2017 IEEE 56th Annual Conference on Decision and Control (CDC)
December 12-15, 2017, Melbourne, Australia

Hamilton-Jacobi Reachability: A Brief Overview and Recent Advances

Somil Bansal*, Mo Chen*, Sylvia Herbert* and Claire J. Tomlin

Abstract— **Hamilton-Jacobi (HJ) reachability analysis is an important formal verification method for guaranteeing performance and safety properties of dynamical systems; it has been applied to many small-scale systems in the past decade. Its advantages include compatibility with general nonlinear system dynamics, formal treatment of bounded disturbances, and the availability of well-developed numerical tools. The main challenge is addressing its exponential computational complexity with respect to the number of state variables. In this tutorial, we present an overview of basic HJ reachability theory and provide instructions for using the most recent numerical tools, including an efficient GPU-parallelized implementation of a Level Set Toolbox for computing reachable sets. In addition, we review some of the current work in high-dimensional HJ reachability to show how the dimensionality challenge can be alleviated via various general theoretical and application-specific insights.**

I. Introduction

As the systems we design grow more complex, determining whether they work according to specification becomes more difficult. Consequently, verification and validation have received major attention in many fields of engineering. However, verification of systems is challenging for many reasons. First, all possible system behaviors must be accounted for. This makes most simulation-based approaches insufficient, and thus formal verification methods are needed. Second, many practical systems are affected by disturbances in the environment, which can be unpredictable, and may even contain adversarial agents. In addition, these systems often have high dimensional state spaces and evolve in continuous time with complex, nonlinear dynamics.

Hamilton-Jacobi (HJ) reachability analysis is a verification method for guaranteeing performance and safety properties of systems, overcoming some of the above challenges. In reachability analysis, one computes the reach-avoid set, defined as the set of states from which the system can be driven to a target set while satisfying time-varying state constraints at all times. A major practical appeal of this approach stems from the availability of modern numerical tools, which can compute various definitions of reachable sets [1]–[4]. For example, these numerical tools have been successfully used to solve a variety of differential games, path planning problems, and optimal control problems. Concrete practical applications include aircraft auto-landing [5], automated aerial refueling [6], model predictive control (MPC) of quadrotors [7], [8], multiplayer reach-avoid games [9], large-scale multiple-vehicle path planning [10], [11], and real-time safe motion planning [12]. However, HJ reachability becomes computationally intractable as the state space dimension increases. Traditionally, reachable set computations involve solving an HJ partial differential equation (PDE) on a grid representing a discretization of the state space, resulting in an *exponential* scaling of computational complexity with respect to system dimensionality; this is often referred to as the "curse of dimensionality." However, recent work has made a significant leap in overcoming these challenges by exploiting system structures to decompose the computation of reachable set into several small dimensional computations [13], [14]. In addition, convex optimization applied to the Hopf-Lax formula allows real-time computation of the HJ PDE solution at any desired state and time instant when the system dynamics are linear [15], [16].

Besides HJ reachability, alternative approaches to verification exist. In particular, satisfaction of properties such as safety, liveness, and fairness in computer software and in discrete-time dynamical systems can be verified by checking whether runs of a transition system, or words of a finite automaton satisfy certain desired properties [17], [18]. These properties may be specified by a variety of logical formalisms such as linear temporal logic. For specifications of properties of interest in autonomous robots, richer formalisms have been proposed. For example, propositional temporal logic over the reals [19], [20] allows specification of properties such as time in terms of real numbers, and chance-constrained temporal logic [21] allows specification of requirements in the presence of uncertainty. Besides autonomous cars and robots, verification approaches based on discrete models have also been successfully used in the context of intelligent transportation systems [22] and human-automation interaction [23].

For continuous and hybrid systems, safety properties can be verified by checking whether the forward reachable set or an over-approximation of it intersects with a set of undesirable states, akin to checking runs of transition systems. Numerous tools such as SpaceEx [24], Flow* [25], CORA [26], C2E2 [27], [28], and dReach [29] have been developed for this purpose; the authors in [30] present a tutorial on combining different tools for hybrid systems verification. In addition, methods that utilize semidefinite programming to search for Lyapunov functions can be used to verify safety [31], [32]. This is done, for example, by constructing

* All authors contributed equally to this article. Authors' names are written in the alphabetical order. All authors are with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. {somil, mochen72, sylvia.herbert, tomlin}@eecs.berkeley.edu

This tutorial is supported by NSF under the CPS Frontiers VehiCal project (1545126) and CPS:ActionWebs (CNS-931843), by the UC-Philippine-California Advanced Research Institute under project IIID-2016-005, by the ONR MURI Embedded Humans (N00014-16-1-2206), and by NASA under grants NNX12AR18A and UCSCMCA-14-022 (UARC).

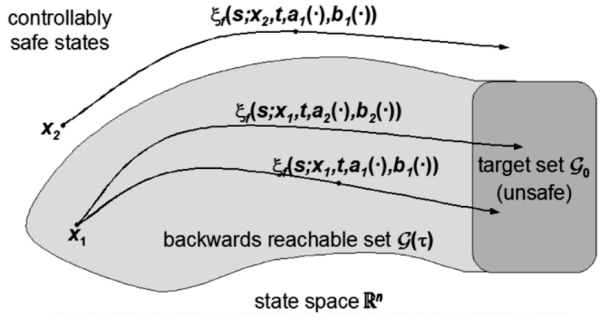


Fig. 1: Target set and backward reachable set. Several trajectories are shown starting at the same time t but from different states x and subject to different input signals $a(\cdot)$ and $b(\cdot)$. Input signal $a(\cdot)$ is chosen to drive the trajectory away from the target set, while input signal $b(\cdot)$ is chosen to drive the trajectory toward the target. Figure taken from [37].

barrier certificates [33] or funnels [34], [35] with Lyapunov properties.

Outside of the realm of checking whether the set of possible future states of a system includes undesirable states, safety can also be verified by starting from known unsafe conditions and computing backward reachable sets, which the system should avoid. In general, the challenges facing verification methods include computational tractability, generality of system dynamics, existence of control and disturbance variables, and representation of sets [36]–[39]. HJ reachability can be distinguished from other methods because it is applicable to general nonlinear systems, easily handles control and disturbance variables, and is able to represent sets of arbitrary shapes. However, this flexibility comes with the cost of computational complexity. Other backward reachability methods make other trade-offs. For example, [24], [40]–[42] present scalable methods for affine systems that rely on polytopic or ellipsoidal representation of sets, while the methods presented in [43]–[45] are well-suited to systems with polynomial dynamics.

The goal of this tutorial is four-fold. First, we aim to provide a formal and self-contained introduction to reachability theory. Second, we familiarize the readers with some of the available tools for the computation of reachable sets. Third, we provide an overview of the recent developments in reachability theory that help overcome the curse of dimensionality. Finally, we illustrate some of the recent applications of reachability theory in the verification of safety-critical systems.

II. Backward Reachable Set (BRS)

In reachability theory, we are often interested in computing the *backward reachable set* of a dynamical system. This is the set of states such that the trajectories that start from this set can reach some given target set (see Figure 1). If the target set consists of those states that are known to be unsafe, then the BRS contains states which are potentially unsafe and should therefore be avoided. As an example, consider collision avoidance protocols for two aircraft in En-Route airspace. The target set would contain those states that are already "in loss of separation," such as those states in which the aircraft are within the five mile horizontal separation distance mandated by the Federal Aviation Administration. The backward reachable set contains those states which could lead to a collision, despite the best possible control actions. We typically formulate such safety-critical scenarios in terms of a two-player game, with Player 1 and Player 2 being control inputs. For example, Player 1 could represent one aircraft, Player 2 another, with Player 1's control input being treated as the control input of the joint system, and with Player 2's control input being treated as the disturbance.

Mathematically, let $x \in \mathbb{R}^n$ be the system state, which evolves according to the ordinary differential equation (ODE)

$$\dot{x}(s) = f(x(s), a(s), b(s)), s \in [t, 0], a(s) \in \mathcal{A}, b(s) \in \mathcal{B}, \tag{1}$$

where $a(s)$ and $b(s)$ denote the input for Player 1 and Player 2 respectively. We assume that the control functions $a(\cdot)$, $b(\cdot)$ are drawn from the set of measurable functions[1]:

$$\begin{aligned} a(\cdot) \in \mathbb{A}(t) =& \{\phi : [t, 0] \to \mathcal{A} : \phi(\cdot) \text{ is measurable}\} \\ b(\cdot) \in \mathbb{B}(t) =& \{\phi : [t, 0] \to \mathcal{B} : \phi(\cdot) \text{ is measurable}\} \end{aligned}$$

where $\mathcal{A} \subset \mathbb{R}^{n_u}$ and $\mathcal{B} \subset \mathbb{R}^{n_d}$ are compact and $t < 0$. The system dynamics, or flow field, $f : \mathbb{R}^n \times \mathcal{A} \times \mathcal{B} \to \mathbb{R}^n$ is assumed to be uniformly continuous, bounded, and Lipschitz continuous in x uniformly in[2] a and b. Therefore, given $a(\cdot) \in \mathbb{A}$ and $b(\cdot) \in \mathbb{B}$, there exists a unique trajectory solving (1) [46]. We will denote solutions, or trajectories of (1) starting from state x at time t under control $a(\cdot)$ and $b(\cdot)$ as $\zeta(s; x, t, a(\cdot), b(\cdot)) : [t, 0] \to \mathbb{R}^n$. ζ satisfies (1) with an initial condition almost everywhere:

$$\begin{aligned} \frac{d}{ds}\zeta(s; x, t, a(\cdot), b(\cdot)) &= f(\zeta(s; x, t, a(\cdot), b(\cdot)), a(s), b(s)) \\ \zeta(t; x, t, a(\cdot), b(\cdot)) &= x \end{aligned} \tag{2}$$

Intuitively, a BRS represents the set of states $x \in \mathbb{R}^n$ from which the system can be driven into some set $\mathcal{G}_0 \subseteq \mathbb{R}^n$ at the *end* of a time horizon of duration $|t|$. We call $\mathcal{G}_0$ the "target set". We assume that Player 1 will try to steer the system away from the target with her input, and Player 2 will try to steer the system toward the target with her input. Consequently, we want to compute the following BRS:

$$\begin{aligned} \mathcal{G}(t) =& \{x : \exists \gamma \in \Gamma(t), \forall a(\cdot) \in \mathbb{A}, \\ & \zeta(0; x, t, a(\cdot), \gamma[a](\cdot)) \in \mathcal{G}_0\}, \end{aligned} \tag{3}$$

where $\Gamma(\cdot)$ in (3) denotes the feasible set of strategies for Player 2.

The computation of the BRS in (3) requires solving a differential game between Player 1 and Player 2 (more on

[1]A function $f : X \to Y$ between two measurable spaces (X, Σ_X) and (Y, Σ_Y) is said to be measurable if the preimage of a measurable set in Y is a measurable set in X, that is: $\forall V \in \Sigma_Y, f^{-1}(V) \in \Sigma_X$, with Σ_X, Σ_Y σ-algebras on X,Y.

[2]For the remainder of the tutorial, we will omit the notation (s) from variables such as x and a when referring to function values.

this in Section III). In a differential game setting, it is important to address what information the players know about each other's decisions which directly affects their strategies, and consequently, the outcome of the game. In reachability problems, we assume that the Player 2 uses only non-anticipative strategies $\Gamma(\cdot)$ [37], defined as follows:

$$\gamma \in \Gamma(t) := \{\mathcal{N} : \mathbb{A}(t) \to \mathbb{B}(t) : a(r) = \hat{a}(r) \text{ a. e. } r \in [t, s] \Rightarrow \mathcal{N}[a](r) = \mathcal{N}[\hat{a}](r) \text{ a. e. } r \in [t, s]\} \tag{4}$$

That is, Player 2 cannot respond differently to two Player 1 controls until they become different. Yet, in this setting, Player 2 has the advantage of factoring in Player 1's choice of input at every instant t and adapting its own accordingly. Thus, Player 2 has an *instantaneous informational advantage*, which allows us to establish safety guarantees under the worst-case scenarios. One particular class of problems in which the notion of non-anticipative strategies is applicable is robust control problems, in which one wants to obtain the robust control (Player 1) with respect to the worst-case disturbance (Player 2), which can then be modeled as an adversary with the instantaneous informational advantage (not because this disturbance is in fact reacting to the controller's input, but rather, because out of all possible disturbances there will be one that will happen to be the worst possible given the chosen control).

The differential game that must be solved in order to compute the BRS in (3) is a "game of kind" rather than a "game of degree", i.e., games in which the outcome is determined by *whether or not* the state of the system reaches a given configuration under specified constraints at any time within the duration of the game. The good news is that an approach known as the *level set method* can transform these games of kind into games of degree in an analytically sound and computationally tractable way. We first provide a brief overview of the theory of differential games and then explain how the problem of computing a BRS can be transformed into a differential game of degree using level set methods.

III. Two-person Zero-sum Differential Games

In many relevant differential game problems, the goal is to optimize a cost function of the final state and some running cost or reward accumulated over system trajectories. The system is steered towards this final state after a finite time horizon. Formally, let $J_t(x, a(\cdot), b(\cdot))$ denote the cost accumulated during horizon $[t, 0]$ when Player 1 and Player 2 play control $a(\cdot)$ and $b(\cdot)$, respectively. $J_t(\cdot)$ can be expressed as

$$J_t(x, a(\cdot), b(\cdot)) = \int_t^0 c(x(s), a(s), b(s), s)ds + q(x(0)) \tag{5}$$

In the zero-sum setting, Player 1 will attempt to maximize this outcome, while the Player 2 will aim to minimize it, subject to the system dynamics in (1). Under the non-anticipative strategy assumption, we can readily define the so-called *lower value*[3] of the game as

$$G(t, x) = \inf_{\gamma \in \Gamma(t)} \sup_{a(\cdot) \in \mathbb{A}} J_t(x, a(\cdot), \gamma[a](\cdot)), \tag{6}$$

where $\Gamma(\cdot)$ is defined in (4).

Using the principle of dynamic programming, it can be shown that the value function $G(t, x)$ in (6) is the viscosity solution [47] of the following Hamilton-Jacobi Isaacs (HJI) PDE:

$$D_t G(t, x) + H(t, x, \nabla G(t, x)) = 0, \quad G(0, x) = q(x), \tag{7}$$

where $H(t, x, \nabla G(t, x))$ is called the Hamiltonian and is given by

$$H(t, x, \lambda) = \max_{a \in \mathcal{A}} \min_{b \in \mathcal{B}} c(x, a, b, t) + \lambda \cdot f(x, a, b). \tag{8}$$

λ in (8) denotes $\nabla G(t, x)$ and is called the *costate*. Given the value function, the optimal control for Player 1 can be obtained as:

$$a^*(t, x) = \arg\max_{a \in \mathcal{A}} \min_{b \in \mathcal{B}} c(x, a, b, t) + \lambda \cdot f(x, a, b). \tag{9}$$

The optimal control for Player 2 can be similarly obtained. A more detailed discussion of this material can be found in [47].

IV. The Level Set Approach: From Games of Kind to Games of Degree

We are now ready to solve the original intended problem of this tutorial: the computation of BRS. In Section III, we discussed how the differential games of degree can be solved using an HJ PDE. The computation of the BRS, however, is a differential game of kind where the outcome is Boolean: the system either reaches the target set or not. It turns out that we can "encode" this Boolean outcome through a quantitative value function: for example, if we consider $J_t(\cdot)$ as the distance between the system state and the target region at the terminal state of the system, it is easy to determine whether the system reached the target by comparing this distance to some threshold value (simply 0 in this case). This allows us to find the solution to a game of kind by posing an auxiliary game of degree whose solution encodes that of the original problem: this is, in essence, the level set approach.

In particular, one can always find a Lipschitz function $g(x)$ such that $\mathcal{G}_0$ (the target set) is equal to the zero sublevel set of g, that is, $x \in \mathcal{G}_0 \Leftrightarrow g(x) \leq 0$. The Lipschitz function g can always be found, since one can always choose the signed distance to the respective sets. If we define the cost function to be

$$J_t(x, a(\cdot), b(\cdot)) = g(x(0)), \tag{10}$$

then the system reaches the target set under controls a and b if and only if $J_t(x, a(\cdot), b(\cdot)) \leq 0$. Since Player 2 wants to drive the system to the target, it wants to minimize the cost in (10), and Player 1 wants to maximize this cost. We can now compute the value function $G(t, x)$ for this differential

[3]Note that, in general, one needs to define both the upper and lower values of the game, but for the scenarios that we are interested in, the lower value will suffice.

game in a similar fashion to Section III. Consequently, the BRS can be obtained as

$$\mathcal{G}(t) = \{x : G(t,x) \leq 0\}, \tag{11}$$

where $G(t,x)$ satisfies the following HJI PDE:

$$D_t G(t,x) + H(t,x,\lambda) = 0, \quad G(0,x) = g(x). \tag{12}$$

The Hamiltonian is given by

$$H(t,x,\lambda) = \max_{a\in\mathcal{A}} \min_{b\in\mathcal{B}} \lambda \cdot f(x,a,b). \tag{13}$$

The interpretation of $\mathcal{G}(t)$ is that if $x(t) \in \mathcal{G}(t)$, then Player 2 has a control sequence that will drive the system to the target at time 0, irrespective of the control of Player 1. If $x(t) \in \partial\mathcal{G}(t)$, where $\partial\mathcal{G}(t)$ denotes the boundary of $\mathcal{G}(t)$, then Player 1 will *barely* miss the target at time 0 if it applies the optimal control

$$a^*(t,x) = \arg\max_{a\in\mathcal{A}} \min_{b\in\mathcal{B}} \lambda \cdot f(x,a,b). \tag{14}$$

Finally, if $x(t) \in \mathcal{G}(t)^C$, then Player 1 has a control sequence (given by (14)) that will keep the system out of the target set, irrespective of the control applied by Player 2. In particular, when the target set $\mathcal{G}_0$ represents unsafe/undesired states of the system and Player 2 represents the disturbances in the system, then $\mathcal{G}(t)$ represents the *effective* unsafe set, i.e., the set of states from which the disturbance can drive the system to the *actual* unsafe set despite the best control efforts. Thus, reachability analysis gives us the safe set (in this case $\mathcal{G}(t)^C$) as well as a controller (in this case $a^*(t,x)$) that will keep the system in the safe set, given that the system starts in the safe set.

V. Different flavors of reachability

So far, we have presented the computation of BRSs, but reachability analysis is not limited to BRSs. One can compute various other kinds of sets that may be more useful, depending on the verification problem at hand. In this section, we provide a brief overview of some of these sets.

A. Forward vs. Backward Reachable Set

In some cases, we might be interested in computing a forward reachable set (FRS): the set of all states that a system can reach from a given initial set of states after a time duration of $|t|$. Formally, we want to compute the following set:

$$\begin{aligned}\mathcal{W}(t) =& \{y : \exists\gamma \in \Gamma(t), \forall a(\cdot) \in \mathbb{A}, \\ & \zeta(t; x, 0, a(\cdot), \gamma[a](\cdot)) = y, x \in \mathcal{G}_0\}, t > 0.\end{aligned} \tag{15}$$

Here, $\mathcal{G}_0$ represents the set of initial states of system. $\mathcal{W}(t)$ is the set of all states that system can reach in a duration of t, while Player 1 applies the control to keep the system in $\mathcal{G}_0$ and Player 2 applies the control to drive the system out of $\mathcal{G}_0$. The FRS can be computed in a similar fashion as the BRS. The only difference is that an initial value HJ PDE needs to be solved instead of a final value PDE, which can always be converted into an equivalent final value PDE by change of variables [48]. More details on the computation of FRS and some of their concrete applications can be found in [10], [49].

B. Reachable Sets vs. Tubes

Another important aspect in reachability is that of reachable tubes. The reachable set is the set of states from which the system can reach a target at *exactly* time 0. Perhaps a more useful notion is to compute the set of states from which the system can reach a target *within* a duration of $|t|$. For example, for safety analysis, we are interested in verifying if a disturbance can drive the system to the unsafe states *ever* within a horizon, and not just at the end of the horizon. This notion is captured by reachable tubes. Here, we present the formal definition of backward reachable tube (BRT), but forward reachable tube (FRT) can be similarly defined:

$$\begin{aligned}\mathcal{G}(t) =& \{x : \exists\gamma \in \Gamma(t), \forall a(\cdot) \in \mathbb{A}, \\ & \exists s \in [t,0], \zeta(s; x, t, a(\cdot), \gamma[a](\cdot)) \in \mathcal{G}_0\}.\end{aligned} \tag{16}$$

Once again, the BRT can be computed by solving a final value PDE similar to that in (12) [37], [50].

C. Roles of the Control and Disturbance

Depending on the role of Player 1 and Player 2, we may need to use different max-min combinations. As a rule of thumb, whenever the existence of a control ("$\exists a$") is sought, the optimization is a minimum over the set of controls in the corresponding Hamiltonian. Whenever a set/tube characterizes the behavior of the system for all controls ("$\forall a$"), the optimization is a maximum. For example, for the BRS in (3), we sought the *existence* of a Player 2 controller *for all* Player 1 controls, so we used minimum for Player 2 and maximum for Player 1 in the Hamiltonian (see (13)). When the target set represents the set of the desired states that we want the system to reach and Player 2's control represents the disturbance, then we are interested in verifying if there exists a control of Player 1 such that the system reaches its target despite the worst-case disturbance. In this case, we should use maximum for Player 2's control and minimum for Player 1's control in the corresponding Hamiltonian.

D. Presence of State Constraints

Another interesting problem that arises in verification is the reachability to and from a target set subject to some state constraints; this can be handled efficiently for even time-dependent constraints within the reachability framework [39], [51]. In general, any combination of the above four variants can be solved using the HJ reachability formulation. Partially, it is this flexibility of the reachability framework that has facilitated its use in various safety-critical applications, some of which we will discuss in this tutorial.

VI. Computational Tools for HJ Reachability

In this section, we will present an overview of two available computational tools that can be used to compute different definitions of reachable sets.

A. The Level Set Toolbox (toolboxLS)

The level set toolbox (or *toolboxLS*) was developed by Professor Ian Mitchell [4] to solve partial differential equations using level set methods, and is the foundation of the HJ reachability code. The toolbox is implemented in MATLAB and is equipped to solve any final-value HJ PDE. Since different reachable set computations can be ultimately posed as solving a final-value HJ PDE (see Sections IV and V), the level set toolbox is fully equipped to compute various types of reachable sets. Information on how to install and use toolboxLS can be found here: *http://www.cs.ubc.ca/∼mitchell/ToolboxLS*. This toolbox can be further augmented by the Hamilton-Jacobi optimal control toolbox (or *helperOC*). A quick-start guide to using toolboxLS and helperOC is presented in the Appendix and is also available at: *http://www.github.com/HJReachability/helperOC*.

B. The Berkeley Efficient API in C++ for Level Set methods (BEACLS) Toolbox

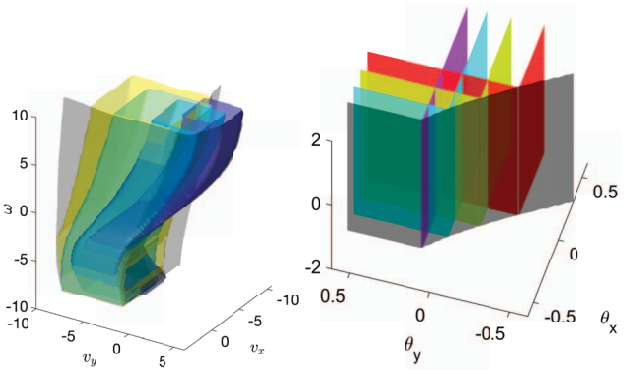
The Berkeley Efficient API in C++ for Level Set methods (*BEACLS*) Toolbox was developed by Ken Tanabe. This toolbox implements the functions from helperOC and toolboxLS in C++ for fast computation of reachability analyses. The library also uses GPUs for parallelizing different computations in the level set toolbox. The installation instructions and user guide can be found at: *http://www.github.com/HJReachability/beacls*. This GPU library has been used for large-scale multi-vehicle reachability problems, such as safe path planning (see Section VIII-B).

VII. Current Research in HJ Reachability Theory

Recently there have been several advances in HJ reachability theory and applications. Research on restructuring dynamics, new formulations for analysis, and the addition of learning techniques provided HJ reachability with a broadened and deeper span of feasible applications. These advances are used in safety-critical applications to provide safety guarantees, liveness properties, and optimal controllers.

A. System Decomposition Techniques for Nonlinear Systems

Decomposition methods address the exponentially scaling computational complexity of previous approaches for solving HJ reachability problems, which makes application to high-dimensional systems intractable. In [13], [52] a new technique is proposed that decomposes the dynamics of a general class of nonlinear systems into subsystems which may be coupled through common states, controls, and disturbances. Despite this coupling, BRSs and BRTs can be computed efficiently and exactly using this technique without the need for linearizing dynamics or approximating sets as polytopes. Computations of BRSs and BRTs now become orders of magnitude faster, and for the first time BRSs and BRTs for many high-dimensional nonlinear control systems can be exactly computed. In situations where the exact solution cannot be computed, this method can obtain slightly conservative results. The paper demonstrates this theory by numerically computing BRSs and BRTs for several systems, including the 6D Acrobatic Quadrotor and the 10D near-Hover Quadrotor. Reachable sets computed using the decomposition process are illustrated in Figure 2, with details in [13], [52].

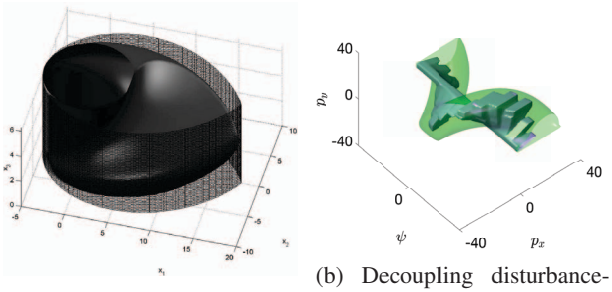


(a) BRS & BRT for a 6D quadrotor avoiding an obstacle.

(b) Reachable set and tube for a 10D quadrotor reaching a target.

Fig. 2: Decomposition results for nonlinear systems. Figures taken from [13].

In more general settings, approximate decomposition of nonlinear systems can be achieved by treating key states as disturbances, as in [14], [53]. These methods are able to maintain a direction of conservatism in order to provide guarantees on system performance and safety by either computing overapproximations or underapproximations of reachable sets and tubes. In [14], the authors also propose a way to trade off conservatism of the solution with computational cost.



(a) Projection-based approximation of a reachable tube. Figure taken from [53].

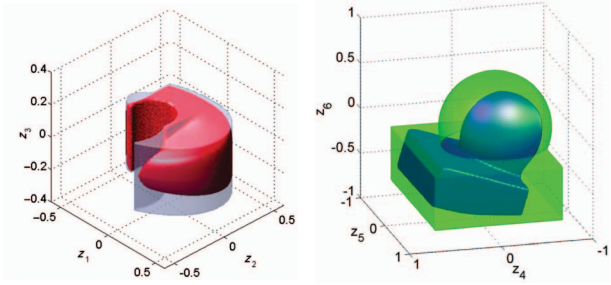
(b) Decoupling disturbance-based approximation of a reachable set. Figure taken from [14].

Fig. 3: Approximate decomposition results for nonlinear systems.

B. System Decomposition Techniques for Linear Time-Invariant Systems

In the linear time-invariant case, many non-HJ-based computation techniques have been developed for approximating reachable sets. In the area of HJ reachability, specific decomposition techniques also exist, and provide a substantial reduction in computational burden with a small degree of

conservatism. In [54], the authors proposed a Schur-based decomposition technique for computing reachable sets and synthesizing safety-preserving controllers. Subsystems are analyzed separately, and reachable sets of subsystems are back-projected and intersected to construct an overapproximation of the reachable set, so that safety can still be guaranteed. In [55], a similar approach based on a modified Riccati transformation is used. Here, decentralized computations are done in transformed coordinates of subspaces. The computation results are combined to obtain an approximation of the viability kernel, which is the complement of the reachable set. Figure 4 shows the conservative approximations obtained from these decomposition techniques.



(a) Overapproximation (translucent) of a reachable set (solid). Figure taken from [54].

(b) Constraint set (translucent) and the approximate viability kernel (solid). Figure taken from [55].

Fig. 4: Decomposition results for linear time-invariant systems.

C. Fast and Safe Tracking for Motion Planning

Fast and safe navigation of dynamical systems through a priori unknown cluttered environments is vital to many applications of autonomous systems. However, trajectory planning for autonomous systems is computationally intensive, often requiring simplified dynamics that sacrifice safety and dynamic feasibility in order to plan efficiently. Conversely, safe trajectories can be computed using more sophisticated dynamic models, but this is typically too slow to be used for real-time planning. In [12], a new algorithm is developed called FaSTrack: Fast and Safe Tracking. A path or trajectory planner using simplified dynamics to plan quickly can be incorporated into the FaSTrack framework, which provides a safety controller for the vehicle along with a guaranteed tracking error bound. By formulating a differential game and leveraging HJ reachability's flexibility with respect to nonlinear system dynamics, this tracking error bound is computed in the error coordinates, which evolve according to the error dynamics, and captures all possible deviations due to dynamic infeasibility of the planned path and external disturbances. Note that FaSTrack is modular and can be used with other path or trajectory planners. This framework is demonstrated using a 10D nonlinear quadrotor model tracking a 3D path obtained from an RRT planner, shown in Figure 5.

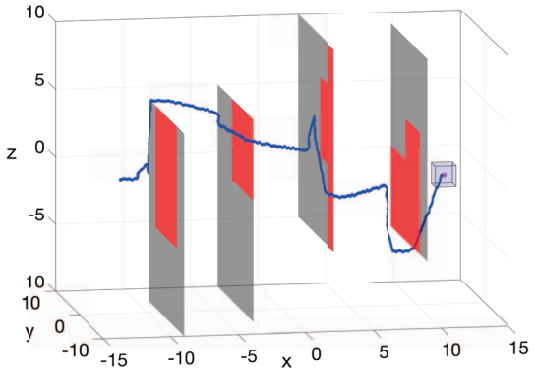
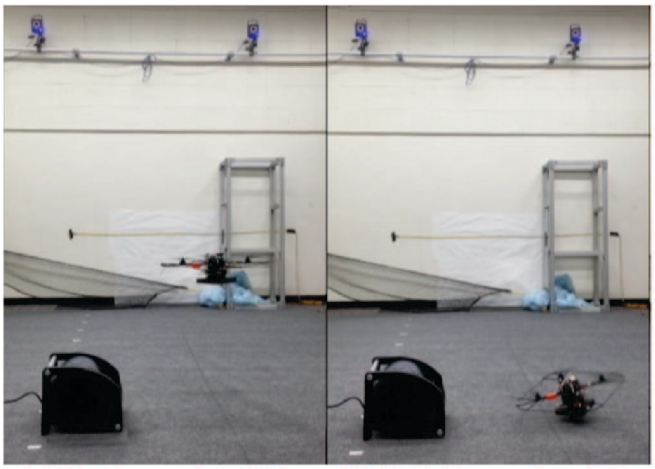


Fig. 5: Real-time safe planning using FaSTrack. Figure obtained from [12].

D. HJ Reachability for Safe Learning-Based Control

The proven efficacy of learning-based control schemes strongly motivates their application to robotic systems operating in the physical world. However, guaranteeing correct operation during the learning process is currently an unresolved issue, which is of vital importance in safety-critical systems.



(a) With online guarantee validation (b) Without online guarantee validation

Fig. 6: A Hummingbird UAV is able to successfully reject disturbances using online learning, and fails to do so without learning. Figure obtained from [56].

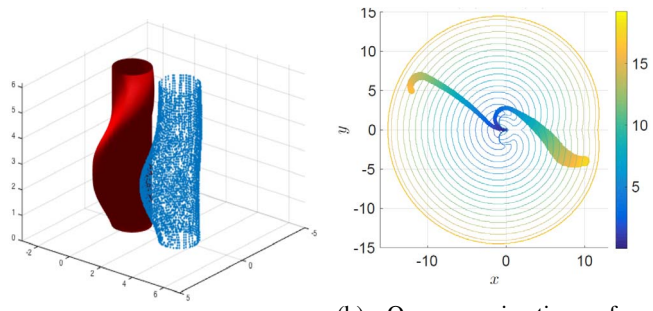
In [56], [57], a general safety framework is proposed based on HJ reachability methods that can work in conjunction with an arbitrary learning algorithm. The method exploits approximate knowledge of the system dynamics to guarantee constraint satisfaction while minimally interfering with the learning process. The authors further introduce a Bayesian mechanism that refines the safety analysis as the system acquires new evidence, reducing initial conservativeness when appropriate while strengthening guarantees through real-time validation. The result is a least-restrictive, safety-preserving control law that intervenes only when (a) the computed safety guarantees require it, or (b) confidence in the computed guarantees decays in light of new observations.

The authors provide safety guarantees combining probabilistic and worst-case analysis and demonstrate the proposed

framework experimentally on a quadrotor vehicle. Even though safety analysis is based on a simple point-mass model, the quadrotor is able to successfully run policy-gradient reinforcement learning without crashing, and safely retracts away from a strong external disturbance introduced during one of the experiments, as shown in Figure 6.

E. HJ Reachability Analysis using Neural Networks

Many of the recent breakthroughs in machine learning and AI have been possible thanks in part to the use of powerful function approximators, and in particular (deep) neural networks. In AI, these approximators are used to represent a myriad of complex functions such as Value functions, Q-functions and control policies, which often have high-dimensional data as inputs. In [58]–[61], the authors use these same tools in the context of reachability to approximate solutions of the HJ PDE by implementing and analyzing learning-based algorithms to approximate the solution of certain types of HJ PDEs using neural networks. Some recent results on 2D and 3D systems show that these learning-based algorithms require less memory to run and less memory to store the resulting approximation than traditional gridding-based methods. Further work involves exploring how well these algorithms scale with the number of dimensions in the state space, as well as the types of safety guarantees that can be derived from these types of approximations. In some cases, conservative guarantees for the computed value functions are possible despite the use of neural networks. Figure 7 shows preliminary results.



(a) Approximation a reachable set (red) using a neural network (point cloud). Figure taken from [60].

(b) Overapproximation of a value function (contours) using a neural network (color gradient) in small regions of the state space. Figure taken from [61].

Fig. 7: Neural network-based approximations of value functions representing reachable sets.

F. Generalized Hopf Formula for Linear Systems

In [15], [16], the authors proposed using a generalized Hopf formula for solving HJ PDEs arising from linear systems, which may be time-varying. Obtaining HJ PDE solutions here involves solving the minimization problem in the generalized Hopf formula. This minimization problem can be solved using any optimization algorithm; the authors suggest using coordinate descent with multiple initializations, as well as a numerical quadrature rule for an integral with respect to time. Alternative algorithms such as ADMM can also be used. By reformulating the problem of solving the HJ PDE as an optimization problem, the solution for HJ PDEs can be obtained at any desired points in state space and time, effectively alleviating the exponentially scaling computational complexity in finite difference-based methods. Figure 8 shows the results of this method.

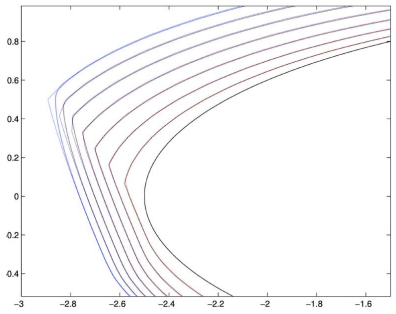


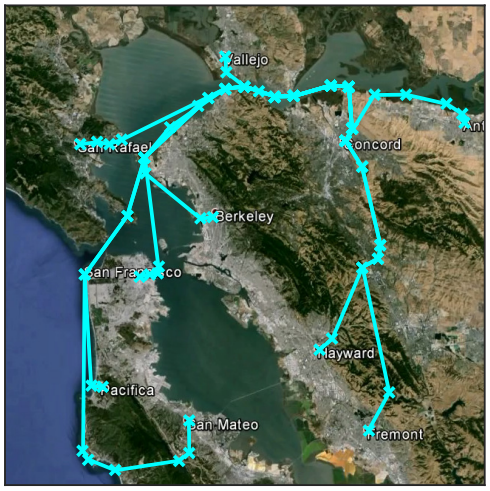
Fig. 8: Comparison between HJ PDE solutions obtained using the Hopf formula (colored) and using Lax-Friedrichs finite difference (black-and-white). Figure obtained from [16].

VIII. Some Current Applications of HJ Reachability

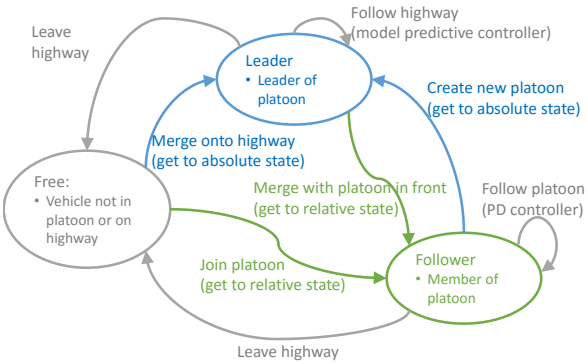
A. Unmanned Aerial Systems Traffic Management (UTM) using Air Highways

In collaboration with the National Aeronautics and Space Administration (NASA), HJ reachability has been applied to UTM [62]. In [11], [63], the authors proposed an efficient and flexible method for the placement of air highways, which are designated virtual pathways in the airspace. Air highways provide a scalable and intuitive way for monitoring and managing a large number of unmanned aerial vehicles (UAVs) flying in civilian airspace. The proposed method starts with a cost map encoding the desirability of having UAVs fly in different parts of a region, and computes minimum-cost paths connecting origins and destinations. These paths can be updated in real time according to changes in the airspace. Trunks and branches of air highways, similar to ground-based highway systems, naturally emerge from the proposed method. Applying the method to the San Francisco Bay Area, these air highways, which avoid urban areas and airports as much as possible, are shown in Figure 9a.

To fulfill potential traffic rules on the air highways, a hybrid system model for each UAV is used. On the highway system, a UAV can be in the “Free”, “Leader” or “Follower” modes. In this context, HJ reachability is used to ensure the success and safety of mode transitions. For example, the transition from the Free mode to the Leader mode involves using a controller from a maximal backward reachable set to arrive at a prescribed destination on the highway at a prescribed time. The highway and platoon structure greatly reduces the chance of multiple conflicts, enabling the use of pairwise safety analysis. Pairwise safety can be guaranteed using a minimal backward reachable set defined in the

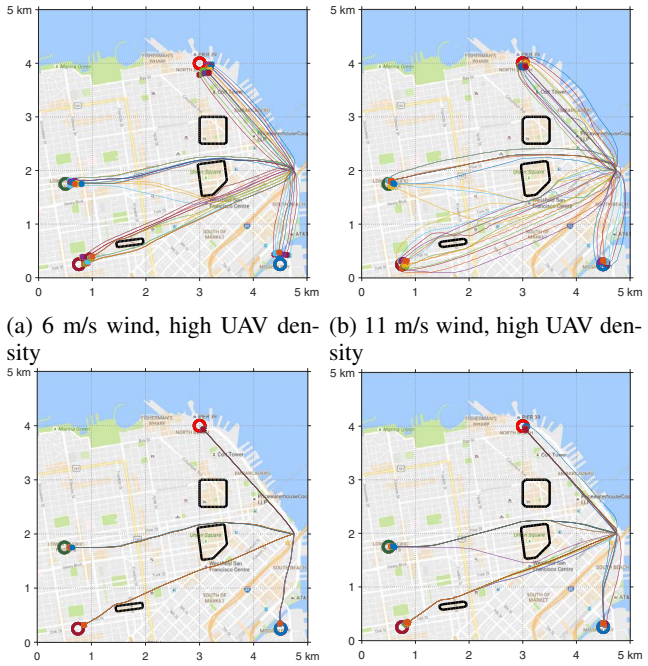


(a) Air highway placement over the San Francisco Bay Area.



(b) The purple vehicle is joining the platoon while avoiding collisions.

Fig. 9: The air highway and platooning concept for UTM. Figures are taken from [63].



(a) 6 m/s wind, high UAV density

(b) 11 m/s wind, high UAV density

(c) 6 m/s wind, medium UAV density

(d) 11 m/s wind, medium UAV density

Fig. 10: Natural lane forming of UAVs due to disturbance rejection and arrival time constraints. Figures taken from [64].

relative coordinates of two vehicles. The hybrid systems model is shown in Figure 9b. The proposed platooning concept has been implemented in the quadrotor lab at UC Berkeley on Crazyflies 2.0, which is an open source nano quadrotor platform developed by Bitcraze.

B. Sequential Robust Space-Time Reservations

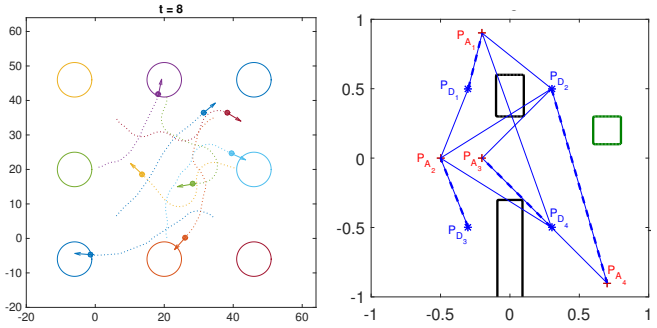
The trajectory planning of large-scale multi-robot systems has been addressed in work on sequential path planning [10], which robustly synthesizes controllers for many vehicles to reach their destinations while avoiding collisions under the presence of disturbances and a single intruder vehicle. Although reachability is well-suited for these robustness requirements, simultaneous analysis of all vehicles is intractable. Instead, vehicles are assigned a strict priority ordering, with lower-priority vehicles treating higher-priority vehicles as moving obstacles. Robust path planning around these induced obstacles is done using a novel time-varying formulation of reachability [39]. The result is a reserved "space-time" in the airspace for each vehicle, which can be used as a "last-mile" solution for getting from air highways to a final postal address. The space-time reservation is dynamically feasible to track even when the vehicle experiences disturbances and performs collision avoidance against an adversarial intruder. Simulations of the robust SPP method over San Francisco for different combinations of wind speeds and UAV densities are shown in Figure 10. Details can be found in [10], [64], [65].

C. Multi-Vehicle Coordination Using HJ Reachability and High-Level Logic

In [66], [67], the scalability limitations of HJ reachability are overcome by a mixed integer program that exploits the properties of pair-wise HJ solutions to provide higher-level control logic. This logic is applied in a couple of different contexts. First, safety guarantees for three-vehicle collision avoidance is proved – a previously intractable task for HJ reachability – without incurring significant additional computation cost [66]. The collision avoidance protocol method is also scalable beyond three vehicles and performs significantly better by several metrics than an extension of pairwise collision avoidance to multi-vehicle collision avoidance. Figure 11a shows an 8-vehicle collision avoidance simulation.

Second, in multiplayer reach-avoid games, two teams of cooperative players with conflicting and asymmetric goals play against each other on some domain, possibly with obstacles. The attacking team tries to arrive at some arbitrary target set in the domain, and the defending team seeks to prevent that by capturing attackers. Such a scenario is useful

for intercepting "rogue" UAVs trying to enter restricted areas of the airspace. The joint solution to this problem is intractable, so a maximum matching approach is taken instead. To each defender, the maximum matching process tries to assign an attacker who is guaranteed to lose to the defender, and the team of defenders coordinate the vehicle-to-vehicle defense. As a result, an upper bound on the number of attackers that can reach the target set can be obtained [67]. The maximum matching result for a particular game setup is shown in Figure 11b.



(a) Multi-vehicle collision avoidance simulation. Figure taken from [66].

(b) The maximum matching process for rogue UAV interception. Figure taken from [67].

Fig. 11: Multi-vehicle analysis using HJ reachability and higher-level logic.

IX. Conclusions

Hamilton-Jacobi (HJ) reachability is a useful tool for guaranteeing goal satisfaction and safety under controlled safety-critical scenarios with bounded disturbances. However, a direct application of HJ reachability in most cases becomes intractable due to its exponentially-scaled computational complexity with respect to the continuous state dimension. In this tutorial, we start from a comprehensive overview of HJ reachability theory from its roots in differential games theory. We then provide an overview of the recent theoretical work that aims at alleviating the curse of dimensionality, including several applications that leverage these ideas to ensure safety.

Acknowledgment

The authors would like to thank Jaime F. Fisac whose write up on differential games was immensely helpful in preparing this tutorial document.

X. Appendix: Quick-Start Guide

To familiarize ourselves with the tools available in *toolboxLS* and *helperOC*, we will walk through a simple example file to run several different forms of reachability analysis for a 3D Dubins car example.

A. Defining and Handling Dynamic Systems

Before setting up the analyses, we must first understand how to use the code to create and handle dynamic systems like the 3D Dubins car. In *helperOC* we use object-oriented code to define our dynamics. This allows us to create, for example, a Dubins car "object" that inherits the properties and functions related to a dynamic system and its own dynamics and parameters. In this section we will review the class structure.

The dynamic systems class is found under *helperOC/dynSys/@DynSys*. This class defines several properties and functions inherent to any dynamical system used for reachability analysis. All systems in *helperOC* are sub-classes of *@DynSys*. The sub-classes are also found in *helperOC/dynSys*. For now we will review the sub-class *@DubinsCar* as an example. This folder contains four files: *DubinsCar.m, dynamics.m, optCtrl.m*, and *optDstb.m.* The dynamics of the Dubins car is defined by:

$$\begin{bmatrix} \dot{p_x} \\ \dot{p_y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos(\theta) + b_1 \\ v\sin(\theta) + b_2 \\ a + b_3 \end{bmatrix} \tag{17}$$

$$a \in \mathcal{A}, \quad b = [b_1, b_2, b_3] \in \mathcal{B}$$

where a is the control, and b is the disturbance. *DubinsCar.m* is the main function of the Dubins car sub-class. This function defines the properties of a Dubins car (e.g. speed v, angular control a, and disturbance b), as well as the constructor function for creating a Dubins car object. This function takes in the object parameters and constructs a Dubins car object with said parameters.

The function *dynamics.m* sets the dynamics for the system. Open this file for a demonstration of how to incorporate the dynamics of your system into the code. Note that the inputs are the object, state, control, and disturbance. For time-varying systems, a time input can be included as well.

The functions *optCtrl.m* and *optDstb.m* are used to find the optimal control and disturbance at every grid point in the state space for each time step. These functions are determined by taking the inner product between the spatial gradients of the value function and the system dynamics, i.e. by computing the Hamiltonian, as required in equation (13), and as of now must be defined by hand. The control that either maximizes or minimizes the Hamiltonian (depending on what is desired) is the optimal control. The optimal disturbance does the opposite. As an example we will derive the optimal control and disturbance for the Dubins car in the case where uMode = 'min' and dMode = 'max.' Note that we will use G_{p_x} to denote the partial derivative of the value function with respect to state p_x.

$$\begin{gathered} \nabla G \cdot f(x,a,b) \\ = G_{p_x}(v\cos(\theta) + b_1) + G_{p_y}(v\sin(\theta) + b_2) + G_\theta(a + b_3) \\ a \in [a_{\min}, a_{\max}], [b_1, b_2, b_3] \in [b_{\min}, b_{\max}] \end{gathered} \tag{18}$$

Gathering terms multiplied by the control, we can find the optimal control by taking the argmin of these terms.

$$\begin{gathered} a^* = \arg\min_a \{< \nabla G, f(x,a,b) >\} = \arg\min_a \{G_\theta * a\} \\ = a_{\min} \text{ if } G_\theta \geq 0,\ a_{\max} \text{ if } G_\theta \leq 0 \end{gathered} \tag{19}$$

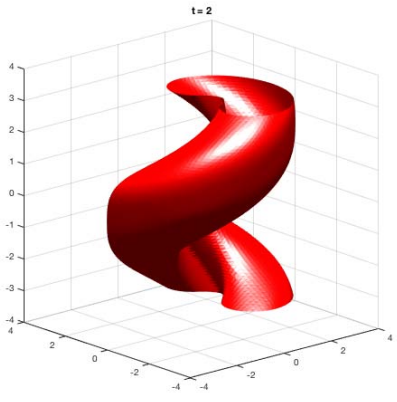


Fig. 12: Visualization that should appear when running *tutorial_test.m*, illustrating a backward reachable set (BRS) for a Dubins car.

We follow a similar procedure to find one of the optimal disturbances.

$$\begin{aligned} b_1^* &= \arg\max_{b_1} \{< \nabla G, f(x,a,b) >\} \\ &= \arg\max_{b_1} \{G_{p_x} * b_1\} \\ &= b_{1\max} \text{ if } G_{p_x} \geq 0,\ b_{1\min} \text{ if } G_{p_x} \leq 0 \end{aligned} \tag{20}$$

The optimal disturbances b_2^* and b_3^* can be similarly computed. These results are coded into *optCtrl.m* and *optDstb.m* for the Dubins car sub-class. For examples using more complicated dynamics, one can explore other sub-classes within *helperOC/dynSys*.

B. Reachability Analysis Setup

The example file we will use here to define the reachability analysis is *tutorial_test.m*, and is contained within the helperOC repository. This function consists of modifiable code to run several different forms of reachability analysis for a 3D Dubins car example. Try running this function to verify correct installation. A visualization of a spiral red set in 3D should appear, as shown in Fig. 12.

The comments at the top of the script explain how to modify the function to test different versions. In this section we will briefly go through the different blocks of the code in this file.

1) *Trajectory Computation?*
   This block is set to true when you want to test the results of the reachability analysis using a test trajectory. Note that this can only be used for backward reachable sets and tubes.
2) *Grid*
   In order to compute a reachable set numerically, the level set toolbox discretizes the state space and solves for the value function over a discrete grid.[4] This block defines the grid by setting the minimum and maximum states, along with the number of grid points in each dimension. Periodic dimensions should be noted (to account for periodic behavior), and the grid is created. Note that the grid bounds should be large enough to enclose the target and the reachable set or tube. Also note that a finer discretization will lead to more accurate results.
3) *Target Set*
   Here we define the target set of the system. As noted in Section II, this is either a subset of the state space we want the system to reach, or an unsafe set that we want the system to avoid. In this example we use the function *shapeCylinder.m* to create a target set that is circular in p_x, p_y space and encompasses all θ states. Functions for other shapes can be found in the *toolboxLS* user manual.
4) *Time Vector*
   In this block the initial and final times are set, as well as the time step desired. Note that for forward reachable sets the variable *tau* moves forward in time, and for backward reachable sets *tau* moves backward in time. See Section V-A for more details on FRSs and BRSs.
5) *Problem Parameters*
   Here the problem parameters for the dynamical system are introduced. These problem parameters are defined by the class of the dynamical system (in this case, a Dubins car). The control and (if applicable) disturbance modes (*uMode* and *dMode*) are also defined here. This refers to whether the control (or disturbance) is trying to maximize or minimize the value function (see Section V-C for details). Table I shows the modes needed depending on the reachability problem. The disturbance mode *dMode* is generally the opposite of *uMode* for the worst-case analysis. The table also differentiates between whether the control is trying to reach the target set (goal), or avoid the target set (avoid).

TABLE I: *uMode* Conditions

	uMode	
Target set	goal (larger set)	avoid (smaller set)
Forward	max	min
Backward	min	max

6) *Pack Problem Parameters*
   This block packs problem parameters into the variables needed for the reachability computation. The dynamical system is defined using the input parameters from the previous block and calling upon the appropriate dynamic system class that was created in Section X-A. The system, grid, *uMode* (and *dMode* if applicable), and accuracy level are set. The accuracy options are *low*, *medium*, *high*, and *veryHigh*. Note that higher accuracy results in a more accurate gradient calculation of the value function, but takes more time to compute

[4]This is the key cause of the curse of dimensionality in the BRS computation.

the value function.

7) *Obstacles*
Obstacles (or unsafe sets) should be defined here using the same format used for creating the target set. The obstacles should then be combined in a cell structure and set to *HJIextraArgs.obstacles*.
8) *Compute Value Function*
In this block we set the *HJIextraArgs* parameter to visualize the reachability analysis during computation. We then use the main function of *helperOC*, *HJIPDE_solve.m*, to perform the reachability analysis and to acquire the discrete form of the continuous value function in equation (6). Note that the function can solve for a reachable set by setting the *minWith* input to 'none', or a tube by setting the *minWith* input to 'zero'. The differences between sets and tubes are explained in Section V-B. More information on *HJIPDE_solve.m* and extra functionalities are in Section X-C.
9) *Compute Optimal Trajectory for Some Initial State*
If the Trajectory Computation block is set to true, this block computes and visualizes an optimal trajectory from a given initial state and the optimal controller derived from the value function, which is computed using equation (14) for a BRS, for example.

C. Using HJIPDE_Solve.m

The main function used by *helperOC* is *HJIPDE_Solve.m*, which can be found in *helperOC/valFuncs*. This function interfaces the tools developed in helperOC with the functions used in toolboxLS. The inputs are the initial values at each grid point (*data0*), the time vector (*tau*), the problem parameters for toolboxLS (*schemeData*), whether to compute a set or a tube (*minWith*), and any additional inputs desired (*extraArgs*). The outputs are the value function at each grid point at each time step (*data*), the time vector (*tau*), and any additional outputs desired (*extraOuts*).

The range of possibilities for the *extraArgs* input are described in comments at the beginning of the function. You can include obstacles, visualize the set over time, stop when the set reaches some initial state, save the data periodically, and more.

REFERENCES

[1] J. Sethian, "A fast marching level set method for monotonically advancing fronts," *National Academy of Sciences*, vol. 93, no. 4, pp. 1591–1595, 1996.

[2] S. Osher and R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*. Springer-Verlag, 2006.

[3] I. Mitchell, "Application of level set methods to control and reachability problems in continuous and hybrid systems," Ph.D. dissertation, Stanford University, 2002.

[4] ——, "A toolbox of level set methods," *Department of Computer Science, University of British Columbia, Vancouver, BC, Canada, http://www. cs. ubc. ca/˜ mitchell/ToolboxLS/toolboxLS.pdf, Tech. Rep. TR-2004-09*, 2004.

[5] A. Bayen, I. Mitchell, M. Osihi, and C. Tomlin, "Aircraft autolander safety analysis through optimal control-based reach set computation," *AIAA J. Guidance, Control, and Dynamics*, vol. 30, no. 1, pp. 68–77, 2007.

[6] J. Ding, J. Sprinkle, S. Sastry, and C. Tomlin, "Reachability calculations for automated aerial refueling," in *Proc. IEEE Conf. Decision and Control*, 2008.

[7] P. Bouffard, "On-board model predictive control of a quadrotor helicopter: Design, implementation, and experiments," Master's thesis, University of California, Berkeley, 2012.

[8] A. Aswani, H. Gonzalez, S. Sastry, and C. Tomlin, "Provably safe and robust learning-based model predictive control," *Automatica*, vol. 49, no. 5, pp. 1216–1226, 2013.

[9] H. Huang, J. Ding, W. Zhang, and C. Tomlin, "A differential game approach to planning in adversarial scenarios: A case study on capture-the-flag," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2011.

[10] M. Chen, S. Bansal, J. Fisac, and C. Tomlin, "Robust Sequential Path Planning Under Disturbances and Adversarial Intruder," *IEEE Trans. Control Syst. Technol.*, to appear.

[11] M. Chen, Q. Hu, C. Mackin, J. Fisac, and C. Tomlin, "Safe platooning of unmanned aerial vehicles via reachability," in *Proc. IEEE Conf. Decision and Control*, 2015.

[12] S. Herbert, M. Chen, S. Han, S. Bansal, J. Fisac, and C. Tomlin, "FaSTrack: a modular framework for fast and guaranteed safe motion planning," *Proc. IEEE Conf. Decision and Control*, 2017.

[13] M. Chen, S. Herbert, M. Vashishtha, S. Bansal, and C. Tomlin, "Decomposition of reachable sets and tubes for a class of nonlinear systems," *arXiv preprint arXiv:1611.00122*, 2016.

[14] M. Chen, S. Herbert, and C. Tomlin, "Fast reachable set approximations via state decoupling disturbances," in *Proc. IEEE Conf. Decision and Control*, 2016.

[15] J. Darbon and S. Osher, "Algorithms for overcoming the curse of dimensionality for certain Hamilton-Jacobi equations arising in control theory and elsewhere," *Research in the Math. Sciences*, vol. 3, no. 1, p. 19, 2016.

[16] Y. Chow, J. Darbon, S. Osher, and W. Yin, "Algorithm for overcoming the curse of dimensionality for time-dependent non-convex Hamilton-Jacobi equations arising from optimal control and differential games problems," *J. Scientific Computing*, pp. 1–27, 2016.

[17] C. Baier, J. Katoen, and K. Larsen, *Principles of Model Checking*. Cambridge, MA: MIT Press, 2008.

[18] C. Belta, B. Yordanov, and E. Gol, *Formal Methods for Discrete-Time Dynamical Systems*, ser. Studies in Systems, Decision and Control. Springer International Publishing, 2017, vol. 89.

[19] M. Reynolds, "Continuous temporal models," in *Proc. Australian Joint Conf. Artificial Intelligence*, 2001.

[20] G. Fainekos, A. Girard, H. Kress-Gazit, and G. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343–352, 2009.

[21] S. Jha, V. Raman, D. Sadigh, and S. Seshia, "Safe autonomy under perception uncertainty using chance-constrained temporal logic," *J. Automated Reasoning*, 2017.

[22] S. Coogan, M. Arcak, and C. Belta, "Formal methods for control of traffic flow: Automated control synthesis from finite-state transition models," *IEEE Control Systems*, vol. 37, no. 2, pp. 109–128, 2017.

[23] M. Bolton, E. Bass, and R. Siminiceanu, "Using formal verification to evaluate human-automation interaction: A review," *IEEE Trans. Syst. Man Cybern. A., Syst. Humans*, vol. 43, no. 3, pp. 488–503, 2013.

[24] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "SpaceEx: Scalable verification of hybrid systems," in *Proc. Int. Conf. Computer Aided Verification*, 2011.

[25] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Flow*: An analyzer for non-linear hybrid systems," in *Proc. Int. Conf. Computer Aided Verification*, 2013.

[26] M. Althoff, "An introduction to CORA 2015," in *Proc. ARCH@ CPSWeek*, 2015.

[27] P. Duggirala, S. Mitra, M. Viswanathan, and M. Potok, "C2E2: A verification tool for stateflow models," in *Proc. Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems*, 2015.

[28] C. Fan, B. Qi, S. Mitra, M. Viswanathan, and P. Duggirala, "Automatic reachability analysis for nonlinear hybrid models with C2E2," in *Proc. Int. Conf. Computer Aided Verification*, 2016.

[29] S. Kong, S. Gao, W. Chen, and E. Clarke, "dReach: δ-Reachability analysis for hybrid systems," in *Proc. Int. Conf. Tools and Algorithms*

for the Construction and Analysis of Systems, 2015. [Online]. Available: http://link.springer.com/10.1007/978-3-662-46681-0_15

[30] P. Duggirala, C. Fan, M. Potok, B. Qi, S. Mitra, M. Viswanathan, S. Bak, S. Bogomolov, T. Johnson, L. Nguyen *et al.*, "Tutorial: Software tools for hybrid systems verification, transformation, and synthesis: C2E2, HyST, and TuLiP," in *Proc. Conf. Control Applications*, 2016.

[31] P. Parrilo, "Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization," Ph.D. Dissertation, California Institute of Technology, 2000. [Online]. Available: http://resolver.caltech.edu/CaltechETD:etd-05062004-055516

[32] R. Tedrake, I. Manchester, M. Tobenkin, and J. Roberts, "LQR-trees: Feedback motion planning via sums-of-squares verification," *Int. J. Robotics Research*, vol. 29, no. 8, pp. 1038–1052, 2010.

[33] A. Barry, A. Majumdar, and R. Tedrake, "Safety verification of reactive controllers for UAV flight in cluttered environments using barrier certificates," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2012.

[34] A. Majumdar, A. Ahmadi, and R. Tedrake, "Control design along trajectories with sums of squares programming," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2013.

[35] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *Int. J. Robotics Research*, vol. 36, no. 8, pp. 947–982, 2017.

[36] E. Barron, "Differential games with maximum cost," *Nonlinear analysis: Theory, methods & applications*, vol. 14, no. 11, pp. 971–989, 1990.

[37] I. Mitchell, A. Bayen, and C. Tomlin, "A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games," *IEEE Trans. Autom. Control*, vol. 50, no. 7, pp. 947–957, 2005.

[38] O. Bokanowski and H. Zidani, "Minimal time problems with moving targets and obstacles," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 2589–2593, 2011.

[39] J. Fisac, M. Chen, C. Tomlin, and S. Sastry, "Reach-avoid problems with time-varying dynamics, targets and constraints," in *Proc. ACM Int. Conf. Hybrid Systems: Computation and Control*, 2015.

[40] A. Kurzhanski and P. Varaiya, "Ellipsoidal techniques for reachability analysis: internal approximation," *Systems & control letters*, vol. 41, no. 3, pp. 201–211, 2000.

[41] ——, "On ellipsoidal techniques for reachability analysis. Part II: Internal approximations box-valued constraints," *Optimization Methods and Software*, vol. 17, no. 2, pp. 207–237, 2002.

[42] J. Maidens, S. Kaynama, I. Mitchell, M. Oishi, and G. Dumont, "Lagrangian methods for approximating the viability kernel in high-dimensional systems," *Automatica*, vol. 49, no. 7, pp. 2017–2029, 2013.

[43] A. Majumdar, R. Vasudevan, M. Tobenkin, and R. Tedrake, *Convex optimization of nonlinear feedback controllers via occupation measures*, 2014, vol. 33, no. 9, pp. 1209–1230.

[44] T. Dreossi, T. Dang, and C. Piazza, "Parallelotope bundles for polynomial reachability," in *Proc. ACM Int. Conf. Hybrid Systems: Computation and Control*, 2016.

[45] D. Henrion and M. Korda, "Convex computation of the region of attraction of polynomial control systems," *IEEE Trans. Autom. Control*, vol. 59, no. 2, pp. 297–312, 2014.

[46] E. Coddington and N. Levinson, *Theory of ordinary differential equations*. Tata McGraw-Hill Education, 1955.

[47] L. Evans and P. Souganidis, "Differential games and representation formulas for solutions of Hamilton-Jacobi-Isaacs equations," WISCONSIN UNIV-MADISON MATHEMATICS RESEARCH CENTER, Tech. Rep., 1983.

[48] L. Evans, *Partial differential equations*. Amer. Math. Soc., 2010.

[49] I. Mitchell, "Comparing forward and backward reachability as tools for safety analysis," in *Proc. Int. Workshop on Hybrid Systems: Computation and Control*, 2007.

[50] J. Lygeros, "On reachability and minimum cost optimal control," *Automatica*, vol. 40, no. 6, pp. 917–927, 2004.

[51] K. Margellos and J. Lygeros, "HamiltonJacobi formulation for reachavoid differential games," *IEEE Trans. Autom. Control*, vol. 56, no. 8, pp. 1849–1861, 2011.

[52] M. Chen, S. Herbert, and C. Tomlin, "Exact and efficient Hamilton-Jacobi-based guaranteed safety analysis via system decomposition," *Proc. IEEE Int. Conf. Robotics and Automation*, 2017.

[53] I. Mitchell and C. Tomlin, "Overapproximating reachable sets by Hamilton-Jacobi projections," *J. Scientific Computing*, vol. 19, no. 1-3, pp. 323–346, 2003.

[54] S. Kaynama and M. Oishi, "Complexity reduction through a Schur-based decomposition for reachability analysis of linear time-invariant systems," *Int. J. Control*, vol. 84, no. 1, pp. 165–179, 2011.

[55] ——, "A modified Riccati transformation for decentralized computation of the viability kernel under LTI dynamics," *IEEE Trans. Autom. Control*, vol. 58, no. 11, pp. 2878–2892, 2013.

[56] J. Fisac, A. Akametalu, M. Zeilinger, S. Kaynama, J. Gillula, and C. Tomlin, "A general safety framework for learning-based control in uncertain robotic systems," *arXiv preprint arXiv:1705.01292*, 2017.

[57] A. Akametalu, J. Fisac, J. Gillula, S. Kaynama, M. Zeilinger, and C. Tomlin, "Reachability-based safe learning with Gaussian processes," in *Proc. IEEE Conf. Decision and Control*, 2014.

[58] K. N. Niarchos and J. Lygeros, "A neural approximation to continuous time reachability computations," in *Proc. IEEE Conf. Decision and Control*, 2006.

[59] B. Djeridane and J. Lygeros, "Neural approximation of PDE solutions: An application to reachability computations," in *Proc. IEEE Conf. Decision and Control*, 2006.

[60] V. R. Royo and C. Tomlin, "Recursive regression with neural networks: Approximating the HJI PDE solution," 2016. [Online]. Available: http://arxiv.org/abs/1611.02739

[61] F. Jiang, G. Chou, M. Chen, and C. Tomlin, "Using neural networks to compute approximate and guaranteed feasible Hamilton-Jacobi-Bellman PDE solutions," 2016. [Online]. Available: http://arxiv.org/abs/1611.03158

[62] T. Prevot, J. Rios, P. Kopardekar, J. Robinson III, M. Johnson, and J. Jung, "UAS Traffic Management (UTM) concept of operations to safely enable low altitude flight operations," in *Proc. AIAA Aviation Technol., Integration, and Operations Conf.*, 2016.

[63] M. Chen, Q. Hu, J. Fisac, K. Akametalu, C. Mackin, and C. Tomlin, "Reachability-based safety and goal satisfaction of unmanned aerial platoons on air highways," *AIAA J. Guidance, Control, and Dynamics*, pp. 1–14, 2017.

[64] M. Chen, S. Bansal, K. Tanabe, and C. Tomlin, "Provably safe and robust drone routing via sequential path planning: A case study in San Francisco and the Bay Area," 2017. [Online]. Available: http://arxiv.org/abs/1705.04585

[65] S. Bansal, M. Chen, J. Fisac, and C. Tomlin, "Safe sequential path planning of multi-vehicle systems under presence of disturbances and imperfect information," in *Proc. Amer. Control Conf.*, 2017.

[66] M. Chen, J. Shih, and C. Tomlin, "Multi-vehicle collision avoidance via Hamilton-Jacobi reachability and mixed integer programming," in *Proc. IEEE Conf. Decision and Control*, 2016.

[67] M. Chen, Z. Zhou, and C. Tomlin, "Multiplayer reach-avoid games via pairwise outcomes," *IEEE Trans. Autom. Control*, vol. 62, no. 3, pp. 1451–1457, 2017.