas of equal travel time to a goal state. The level sets of (75) correspond to the *isochrones* of the system as illustrated in Fig. 6.

7.2.4 Approximate Time to Reach the Origin. We compare the analytical solution to the *time to reach (TTR) the origin* problem (see Fig. 6) against the approximated TTR solution using a dynamic implicit surface representation of the approximate value function. An ellipsoid with a radius of 1.0 along its major axis was chosen to represent the initial time to reach interface (see Fig. 7a, right inset). We then choose a controller with values ±1 depending on which side of the switch surface Fig. 6 we are on in generating the system's phase portrait illustrated in Fig. 4.

The closed-form solution to the time-to-reach the origin problem on a 2-D grid with x/y axis limits $[[-1, 1], [-1, 1]]$ is shown in the left inset of Fig. 4a. We set out to investigate the result of adding dynamics (with levelsets) to the elliptic implicit representation of the analytical TTR and evaluate the efficacy of our computational scheme. We proceed as shown in Listing 6.

```
finite_diff_data = {"innerFunc": termLaxFriedrichs,
 "innerData": {"grid": g, "hamFunc": dint.ham,
 "partialFunc": dint.dissipation,
 "dissFunc": artificialDissipationGLF,
 "CoStateCalc": upwindFirstENO2},
 "positive": False} // direction of approx. growth
```

Listing 5. Overapproximation setup for the double integrator TTR problem.

As a custom, a separate class (see `DoubleIntegrator` in the folder `DynamicalSystems`) holds all the dynamics (cref. equations 62 and 63), switching surface (cref. equations 68, 69, and 70), Hamiltonian (cref. equation 64), dissipation, and costates (cref. 74) of the double integrator plant. Over a twenty-step timespan ranging from 0 to 20, we integrate the right-hand-side of (75) forward in time by the Courant-Friedrichs-Lewy constrained second-order accurate integrator i.e. `odeCFL2` in our library:

```
t, y, ~ = odeCFL2(termRestrictUpdate, t_span, y0, options, finite_diff_data)
```

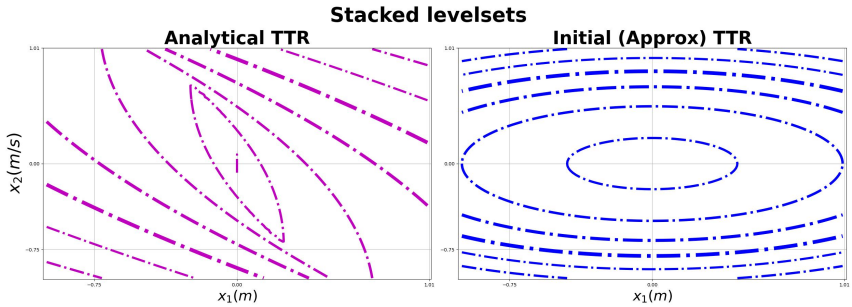
Listing 6. Overapproximation setup for the double integrator TTR problem.

where `y0` is the initial elliptic function that represents Φ in (75), `options` are the set of integration parameters such as the number of actual timesteps to take in the adaptive integration scheme, the maximum step size and so on. The routine `termRestrictUpdate` restricts the sign of the update of the HJ approximation by either increasing or decreasing the levelset.

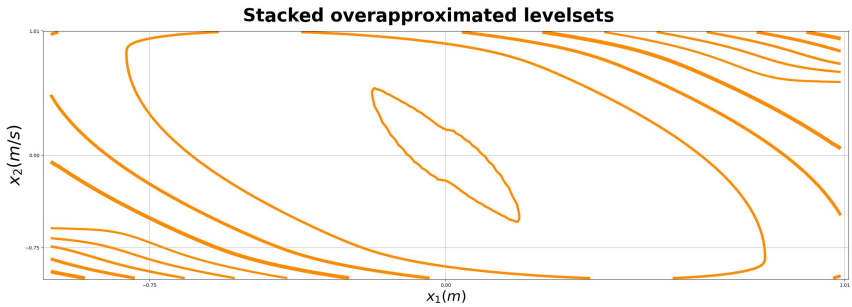
A side-by-side comparison of the level sets is shown in Fig. 7a. The approximation to the isochrones by our integration scheme is an overapproximation of the analytical TTR problem. This is illustrated in the right inset of Fig. 7a. Because we are not concerned with the safe set (unlike the example in 7.1), we do not overapproximate the time-to-reach solution. On the overall, we obtain similar isochrones to the analytical result, hence validating our hypothesis.

7.3 Reach-Avoid Games: Flocks within Starling Murmurations

Here, we will borrow inspiration from natural swarms, particularly the murmuration [21] of European starlings – the *sturnus vulgaris* – in our problem construction and solution concept. We are concerned with reach-avoid games in multiagent systems, whereupon agents must safely navigate a phase space (e.g. in achieving an attitude convergence goal), whilst avoiding collision with one another and capture by an external predator. A natural environment where this problem occurs is in the murmuration of European Starlings. The problem that we study is of importance in multiagent



(a) Analytical and Initial BRS (with implicit ellipsoids).



(b) Overapproximated BRS at time $t = 0.25$ secs

Fig. 7. Time to reach the origin at different integration time steps. Top-left: Closed-form Solution to the time to reach the origin problem. Top-right: Implicit representation of the initial TTR solution. Bottom: Lax-Friedrichs Approximation to the TTR the origin problem.

systems such as the safe control of quadcopters, safe interaction among distributed agents on a computing network where local nearest neighbor rules apply. In what follows, we formulate the problem mathematically and pose the collision avoidance for a local set of birds as a reachable differential game.

7.3.1 Problem Description. Consider a group of starlings moving on a space-time continuum $\Omega \times T$[10] as illustrated in Fig. 8. Recent field studies [8] suggest that emergent collective motion observed among these birds is as a result of local nearest neighbor interactions among separable subsets of bird groups on $\Omega \times T$. There is evidence with justifiable confidence [27] suggesting that when density varies among the birds (henceforth called agents), the relationship among agents in local groups is not determined by the metric distance among nearest neighbors but rather by a topological notion of distance (defined as the number of intermediate birds between one agent from another [2]).

Starlings exhibit complex formation patterns that are effective in avoiding capture – mostly by peregrine Falcons in the wild [20]. We will leverage this notion of topological distance between agents in developing a target or safe set [24] for a subset of agents within the murmuration. If we can compute this safe set, it can serve as e.g. a safety filter for multiagent systems (to be controlled) and their actual controller whilst respecting state constraints [24] and the control constraints (admissible inputs, dynamic update frequency of input control laws e.t.c.). An illustration of the problem setup is illustrated in Fig. 8. For a comprehensive understanding of the intuition that guides our

[10]Here Ω is the open set that contains all states of the birds and T is the length of time over the real line.

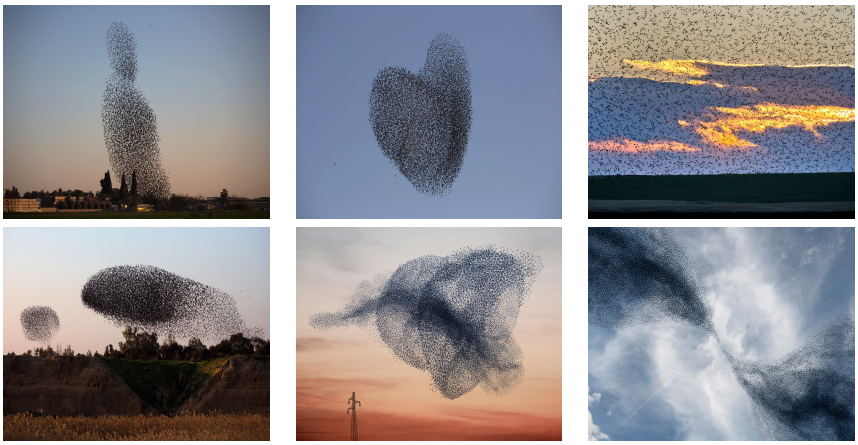


Fig. 8. Starlings murmurations. From the top-left and clockwise. (i) A starlings flock rises into the air, in a dense structure (Reuters/Amir Cohen). (ii) Starlings migrating over an Israeli village (AP Photo/Oded Balilty). (iii) Starlings feeding on laid seeds in the ground in Romania. (iv) Two flocks of migrating starlings (Menahem Kahana/AFP/Getty Images). (v) A concentric conical formation of starlings (Courtesy of The Gathering Site.). (vi) Splitting and joining of a flock of starlings.

mathematical formulation here, we refer agents to the works of Ballerini et al. [2], Cavagna et al. [8], and Cardaliaguet [7].

7.3.2 Mathematical abstractions. Individual agents self-organize into phases or regions $\mathcal{S}$ which are in turn members of a union of multiple regions C. Every $\mathcal{S} \subseteq C$ and all members of C are disjoint from one another i.e. $\mathcal{S}_i \cup \mathcal{S}_j = \emptyset$ for any $i \neq j$. The total number of elements in $\mathcal{S}$ is denoted $[\mathcal{S}]$, and we denote by $\text{int}\,\Omega$ the interior of Ω. The closure of Ω is $\bar{\Omega}$. We let $\delta\Omega\,(:= \bar{\Omega}\backslash\text{int}\,\Omega)$ be the boundary of Ω.

An evading agent in a region $\mathcal{S}_i$ has a state notation $\boldsymbol{x}_a^i$ (read: the state of agent a in region i). A state $\boldsymbol{x}_a^i$ has linear velocity components, $\boldsymbol{x}_{a_1}^i, \boldsymbol{x}_{a_2}^i$, and heading $\boldsymbol{x}_{a_3}^i := w_a^i$. When we must distinguish an agent $\boldsymbol{x}_a^i \in C_x$ from some other agent e.g. in another multiphase C_y, we shall write ${}^x\boldsymbol{x}_a^i$ and ${}^y\boldsymbol{x}_a^i$ respectively.

The set of players in a game shall be denoted by $\mathcal{N} = \{i, j, \ldots\}$ with the subscript index indicating players e.g. $\mathcal{N}_i$ for player i. The set of neighbors of player i is $\mathcal{N}(i) \subseteq \mathcal{N}$. Player i moves dynamically with a control $u_i \in \pi_i$ (i.e. u_i belongs to a policy class π_i) that is both (a) optimal with respect to its own objective $\mathcal{J}_i$; and (b) optimal with respect to its neighboring players' current policy $\pi_{-i} \in \Pi_{j\in N, j\neq i}$. Neighbors of agent i at time t are those which are either within, or on a circle specified by a fixed topological range, r_c. This topological range is given by the difference in the numerical label of individuals (see Definition 3), and is consistent with findings in collective swarm behaviors as it reinforces *group cohesion* [2].

The topological metric is given by the label of an agent and it quantifies the number of intermediate agents that separate two agents. This is consistent with collective animal behaviors where individuals' bookkeeping on their neighbors' positions help maintain the strength of an interaction when density varies or when they need to reorient a control input, given by the average of its own orientation and

that of its neighbors. Instead of metric distance interaction rules that make agents very vulnerable to predation [2], we resort to a topological interaction rule[11]. Let us set forth with a few definitions first.

DEFINITION 1. *Every agent within a flock has similar dynamics to that of its neighbor(s). Furthermore, agents travel at the same linear speed, v; the angular headings, w, however, may be different between agents, seeing we are dealing with a many-bodied system. Each agent's continuous-time dynamics, $\dot{\boldsymbol{x}}^{(i)}(t)$, evolves as*

$$\begin{bmatrix} \dot{\boldsymbol{x}}_1^{(i)}(t) \\ \dot{\boldsymbol{x}}_2^{(i)}(t) \\ \dot{\boldsymbol{x}}_3^{(i)}(t) \end{bmatrix} = \begin{bmatrix} v(t)\cos \boldsymbol{x}_3^{(i)}(t) \\ v(t)\sin \boldsymbol{x}_3^{(i)}(t) \\ \langle w^{(i)}(t)\rangle_r \end{bmatrix}, \langle w^{(i)}(t)\rangle_r = \frac{1}{1+n_i(t)}\left(w^{(i)}(t) + \sum_{j\in \mathcal{N}_i(t)} w_j(t) \right) \tag{76}$$

for agents $i = \{1, 2, 3, ..., n_a\}$, where t is the continuous-time index, $n_i(t)$ is the number of agent i's neighbors at time t, $\mathcal{N}_i(t)$ denotes the sets of labels of agent i's neighbors at time t, and $\langle w^{(i)}(t)\rangle_r$ is the average orientation of agent i and its neighbors at time t. Note that for a game where all agents share the same constant linear speed and heading, (76) *reduces to the dynamics of a Dubins' vehicle in absolute coordinates with $-\pi \leq w^{(i)}(t) < \pi$. The averaging over the degrees of freedom of other agents in* (76) *is consistent with the mean field theory, whereby the effect of all other agents on any one agent is an approximation of a single averaged influence.*

DEFINITION 2 (NEIGHBORS OF AN AGENT). *We define the neighbors $\mathcal{N}_i(t)$ of agent i at time t as the set of all agents that lie within a predefined radius, r_i.*

DEFINITION 3. *We define a flock, F, consisting of agents labeled $\{1, 2, \cdots, n_a\}$ as a collection of agents within a phase space $(\Omega \times T)$ such that all agents within the flock interact with their nearest neighbors in a topological sense.*

REMARK 1. *Note that for a game where all agents share the same constant linear speed and heading,* (76) *reduces to the dynamics of a Dubins' vehicle in absolute coordinates with $-\pi \leq w^{(i)}(t) < \pi$. The averaging over the degrees of freedom of other agents in* (76) *is consistent with mean field theory, whereby the effect of all other agents on any one agent is an approximation of a single averaged influence.*

DEFINITION 4 (PAYOFF OF A FLOCK). *To every flock F_j (with a finite number of agents n_a) within a murmuration, $j = \{1, 2, \cdots, n_f\}$, we associate a payoff, Φ_j, that is the union of all respective agent's payoffs for expressing the outcome of a desired kinematic behavior.*

7.3.3 Flock Motion as Differential Games. We restrict our analysis to a single local flock within a murmuration. We must find a mathematical way to replicate the collision-avoidance scheme that agents execute structural homogeneity of movement in every region $\mathcal{S}_i \in C$ for $i = 1, \cdots, [C]$ as observed in natural systems. We will locally synthesize the kinematics of agents in a manner amenable to state representation by resolving local payoff extremals, $\{\boldsymbol{\phi}_1, \cdots, \boldsymbol{\phi}_{n_f}\}$. This is a state space partition induced by an aggregation of desired collective behavior from local flocks' values $\{\boldsymbol{v}_1, \cdots, \boldsymbol{v}_{n_f}\}$. Let the cursory reader understand that we use the concept of a flock loosely. The value function could represent a palette of composed value functions whose extremals resolve local behaviors we would like to emerge over separated local regions of the state space of dexterous drone acrobatics [29], a robot balls juggling task [6], or any parallel task domain verification problem.

[11]With metric distance rules, we will have to formulate the breaking apart of value functions that encode a consensus heading problem in order to resolve the extrema of multiple payoffs; which is typically what we want to mitigate against during real-world autonomous tasks.

7.3.4 Framework for Separated Payoffs. Suppose that a murmuration's global heading is given and that each agent i within each flock, F_j, $(j = \{1, \cdots, n_f\})$ in the murmuration has a constant linear velocity, v^i. An agent's orientation is its control input, given by the average of its own orientation and that of its neighbors. What constitutes an agent's neighbors are computed based on empirical findings and studies from the lateral vision of birds and fishes [2, 23, 27] that provide insights into their anisotropic kinematic density and structure. Importantly, starlings' lateral visual axes and their lack of a rear sector reinforces their lack of nearest neighbors in the front-rear direction. As such, this enables them to maintain a tight density and robust heading during formation and flight.

Each agent within flock F_j interacts with a fixed number of neighbors, n_c, within a fixed topological range, r_c. The range is chosen as the difference between the numerical labels of agents in a flock. This is consistent with findings in collective swarm behaviors in that it reinforces *group cohesion* [2]. Since in starlings behavior, flying performance is often spurred by a predator, we emulate this by introducing an external disturbance on the zeroth-index agent within the flock of interest (this aids compactness of the zero levelset of flock $\mathcal{S}_i$ as the theory of HJ Reachability recommends [37]). However, we are interested in *robust group cohesion*. Ergo, we let a pursuer, $\boldsymbol{P}$, with a worst-possible disturbance attack the flock. Here, flocks constitute an evading player, $\boldsymbol{E}$.

The delineation of an agent's nearest neighbors is given in Algorithm 1. On lines 3 and 7 of Algorithm 1, cohesion is reinforced by leveraging the observations above. While the neighbor updates for an agent involve an $O(n^2)$ algorithm in Algorithm 1, we are merely dealing with $6-7$ agents at a time in a local flock – making the computational cost measly.

7.3.5 Global Isotropy via Local Anisotropy. Isotropy of motion fields is a natural characteristic in Starlings motion. The global isotropy of murmurations where group cohesion is maintained in highly uncertain environments with limited or noisy information is often stimulated by Peregrine Falcon attacks. Local birds maintain structural anisotroby via nearest neighbor rules, and a collection of multiple local groups in the entire collection results in the global isotroby that is observed. However, structural anisotropy is not merely an effect of a preferential velocity in animal flocking kinematics but rather an explicit effect of the anisotropic interaction character itself: agents choose a mutual position on the state space in order to maximize the sensitivity to changes in heading and speed of neighbors; the neighbors' anisotropy is optimized via vision-based collision avoidance characteristically unrelated to the eye's structure [2].

To reinforce robust group cohesion in local flocks, we let a pursuer $\boldsymbol{P}_j$ play attack an evading agent $\boldsymbol{E}_j$ in a flock F_j so that one agent within F_j is always in relative coordinates with $\boldsymbol{P}^j$. By averaging the heading of individual agents' orientations with its neighbors (cf. (76)), a flock can achieve fast response to danger when a pursuer is nearby. In this specialized case, $\boldsymbol{E}$ and $\boldsymbol{P}$'s speeds and maximum turn radii are equal: if both players start the game with the same initial velocity and orientation, the relative equations of motion show that $\boldsymbol{E}$ can mimic $\boldsymbol{P}$'s strategy by forever keeping the starting radial separation. As such, the *barrier* is closed and *the central theme in this game of kind is to determine the surface of the boundary* [33]. We defer a thorough analysis of the nature of the surface to a future work. Owing to the high-dimensionality of the state space, we cannot resolve this barrier analytically, hence we resort to our HJ PDE numerical approximation.

For agent i within a flock with index j in a murmuration, the equations of motion under attack from a predator p in relative coordinates is

$$\begin{bmatrix} \dot{\boldsymbol{x}}_1^{(i)_j}(t) \\ \dot{\boldsymbol{x}}_2^{(i)_j}(t) \\ \dot{\boldsymbol{x}}_3^{(i)_j}(t) \end{bmatrix} = \begin{bmatrix} -v_e^{(i)_j}(t) + v_p^{(j)} \cos \boldsymbol{x}_3^{(i)_j}(t) + \langle w_e^{(i)_j} \rangle_r \boldsymbol{x}_2^{(i)_j}(t) \\ v_p^{(i)_j}(t) \sin \boldsymbol{x}_3^{(i)_j}(t) - \langle w_e^{(i)_j} \rangle_r \boldsymbol{x}_1^{(i)_j}(t) \\ w_p^{(j)}(t) - \langle w_e^{(i)_j}(t) \rangle_r \end{bmatrix} \text{ for } i = 1, \cdots, n_a \tag{77}$$

Algorithm 1 Nearest Neighbors For Agents in a Flock.

```
Given a set of agents a = {a_1, a_2, ··· , a_{n_a} | [a] = n_a}        ▷ n_a agents in a flock F_k.
function UpdateNeighbor(n)
    for i in 1, ..., n do                                  ▷ Look to the right and update neighbors.
        for j in i + 1, ..., n do
            COMPARE_NEIGHBOR(a[i], a[j])
        end for
        for j in i − 1 down to 0 do                        ▷ Look to the left and update neighbors.
            COMPARE_NEIGHBOR(a[i], a[j])
        end for
    end for
    for each a_i ∈ F_k, i = 1, ··· n_a do                  ▷ Recursively update agents' headings.
        Update headings according to (76).
    end for
end function
```

```
function Compare_Neighbor(a_1, a_2)                        ▷ (a_1, a_2): distinct instances of AGENT.
    if |a_1.label - a_2.label| < a_1.r_c^1                 ▷ r_c^n: agent n's capture radius, r_c.
        a_1.UPDATE_NEIGHBORS(a_2) then
    end if
end function
```

```
procedure Agent(a_i, Neighbors={})                         ▷ Neighbors: Set of neighbors of this agent.
                                  ▷ Agent a_i with attributes label ∈ ℕ, avoid and capture radii, r_a, r_c.
    function UPDATE_NEIGHBORS(neigh)
        if length(neigh)> 1 then                           ▷ Multiple neighbors.
            for each neighbor of neigh do
                UPDATE_NEIGHBORS(neighbor)                 ▷ Recursive updates.
            end for
        end if
        Add neigh to Neighbors
    end function
end procedure
```

where n_a is the number of agents within a flock, $\left(\boldsymbol{x}_1^{(i)_j}(t), \boldsymbol{x}_2^{(i)_j}(t)\right) \in \mathbb{R}^2$, and we have $\boldsymbol{x}_3^{(i)_j}(t) \in [-\pi, +\pi)$. We have multiplied the dynamics by -1 so that the extremal's resolution evolves backwards in time. Read $\boldsymbol{x}_1^{(i)_j}(t)$: the first component of the state of agent i for flock j at time t. In absolute coordinates, the equation of motion for *free agents* is

$$\begin{bmatrix} \dot{\boldsymbol{x}}_1^{(i)_j}(t) \\ \dot{\boldsymbol{x}}_2^{(i)_j}(t) \\ \dot{\boldsymbol{x}}_3^{(i)_j}(t) \end{bmatrix} = \begin{bmatrix} v_e^{(i)_j}(t) \cos \boldsymbol{x}_3^{(i)_j}(t) \\ v_e^{(i)_j}(t) \sin \boldsymbol{x}_3^{(i)_j}(t) \\ \langle w_e^{(i)_j}(t) \rangle_r \end{bmatrix}. \tag{78}$$

7.3.6 Flock Motion from Aggregated Value Functions. We introduce the union operator i.e. $\cup$ below as an aggregation symbol since the respective payoffs of each agent in a flock may be implicitly or explicitly constructed. In resolving the zero-level sets of HJ value functions, it is typical to represent the payoff's surface as the isocontour of some function (usually a signed distance

function). In these instances, we shall aggregate the payoff of agents 1 and 2, for example, as

$$\cup\{\boldsymbol{\phi}_1(\boldsymbol{x},t),\boldsymbol{\phi}_2(\boldsymbol{x},t)\} \equiv \boldsymbol{\phi}_1(\boldsymbol{x},t) \cup \boldsymbol{\phi}_2(\boldsymbol{x},t) = \min(\boldsymbol{\phi}_1(\boldsymbol{x},t),\boldsymbol{\phi}_2(\boldsymbol{x},t)). \tag{79}$$

Standard assumptions about the existence of a flock's *value* applies. And by an extension of Hamilton's principle of least action, the terminal motion of a flock coincide with the extremal of the payoff functional i.e. ,

$$\boldsymbol{v}(\boldsymbol{x},t) = \inf_{\beta^{(1)}\in\mathcal{B}^{(1)}} \sup_{\boldsymbol{u}^{(1)}\in\mathcal{U}^{(1)}} \cup\left[g^{(1)}(\boldsymbol{x}(T))\right] \cup\cdots\cup \inf_{\beta^{(n_f)}\in\mathcal{B}^{(n_f)}} \sup_{\boldsymbol{u}^{(n_f)}\in\mathcal{U}^{(n_f)}} \left[g^{(n_f)}(\boldsymbol{x}(T))\right]$$

where n_f is the total number of distinct flocks in a murmuration. The resolution of this equation admits a viscosity solution to the following variational terminal HJI PDE [37]

$$\cup_{j=1}^{n_f}\left[\cup_{i=1}^{n_a}\left(\frac{\partial \boldsymbol{v}_i}{\partial t}(\boldsymbol{x},t) + \min\left[0,\boldsymbol{H}^{(i)}(\boldsymbol{x}^{(i)},\boldsymbol{v}_x(\boldsymbol{x},t))\right]\right)\right] = 0. \tag{80}$$

with Hamiltonian,

$$\boldsymbol{H}^{(i)}(t;\boldsymbol{x}^{(i)},\boldsymbol{u}^{(i)},\boldsymbol{v}^{(i)},p^{(i)}) = \max_{u^{(i)}\in\mathcal{U}^{(i)}} \min_{v^{(i)}\in\mathcal{V}^{(i)}} \langle f^{(i)}(t;\boldsymbol{x},\boldsymbol{u}^{(i)},\boldsymbol{v}^{(i)}),p^{(i)}\rangle. \tag{81}$$

In swarms' collective motion, when e.g. a Peregrine Falcon attacks, immediate nearest agents change direction almost instantaneously. And because of the interdependence of the orientations of individual agents with respect to one another, all other agents respond instantaneously. Thus, we only simulate a single attack against a flock within the murmuration to realize robust cohesion. Throughout the game, we assume that the roles of $\boldsymbol{P}$ and $\boldsymbol{E}$ do not change, so that when capture can occur, a necessary condition to be satisfied by the saddle-point controls of the players is the Hamiltonian, $\boldsymbol{H}^i(\boldsymbol{x},p)$.

THEOREM 1. *For a flock, F_j, the Hamiltonian is the total energy given by a summation of the exerted energy by each agent i so that we can write the main equation or total Hamiltonian of a murmuration as*

$$\boldsymbol{H}(\boldsymbol{x},p) = \max_{w_e^{(k)_j}\in[\underline{w}_e^j,\bar{w}_e^j]} \min_{w_p^{(k)_j}\in[\underline{w}_p^j,\bar{w}_p^j]} \cup_{j=1}^{n_f}\left[H_a^{(k)_j}(\boldsymbol{x},p) \cup \left(\cup_{i=1}^{n_a-1} H_f^{(i)_j}(\boldsymbol{x},p)\right)\right] \tag{82}$$

$$\begin{aligned} &\triangleq \cup_{j=1}^{n_f}\left(\cup_{i=1}^{n_a-1}\left[p_1^{(i)_j}v^{(i)_j}\cos\boldsymbol{x}_3 + p_2^{(i)_j}v^{(i)_j}\sin\boldsymbol{x}_3 + p_3^{(i)_j}\langle w_e^{(i)_j}\rangle_r\right]\right. \\ &\quad\cup\left[p_1^{(k)_j}\left(v^{(k)_j} - v^{(k)_j}\cos\boldsymbol{x}_3^{(k)_j}\right) - p_2^{(k)_j}v^{(k)_j}\sin\boldsymbol{x}_3^{(k)_j} - \underline{w}_p^j|p_3^{(k)_j}|\right. \\ &\qquad\left.\left.+\bar{w}_e^j\left|p_2^{(k)_j}\boldsymbol{x}_1^{(k)_j} - p_1^{(k)_j}\boldsymbol{x}_2^{(k)_j} + p_3^{(k)_j}\right|\right]\right). \end{aligned} \tag{83}$$

where $\boldsymbol{H}_a^{(k)_j}(\boldsymbol{x},p)$ is the Hamiltonian of the individual under attack by a pursuing agent, $\boldsymbol{P}$, and $H_f^{(i)_j}(\boldsymbol{x},p)$ are the respective Hamiltonians of the free agents, $i=\{1,\cdots,n_f\}$, within an evading flock, and not under the direct influence of capture or attack by $\boldsymbol{P}$. We denote by $w_e^{(i)_j}$ the heading of an evader i within a flock j and $w_p^{(j)}$ the heading of a pursuer aimed at flock j; $\underline{w}_e^{(k)_j}$ is the orientation that corresponds to the orientation of the agent with minimum turn radius among all the neighbors of agent k, inclusive of agent k at time t; similarly, $\bar{w}_e^{(k)_j}$ is the maximum orientation among all of the orientation of agent k's neighbors.

PROOF. The proof to this theorem is given in Appendix A. □

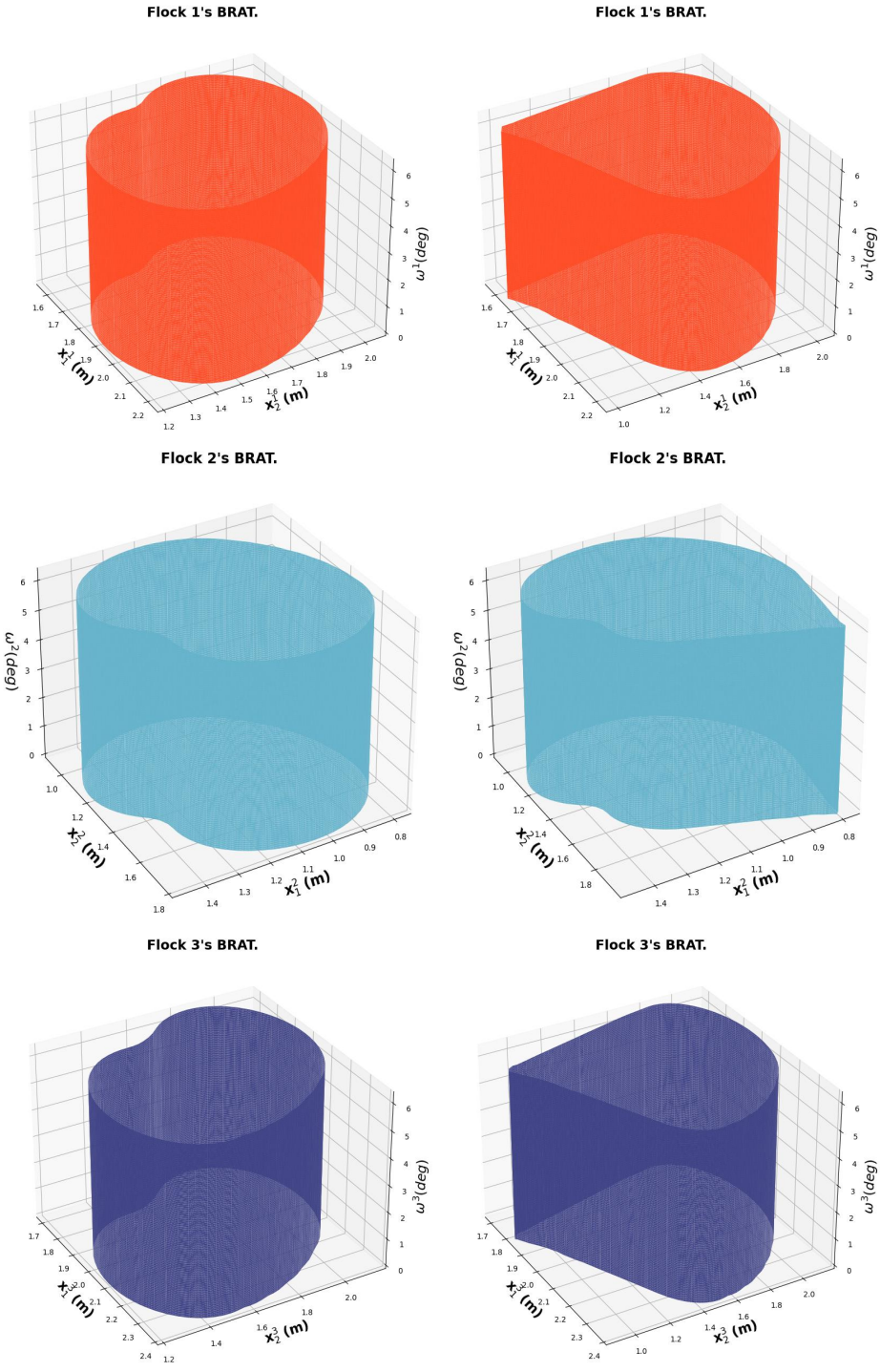


Fig. 9. Left column: Initial zero-level set for various flocks at different initial conditions. Right column: Evading flock's interface under a pursuer's attack after specific Lax-Friedrichs' integration. (Metric reach radius=$0.2m$, Avoid Radius=$0.2m$).

COROLLARY 2. *For the special case where the linear speeds of the evading agents and pursuer are equal i.e.* $v_e^{(i)_j}(t) = v_p(t) = +1m/s$, *we have the Hamiltonian as*

$$\begin{aligned}\boldsymbol{H}(\boldsymbol{x},p) = \cup_{j=1}^{n_f}\Big(\cup_{i=1}^{n_a-1}\Big[p_1^{(i)_j}\cos\boldsymbol{x}_3 + p_2^{(i)_j}\sin\boldsymbol{x}_3 + p_3^{(i)_j}\langle w_e^{(i)_j}\rangle_r\Big] \\ \cup\Big[p_1^{(k)_j}\Big(1-\cos\boldsymbol{x}_3^{(k)_j}\Big) - p_2^{(k)_j}\sin\boldsymbol{x}_3^{(k)_j} - \underline{w}_p^j|p_3^{(k)_j}| \\ +\bar{w}_e^j\Big|p_2^{(k)_j}\boldsymbol{x}_1^{(k)_j} - p_1^{(k)_j}\boldsymbol{x}_2^{(k)_j} + p_3^{(k)_j}\Big|\Big]\Big).\end{aligned} \tag{84}$$

We adopt the essentially non-oscillatory Lax-Friedrichs scheme of [42] in resolving (84). Denote by (x,y,z) a generic point in $\mathbb{R}^3$ so that given mesh sizes Δx, Δy, Δz, $\Delta t > 0$, letters u,v,w will represent functions on the x,y,z lattice $\Delta = \{(x_i,y_j,z_k) : i,j,k \in \mathbb{Z}\}$. We define the numerical monotone flux, $\hat{\boldsymbol{H}}^{(i)_j}(\cdot)$, of $\boldsymbol{H}_j^{(i)}(\cdot)$ as

$$\begin{aligned}\hat{\boldsymbol{H}}^{(i)_j}(u^+,u^-,v^+,v^-,w^+,w^-) = \boldsymbol{H}^{(i)_j}\left(\frac{u^+ + u^-}{2},\frac{v^+ + v^-}{2},\frac{w^+ + w^-}{2}\right) \\ -\frac{1}{2}\left[\alpha_x^{(i)_j}\left(u^+ - u^-\right) + \alpha_y^{(i)_j}\left(v^+ - v^-\right) + \alpha_z^{(i)_j}\left(w^+ - w^-\right)\right]\end{aligned} \tag{85}$$

where

$$\alpha_x^{(i)_j} = \max_{\substack{a\le u\le b\\ c\le v\le d\\ e\le w\le f}}|\boldsymbol{H}_u^{(i)_j}(\cdot)|,\ \alpha_y^{(i)_j} = \max_{\substack{a\le u\le b\\ c\le v\le d\\ e\le w\le f}}|\boldsymbol{H}_v^{(i)_j}(\cdot)|,\ \alpha_z^{(i)_j} = \max_{\substack{a\le u\le b\\ c\le v\le d\\ e\le w\le f}}|\boldsymbol{H}_w^{(i)_j}(\cdot)| \tag{86}$$

are the dissipation coefficients, controlling the level of numerical viscosity in order to realize a stable solution that is physically realistic [12]. Here, the subscripts of $\boldsymbol{H}$ are the partial derivatives w.r.t the grid dimension variable, and the flux, $\hat{\boldsymbol{H}}(\cdot)$ is monotone for $a \le u^\pm \le b, c \le v^\pm \le d, e \le w^\pm \le f$. We adopt the total variation diminishing Runge-Kutta scheme of [41, 45] in efficiently calculating essentially non-oscillating upwinding finite difference gradients of $\boldsymbol{H}(\cdot)$.

The computed safe sets are as shown in Figure 9. Note that the symmetry between non-consecutive flock labels e.g. flock 1 and flock 3's RCBRAT is because the we multiplied the initial position of a flock's state by -1.

7.4 Dubins' Game of Two Identical Vehicles

This example was originally proposed by Merz [33] as an iteration upon Isaacs [26]'s homicidal chauffeur game, whereupon a pursuit-evasion game between two players with similar speeds and minimum turn radii, is thoroughly analyzed. In Mitchell [36], this problem was established as a benchmark for testing the solubility of capturable set of states (the backward reachable tube) in Merz's classical pursuit-evasion game. In this example, we solve the problem with our LevelSetPy toolbox and establish that the approximated barrier surface to the two-player game conforms with standard results.

The game is that of two cars sharing similar Dubins dynamics [16]: $\boldsymbol{P}$ and $\boldsymbol{E}$ both have a positive minimum turn radii, w, and constant speeds v – with motion restricted to a plane as we have for the rocket launch differential game above. In relative coordinates, the diagrammatic structure of the motion is as depicted in Fig. 10. Choosing the Cartesian coordinate for motion representation, the state vector of the game with $\boldsymbol{E}$ at the origin can be characterized by its x_1, x_2 position relative to $\boldsymbol{P}$ and the angle θ between the two vehicles. Capture occurs when the distance $\|\boldsymbol{PE}\|_2$ between the pursuer and the evader becomes less than a specified radius.

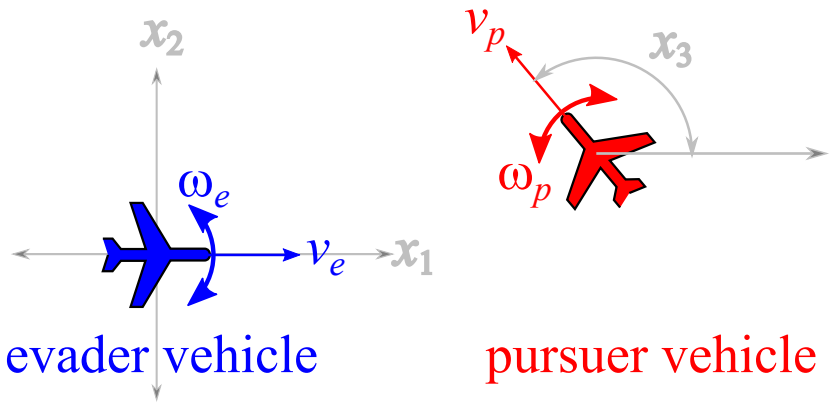


Fig. 10. Two Dubins' vehicles in relative Cartesian coordinates. Reprinted from Mitchell [36].

The relative equations of motion, going by Fig. 10, is

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{pmatrix} = \begin{pmatrix} -v_e + v_p \cos x_3 + w_e x_2 \\ v_p \sin x_3 - w_e x_1 \\ w_p - w_e \end{pmatrix}. \tag{87}$$

We adopt specialization to a case where the two vehicles only possess a unit velocity and unit maximum turn rates. Here, as Merz notes, if the initial velocities are parallel such as $x_3 = 0$, then the equations of relative motion imply that $\boldsymbol{E}$ can be separated from $\boldsymbol{P}$ forever by the initial radial separation if it replicates $\boldsymbol{P}$'s strategy. Whence, the barrier surface is closed and we are presented with Isaacs [26]'s game of kind where we must determine the nature of the surface. This terminal surface possesses a closed-form solution and we refer readers to the treatment by Merz [33]. In this example, our chief concern is to judge the efficacy of our toolbox with respect to the analytical solution of the barrier surface.

The the backward reachable tube that consists of the paths taken by the trajectories of either player is defined as in the rockets pursuit evasion game so that we have

$$\Phi(0, x) = \{x \in \mathcal{X} | x_1^2 + x_2^2 \leq r^2\}, \tag{88}$$

where again r is the capture radius. The target set is a cylinder as Φ above excludes the heading, x_3. It is represented as shown in Fig. 11.

For a detailed treatment of the barrier surface, we refer readers to a proper analysis as elucidated in [36]. Here, we focus on the construction of the BUP. The set of states that constitute the useable part and its boundary are respectively a function of the implicit surface function representation $\Phi : [-T, 0] \times \mathcal{X} \rightarrow \mathbb{R}$ so that for a $t \in [0, T]$, where $T > 0$ is

$$\mathcal{T} = \{x \in \mathcal{X} | \Phi(0, x) \leq 0\} \tag{89}$$

$$R([-t, 0], \mathcal{T}) = \{x \in \mathcal{X} | \Phi(t, x) \leq 0\}, \tag{90}$$

When $t > 0$, the implicit surface representation is the following HJI PDE

$$\frac{\partial}{\partial t}\Phi(t, x) + \min\left(0, \boldsymbol{H}(x, \nabla_x \Phi(t, x))\right) = 0. \tag{91}$$

It is easy to verify that the Hamiltonian is

$$H(x, p) = p_1(v_e - v_p \cos x_3) - p_2(v_p \sin x_3) - w|p_1 x_2 - p_2 x_1 - p3| + w|p_3|. \tag{92}$$

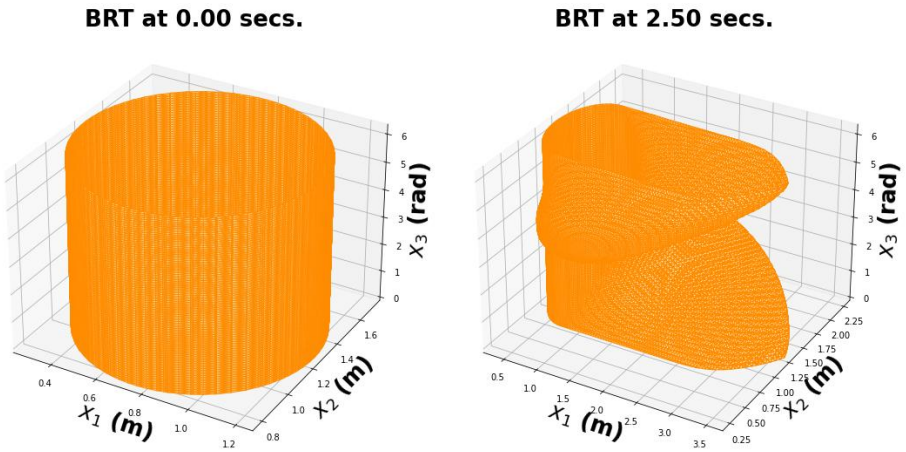


Fig. 11. The target set (left) and the boundary of the useable part of the state space after the differential game between P and E.

Since we are concerned with the special case that the linear and angular speeds are equal, we set $v_e = v_p = w \triangleq +1$ in the foregoing so that the Hamiltonian, in the final analysis is

$$H(x,p) = p_1(1-\cos x_3) - p_2(\sin x_3) - |p_1 x_2 - p_2 x_1 - p3| + |p_3|. \tag{93}$$

As before, we set up the differential game as in Listing 7

```
finite_diff_data = {"innerFunc": termLaxFriedrichs,
 "innerData": {"grid": g, "hamFunc": dubins_rel.ham,
 "partialFunc": dubins_rel.dissipation,
 "dissFunc": artificialDissipationGLF,
 "CoStateCalc": upwindFirstENO2},
 "positive": True} // direction of approx. growth
```

Listing 7. HJ ENO2 computational scheme for the rockets.

The BRTs at various time steps for the approximation of the differential game is shown in Fig. 12. Compared to the standardized benchmark of the analytical solution [33] to the differential game problem and the approximated solutions [36, 37], our results jibe.

7.5 Computational Time Comparison with LevelSet Toolbox

In this subsection, we will compare the solution for recovering the zero level set of the systems presented in the previous examples against Mitchell [35]'s `LevelSet Toolbox` in `MATLAB®`. In all, we compare the efficacy of running various computational problems using our library on a `CPU` – running with `Numpy` and its fast arithmetic libraries – versus on a CPU with `MATLAB®`– as originally written in Mitchell [35]'s library. In addition, we compare the efficacy of running these computational problems on a single `GPU`.

For the `CPU` tests, we run the computation on an Intel Core™i9-10885H 16 cores-processor with a 2.4GHz clock frequency, and 62.4GB memory. We employed an `NVIDIA Quadro RTX` 4000 with 8.192 GiB memory running on a mobile workstation with the `CPU` specifications mentioned erstwhile in all of our `GPU` library accelerations.

Table 1 depicts the time it takes to process a full global optimization and the average time for the Lax-Friedrichs internal computational optimization algorithm for the reachable sets/tubes and time-to-reach sets for the examples we have considered. The column `Avg. local` depicts the

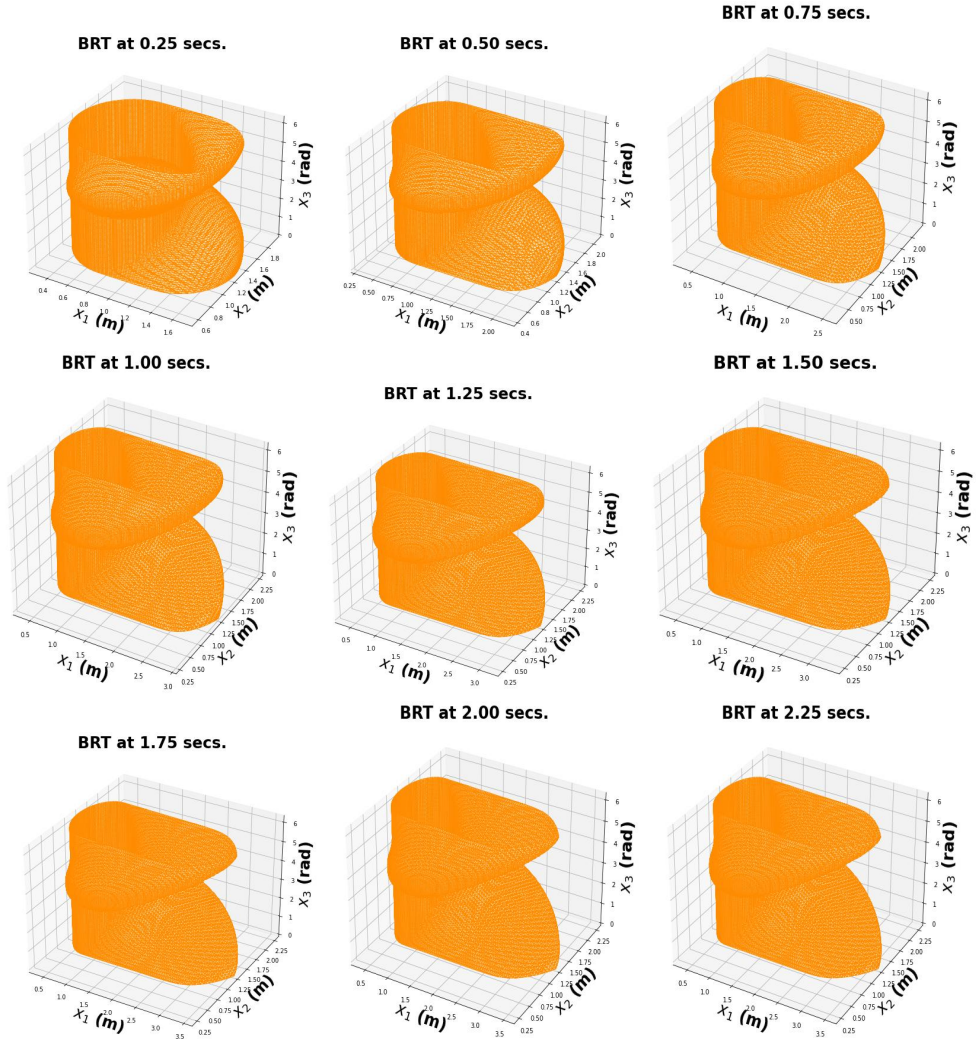


Fig. 12. BRTs for the differential game until termination time (2.25 secs).

average time the employed method of lines for resolving the HJ PDE takes per optimization step. The column `Global` denotes the average time it takes to compute the numerical solution to the HJ PDE . Each time query field represents an average over 20 experiments. We compare results of running the algorithm on a `GPU`, and `CPU` (both in `Numpy` and `MATLAB`). In all our evaluations, we aggressively free up GPU memory between and during computations in order to make GPU data streaming and memory computations more efficient.

Across the `GPU` experiments for the examples presented, we see that computation is significantly faster across all categories save the low-dimensional double integral plant experiment. We attribute this to the little amount of data points used in the overapproximated stacked levelsets. For the Air3D game of two vehicles on a plane problem and the two rockets differential game problem, the average local time for computing the solutions to the stagewise HJ PDE 's using the method of lines for Air3D is a gain of $\sim$ 76%; the global time is a gain of 76.09%. Similarly, we notice substantial computational

Table 1. Time to Resolve HJ PDE's.

Expt \ Library	LevelSetPy GPU Time (secs)		LevelSetPy CPU Time (secs)	
	Global	Avg. local	Global	Avg. local
Rockets Game	11.5153 ± 0.038	1.1833	107.84 ± 0.42	10.4023
Double Integrator	14.7657 ± 0.2643	1.5441	3.4535 ± 0.34	0.4317
Air 3D	30.8654 ± 0.1351	3.0881	129.1165 ± 0.13	12.6373
Starlings flock	8.6889 ± 0.8323	0.42853	15.2693 ± 0.167	7.4387

Expt \ Library	MATLAB CPU Time (secs)	
	Global	Avg. local
Rockets Game	138.50	13.850
Double Integrator	5.23	0.65375
Dubins Game	134.77	16.8462

gains for the two rockets differential game problem: 89% faster global optimization time and 88.62% average local computational time compared to our CPU implementations in `Numpy`. For this rockets game problem, compared against Mitchell [35] library, we notice a speedup of almost 92% in global optimization using our `GPU`-calibrated library versus an 89.32% gain using our `CPU`-calibrated library.

Notice the exception with the `Double Integrator` experiment, however: local and global computations take a little longer compared to deployments on the `CPU` – both on our Harris et al. [22]'s implementation and using Mitchell [35]'s native `MATLAB`®toolbox. We attribute this to the little size of the arrays of interest in this problem. The entire target set of the double integral plant exists on a two-dimensional grid whose analytic and approximate time-to-reach-the-origin computational time involves little computational gain in passing data onto the GPU. Nevertheless, we still see noticeable gains in using our CPU implementation as opposed to Mitchell [35]'s native `MATLAB`®toolbox.

On a CPU, owing to efficient arrays arithmetic native to Harris et al. [22]'s Numpy library, the average time to compute the zero levelsets per optimization step for the `odeCFLx` functions is faster with our Numpy implementation compared against Mitchell [35] LevelSets `MATLAB`® Toolbox library computations across all experiments. The inefficiencies of `MATLAB`®'s array processing routine in the longer time to resolve stagewise BRTs and the effective time to finish the overall HJ PDE resolution per experiment manifests in all of our experimental categories. For `CPU` processing of HJ PDE 's, it is reasonable, based on these presented data to expect that users would find our library far more useful for everyday computations in matters relating to the numerical resolution of HJ PDE 's.

In all, there is conclusive evidence that our implementations are faster, extensible to modern libraries, and scalable for modern complex system design and verification problems that arise.

8 CONCLUSION

With the advent of function approximation (neural networks) and the increasing attention they are gaining in almost every domain of expertise including reinforcement learning, control systems, robotics, modeling, and real-time prediction in sensitive and safety-critical environments, it has become paramount to start considering backup safety filters for these generally unstable large function approximators in safety-critical environments. We have presented all the essential components of the python version of the `LevelSetPy` library for numerically resolving HJ PDEs (pertaining to the problems in the foregoing) and for advancing co-dimension one interfaces on Cartesian grids. We

have motivated the work presented with several numerical examples to demonstrate the efficacy of our library.

The examples we have presented in this document provide a foray into the computational aspects of ascertaining the safety (freedom from harm) of the complex systems that we are continually designing and building. This includes safety-critical analysis encompassing simple and complex multiagent systems whose safe navigation over a phase space can be considered in an HJ PDE framework. As complexity evolves, we hope that our library can serve as a useful tool for tinkerers looking for an easy-to-use proof-of-concept toolkit for verification of dynamical systems based on a principled numerical analysis.

We encourage users to download the library from the author's github webpage (it is available in `CPU` and `GPU` implementations via appropriately tagged branches) and use it robustly for various problems of interest where speed and scale for the solubility of hyperbolic conservation laws and HJ PDE's are of high importance.

REFERENCES

[1] Michael Athans and Peter L Falb. 2013. *Optimal Control: An Introduction to the Theory and its Applications.* Courier Corporation.

[2] M. Ballerini, N. Cabibbo, R. Candelier, A. Cavagna, E. Cisbani, I. Giardina, V. Lecomte, A. Orlandi, G. Parisi, A. Procaccini, M. Viale, and V. Zdravkovic. 2008. interaction Ruling Animal Collective Behavior Depends On Topological Rather Than Metric Distance: Evidence From A Field Study. *Proceedings of the National Academy of Sciences* 105, 4 (2008), 1232–1237. https://doi.org/10.1073/pnas.0711437105 arXiv:https://www.pnas.org/content/105/4/1232.full.pdf

[3] Tamer Basar and Geert Jan Olsder. 1999. *Dynamic Noncooperative Game Theory*. Vol. 23. Society for Industrial and Applied Mathematics.

[4] Richard Bellman. 1957. *Dynamic programming*. Princeton University Press.

[5] Sanjay P Bhat and Dennis S Bernstein. 1998. Continuous finite-time stabilization of the translational and rotational double integrators. *IEEE Transactions on automatic control* 43, 5 (1998), 678–682.

[6] Burridge, R R and Rizzi, A A and Koditschek, D E. 1999. *Sequential Composition of Dynamically Dexterous Robot Behaviors*. Technical Report 6. 534–555 pages.

[7] P Cardaliaguet. 1997. Nonsmooth Semipermeable Barriers, Isaacs' Equation, And Application to a Differential Game with One Target and Two Players. *Applied Mathematics and Optimization* 36, 2 (1997), 125–146.

[8] Andrea Cavagna, Alessio Cimarelli, Irene Giardina, Giorgio Parisi, Raffaele Santagati, Fabio Stefanini, and Massimiliano Viale. 2010. Scale-fFee Correlations In Starling Flocks. *Proceedings of the National Academy of Sciences* 107, 26 (2010), 11865–11870.

[9] M. G. Crandall, L. C. Evans, and P. L. Lions. 1984. Some Properties of Viscosity Solutions of Hamilton-Jacobi Equations. *Trans. Amer. Math. Soc.* 282, 2 (1984), 487.

[10] Michael G Crandall and Pierre-Louis Lions. 1983. Viscosity solutions of Hamilton-Jacobi equations. *Transactions of the American mathematical society* 277, 1 (1983), 1–42.

[11] Michael G Crandall and P-L Lions. 1984. Two Approximations of Solutions of Hamilton-Jacobi Equations. *Mathematics of Computation* 43, 167 (1984), 1–19.

[12] Michael G Crandall and Andrew Majda. 1980. Monotone Difference Approximations For Scalar Conservation Laws. *Math. Comp.* 34, 149 (1980), 1–21.

[13] Defense Acquisition University. 2023. Validation. https://www.dau.edu/tools/se-brainbook/Pages/Technical%20Processes/validation.aspx Accessed April 5, 2023.

[14] Defense Acquisition University. 2023. Verification. https://www.dau.edu/tools/se-brainbook/Pages/Technical%20Processes/verification.aspx Accessed April 5, 2023.

[15] Stuart E Dreyfus. 1966. *Control Problems With Linear Dynamics, Quadratic Criterion, and Linear Terminal Constraints*. Technical Report. Rand Corp, Santa Monica Calif.

[16] Lester E Dubins. 1957. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics* 79, 3 (1957), 497–516.

[17] L.C. Evans and Panagiotis E. Souganidis. 1984. Differential Games And Representation Formulas For Solutions Of Hamilton-Jacobi-Isaacs Equations. *Indiana Univ. Math. J* 33, 5 (1984), 773–797.

[18] L.C. Evans and Panagiotis E. Souganidis. 1984. Differential games and representation formulas for solutions of Hamilton-Jacobi-Isaacs equations. *Indiana Univ. Math. J* 33, 5 (1984), 773–797.

[19] Lawrence C Evans. 2022. *Partial Differential Equations*. Vol. 19. American Mathematical Society.

[20] Melanie Haiken. 2021. Starling murmurations are dazzling, ubiquitous, and puzzling. https://tinyurl.com/4973byey Accessed April 5, 2023.

[21] Melanie Haiken. 2021. These birds flock in mesmerizing swarms of thousands—but why is still a mystery. https://tinyurl.com/4973byey Accessed April 5, 2023.

[22] Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. 2020. Array programming with NumPy. *Nature* 585, 7825 (2020), 357–362.

[23] Dirk Helbing, Illés Farkas, and Tamas Vicsek. 2000. Simulating dynamical features of escape panic. *Nature* 407, 6803 (2000), 487–490.

[24] Kerianne L. Hobbs, Mark L. Mote, Matthew C.L. Abate, Samuel D. Coogan, and Eric M. Feron. 2023. Runtime Assurance for Safety-Critical Systems. *IEEE Control Systems Magazine* 43 (2023), 28–65. Issue 2.

[25] Eberhard Hopf. 1950. The Partial Differential Equation $u_t + uu_x = \mu^{\star}_{xx}$. (1950).

[26] R Isaacs. 1999. *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization.* Kreiger, Huntigton, NY.

[27] Ali Jadbabaie, Jie Lin, and A Stephen Morse. 2003. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on automatic control* 48, 6 (2003), 988–1001.

[28] Guang-Shan Jiang and Danping Peng. 2000. Weighted ENO schemes for Hamilton–Jacobi equations. *SIAM Journal on Scientific computing* 21, 6 (2000), 2126–2143.
[29] Elia Kaufmann, Antonio Loquercio, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. 2020. Deep Drone Acrobatics. *arXiv preprint arXiv:2006.05768* (2020).
[30] S.N. Kruzkov. 1970. First Order Quasilinear Equations In Several Independent Variables. *Mathematics of the USSR-Sbornik* (1970).
[31] Pierre-Louis Lions. 1982. *Generalized solutions of Hamilton-Jacobi equations*. Vol. 69. London Pitman.
[32] John Lygeros. 2004. On reachability and minimum cost optimal control. *Automatica* 40, 6 (2004), 917–927.
[33] AW Merz. 1972. The game of two identical cars. *Journal of Optimization Theory and Applications* 9, 5 (1972), 324–343.
[34] Ian Mitchell. 2001. Games of two identical vehicles. *Dept. Aeronautics and Astronautics, Stanford Univ.* July (2001), 1–29.
[35] Ian Mitchell. 2004. A Toolbox of Level Set Methods, version 1.0. *The University of British Columbia, UBC CS TR-2004-09* (July 2004), 1–94.
[36] Ian Mitchell. 2020. A Robust Controlled Backward Reach Tube with (Almost) Analytic Solution for Two Dubins Cars. *EPiC Series in Computing* 74 (2020), 242–258.
[37] Ian M. Mitchell, Alexandre M. Bayen, and Claire J. Tomlin. 2005. A Time-Dependent Hamilton-Jacobi Formulation of Reachable Sets for Continuous Dynamic Games. *IEEE Trans. Automat. Control* 50, 7 (2005), 947–957.
[38] Ryosuke Okuta, Yuya Unno, Daisuke Nishino, Shohei Hido, and Crissman Loomis. 2017. CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*.
[39] S Osher and R Fedkiw. 2004. Level Set Methods and Dynamic Implicit Surfaces. *Applied Mechanics Reviews* 57, 3 (2004), B15–B15.
[40] Stanley Osher and James A. Sethian. 1988. Fronts Propagating with Curvature-Dependent Speed: Algorithms based on Hamilton-Jacobi Formulations. *Journal of Computational Physics* 79, 1 (1988), 12–49.
[41] Stanley Osher and Chi-Wang Shu. 1988. *Efficient Implementation of Essentially Non-oscillatory Shock-capturing Schemes*. Technical Report 2. Hampton, Virginia. 439–471 pages.
[42] Stanley Osher and Chi-Wang Shu. 1991. High-Order Essentially Nonoscillatory Schemes for Hamilton-Jacobi Equations. *SIAM Journal of Numerical Analysis* 28, 4 (1991), 907–922.
[43] James A Sethian. 1996. A Fast Marching Level Set Method For Monotonically Advancing Fronts. *Proceedings of the National Academy of Sciences* 93, 4 (1996), 1591–1595.
[44] James A. Sethian. 2000. Level Set Methods And Fast Marching Methods: Evolving Interfaces In Computational Geometry, Fluid Mechanics, Computer Vision, And Materials Science. *Robotica* 18, 1 (2000), 89–92.
[45] Chi-Wang Shu and Stanley Osher. 1989. Efficient Implementation of Essentially Non-oscillatory Shock-capturing Schemes, II. *Journal of computational physics* 83, 1 (1989), 32–78.
[46] W Murray Wonham. 1985. Linear Multivariable Control: A Geometric Approach. *Applications of Mathematics* 10 (1985).

A HAMILTONIAN OF A MURMURATION.

In this appendix, we provide a derivation for the overall Hamiltonian of a flock as elucidated in Theorem 1.

PROOF OF THEOREM 1. We write the free agents' Hamiltonians in absolute coordinates and that of the agent under attack in relative coordinates with the pursuer. Henceforth, we drop the templated time arguments for ease of readability. The overall flock's Hamiltonian is

$$\cup_{i=1}^{n_a-1} H_f^{(i)_j}(\boldsymbol{x}, p) = \cup_{i=1}^{n_a-1} \begin{bmatrix} p_1^{(i)_j} & p_2^{(i)_j} & p_3^{(i)_j} \end{bmatrix} \begin{bmatrix} v^{(i)_j} \cos \boldsymbol{x}_3 \\ v^{(i)_j} \sin \boldsymbol{x}_3 \\ \langle w_e^{(i)_j} \rangle_r \end{bmatrix}. \tag{94}$$

It follows that

$$\cup_{i=1}^{n_a-1} H_f^{(i)_j}(\boldsymbol{x}, p) = \cup_{i=1}^{n_a-1} \left[p_1^{(i)_j} v^{(i)_j} \cos \boldsymbol{x}_3 + p_2^{(i)_j} v^{(i)_j} \sin \boldsymbol{x}_3 + p_3^{(i)_j} \langle w_e^{(i)_j} \rangle_r \right]. \tag{95}$$

Equation (83) can be re-written as

$$\boldsymbol{H}_a^{(k)_j}(\boldsymbol{x},p) = -\left(\max_{w_e^{(k)_j}\in[\underline{w}_e^j,\bar{w}_e^j]} \min_{w_p^{(k)_j}\in[\underline{w}_p^j,\bar{w}_p^j]} \left[p_1^{(k)_j}(t) \quad p_2^{(k)_j}(t) \quad p_3^{(k)_j}(t) \right] \right.$$
$$\left. \begin{bmatrix} -v_e^{(k)_j}(t) + v_p^{(j)} \cos \boldsymbol{x}_3^{(k)_j}(t) + \langle w_e^{(k)_j} \rangle_r(t) \boldsymbol{x}_2^{(k)_j}(t) \\ v_p^j(t) \sin \boldsymbol{x}_3^{(k)_j}(t) - \langle w_e^{(k)_j} \rangle_r(t) \boldsymbol{x}_1^{(k)_j}(t) \\ w_p^j(t) - \langle w_e^{(k)_j}(t) \rangle_r \end{bmatrix} \right), \tag{96}$$

where $p_l^{(k)_j}(t)\ |_{l=1,2,3}$ are the adjoint or co-state vectors [33]. For the pursuer, its minimum and maximum turn rates are fixed so that we have $\underline{w}_p^j$ as the minimum turn bound of the pursuing vehicle, and $\bar{w}_p^j$ is the maximum turn bound of the pursuing vehicle. Rewriting (95), we find that

$$\begin{aligned} \boldsymbol{H}_a^{(k)_j}(\boldsymbol{x},p) &= -\left(\max_{w_e^{(k)_j}\in[\underline{w}_e^j,\bar{w}_e^j]} \min_{w_p^{(k)_j}\in[\underline{w}_p^j,\bar{w}_p^j]} \left[-p_1^{(k)_j} v_e^{(k)_j} + p_1^{(k)_j} v_p^j \cos \boldsymbol{x}_3^{(k)_j} \right.\right. \\ &\left.\left. + p_1^{(k)_j} \langle w_e^{(k)_j} \rangle_r \boldsymbol{x}_2^{(k)_j} + p_2^{(k)_j} v_p^j \sin \boldsymbol{x}_3^{(k)_j} - p_2^{(k)_j} \langle w_e^{(k)_j} \rangle_r \boldsymbol{x}_1^{(i)_j} + p_3^{(k)_j} \left(w_p^j - \langle w_e^{(k)} \rangle_r \right) \right] \right), \\ &= p_1^{(k)_j} \left(v_e^{(k)_j} - v_p^j \cos \boldsymbol{x}_3^{(k)_j} \right) - p_2^{(k)_j} v_p^j \sin \boldsymbol{x}_3^{(k)_j} \\ &+ \left(\max_{\langle w_e^{(k)_j} \rangle_r \in [\underline{w}_e^j,\bar{w}_e^j]} \min_{w_p^j\in[\underline{w}_p^j,\bar{w}_p^j]} \left[\langle w_e^{(k)_j} \rangle_r \left(p_2^{(k)_j} \boldsymbol{x}_1^{(k)_j} - p_1^{(k)_j} \boldsymbol{x}_2^{(k)_j} + p_3^{(k)_j} \right) - p_3^{(k)_j} w_p^j \right] \right). \end{aligned} \tag{97}$$

It follows that we have from (97) that

$$\begin{aligned} \boldsymbol{H}_a^{(k)_j}(\boldsymbol{x},p) = p_1^{(k)_j} \left(v_e^{(k)_j} - v_p^j \cos \boldsymbol{x}_3^{(k)_j} \right) - p_2^{(k)_j} v_p^j \sin \boldsymbol{x}_3^{(k)_j} - \underline{w}_p^j |p_3^{(k)_j}| \\ + \bar{w}_e^j \left| p_2^{(k)_j} \boldsymbol{x}_1^{(k)_j} - p_1^{(k)_j} \boldsymbol{x}_2^{(k)_j} + p_3^{(k)_j} \right| \end{aligned} \tag{98}$$

and that

$$\boldsymbol{H}_f^{(i)_j}(\boldsymbol{x},p) = \left[p_1^{(i)_j} v^{(i)_j} \cos \boldsymbol{x}_3 + p_3^{(i)_j} v^{(i)_j} \sin \boldsymbol{x}_3 + p_3^{(i)_j} \langle w_e^{(i)_j} \rangle_r \right]. \tag{99}$$

A fortiori the main equation (83) becomes

$$\begin{aligned} \boldsymbol{H}(\boldsymbol{x},p) = \cup_{j=1}^{n_f} \left(\cup_{i=1}^{n_a-1} \left[p_1^{(i)_j} v^{(i)_j} \cos \boldsymbol{x}_3 + p_2^{(i)_j} v^{(i)_j} \sin \boldsymbol{x}_3 + p_3^{(i)_j} \langle w_e^{(i)_j} \rangle_r \right] \right. \\ \cup \left[p_1^{(k)_j} \left(v^{(k)_j} - v^{(k)_j} \cos \boldsymbol{x}_3^{(k)_j} \right) - p_2^{(k)_j} v^{(k)_j} \sin \boldsymbol{x}_3^{(k)_j} - \underline{w}_p^j |p_3^{(k)_j}| \right. \\ \left.\left. + \bar{w}_e^j \left| p_2^{(k)_j} \boldsymbol{x}_1^{(k)_j} - p_1^{(k)_j} \boldsymbol{x}_2^{(k)_j} + p_3^{(k)_j} \right| \right] \right). \end{aligned} \tag{100}$$

For the special case where the linear speeds of the evading agents and pursuer are equal i.e. $v_e^{(i)_j}(t) = v_p(t) = +1m/s$, we have a murmuration's Hamiltonian as

$$\begin{aligned} \boldsymbol{H}(\boldsymbol{x},p) = \cup_{j=1}^{n_f} \left(\cup_{i=1}^{n_a-1} \left[p_1^{(i)_j} \cos \boldsymbol{x}_3 + p_2^{(i)_j} \sin \boldsymbol{x}_3 + p_3^{(i)_j} \langle w_e^{(i)_j} \rangle_r \right] \right. \\ \cup \left[p_1^{(k)_j} \left(1 - \cos \boldsymbol{x}_3^{(k)_j} \right) - p_2^{(k)_j} \sin \boldsymbol{x}_3^{(k)_j} - \underline{w}_p^j |p_3^{(k)_j}| \right. \\ \left.\left. + \bar{w}_e^j \left| p_2^{(k)_j} \boldsymbol{x}_1^{(k)_j} - p_1^{(k)_j} \boldsymbol{x}_2^{(k)_j} + p_3^{(k)_j} \right| \right] \right). \end{aligned} \tag{101}$$

□

Received 10 April 2023