Demonstrating Numerical Convergence to the Analytic Solution of some Backwards Reachable Sets with Sharp Features*

Ian Mitchell

January 29, 2004

Abstract

We examine the convergence properties of a level set algorithm designed to track evolving interfaces; in particular, its convergence properies on a series of two and three dimensional backwards reachable sets whose flow fields involve kink formation (sharp features) and, in some cases, rarefaction fans introduced by input parameters in the dynamics. The chosen examples have analytic solutions to facilitate the convergence analysis. We describe the error analysis method, the formulation of reachability in terms of a Hamilton-Jacobi equation, and our implementation of the level set method in some detail. In addition to the convergence analysis presented here, these techniques and examples could be used to validate either other nonlinear reachability algorithms or other level set implementations.

keywords: nonlinear reachability, Hamilton-Jacobi equations, convergence analysis

*Research supported by ONR under MURI contract N00014-02-1-0720 and by DARPA under the Software Enabled Control Program (AFRL contract F33615-99-C-3014)

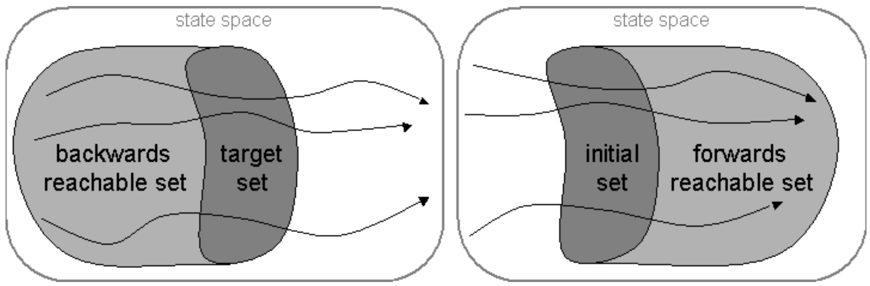


Figure 1: Backwards and forwards reachable sets. In this report, we deal only with the former.

1 Introduction

The *backwards reachable set* of a dynamic system is the set of states from which trajectories start that can reach some given target set of states moving under the system's prescribed dynamics; for example, see figure 1. Calculating reachable sets for continuous dynamic systems is a challenge because the sets involved contain an uncountable number of states. Furthermore, existing algorithms for reachability are complex enough that their implementations must be carefully validated against known examples.

In this report we present a validation by convergence analysis of one such algorithm—based on the Hamilton-Jacobi (HJ) partial differential equation (PDE)—using an example in two dimensions and three examples in three dimensions. The first three examples involve simple constant dynamics while the fourth is nonlinear. All the examples involve free input parameters that give rise to kinks and, in the last two cases, rarefactions in the underlying dynamics. *Kinks* are locations where characteristics (trajectories) of the underlying dynamics collide; topologically they will appear as sharp edges or points on the boundary of the reachable set, and from a PDE perspective the derivative of the implicit surface function (which represents the reachable set) is discontinuous and so this function cannot solve the HJ PDE in the classical sense. *Rarefactions* occur where characteristics are diverging. Both features can pose serious challenges to Lagrangian front evolution algorithms; for example, those that follow specific system trajectories. Level set algorithms are designed to overcome these problems using an Eulerian (fixed grid) computation, and in this report we experimentally examine their effectiveness. For more details on the connection between level sets and reachable sets, see [17, 15, 16].

The reader should note, however, that the tests included here could be used to assess the accuracy of any algorithm designed either to solve general Hamilton-Jacobi partial differential equations, or to solve reachability problems for systems with both control and disturbance inputs. To help with this process, two types of formulations are provided for each example: a reachability problem with nonlinear dynamics and a Hamilton-Jacobi formulation. We also briefly explain how to convert between the two.

The second section explains our error analysis technique. The third and fourth describe the theory and implementation respectively of a level set algorithm for backwards reachability. The remaining sections describe the examples and provide convergence results.

2 Error Analysis Method

The level set method [19] is used to compute the evolution of dynamic surfaces by solving an HJ PDE over some full dimensional subset of the state space near the evolving interface. The zero level set of the HJ PDE's solution is an implicit description of the dynamic surface's boundary. Therefore, to determine the effectiveness of the level set method in tracking a dynamic surface, we are not interested in how accurately the HJ PDE is solved, but rather in the accuracy of the location of the zero level set.

To measure how accurately a level set function represents a surface, we need to have some "correct" representation of that surface. Many representations are possible; for example, in much of the level set literature (such as [24, 21, 20]), error analysis proceeds by comparing the numerical HJ PDE solution to an analytically determined implicit surface function (often the initial conditions provide an appropriate analytic function if periodic dynamics are used). Unfortunately, comparing two implicit surface functions is a rather indirect way of analyzing the error in dynamic surface calculations, even if one of those functions is known analytically.

Instead, we have chosen to represent the analytic solution of our examples as marker particles lying on the surface of the reachable sets. Marker particles have a long history of use in surface representation problems (see, for example, [28, 22]), and their primary disadvantage—the difficulty of computing and maintaining connectivity information—is irrelevant when they are used for error analysis. We mention in passing that it is straightforward to determine analytic implicit surface functions for the reachable sets of the first three of our examples, but for the final example (collision avoidance) we know of no analytic solutions other than the marker particle solutions described in [13, 11].

Let us now turn to a mathematical description of our error analysis procedure for an implicit surface function $v(x) : \mathbb{R}^n \to \mathbb{R}$ which describes a surface $\mathcal{S} = \{x \in \mathbb{R}^n \mid v(x) = 0\}$. We assume that in the neighborhood of $\mathcal{S}$, $\|D_x v(x)\| > 0$ so that there are no flat regions of $v(x)$ near the surface; ideally, $|v(x)|$ should increase monotonically as we get further from the surface. Most level set algorithms advocate regular reinitialization to keep $\|D_x v(x)\| \approx 1$ [18, 24], and therefore the (ideal) assumption will hold.

The true surface is given as a collection of points $\mathcal{C} = \{y_i\}_{i=1}^M$ where each $y_i \in \mathbb{R}^n$. These points should be chosen to cover the surface evenly. To assess the ability of the implicit surface to resolve sharp edges and corners in the interface, we should make sure to include points in $\mathcal{C}$ that lie exactly along these features. While the dimensionality of these features will mean that the number of such points will be small compared with the total number of points M and therefore their contribution to the average error may be insignificant, they often identify the locations where the maximum error occurs.

If the implicit surface function $v(x)$ is correct, then $v(y_i) = 0$ for all i. To quantify the error, we simply evaluate $|v(y_i)|$, and collect statistics on its value over all the points in $\mathcal{C}$; in particular, the maximum, average and root mean square size of the values in the set $\{|v(y_i)|\}_{i=1}^M$. If $v(x)$ is only known numerically on a discrete grid, we interpolate the value $v(y_i)$. We use the same set $\mathcal{C}$ for every grid size on a particular example, and try to choose M large enough that even on the finest grid there are several points in $\mathcal{C}$ for every grid cell near the interface.

Before proceeding to the next section, we briefly comment on the potential error due to interpolation. In general, any numerically demonstrated convergence rate will be limited to the smaller of the orders of accuracy of the integration scheme and of the interpolation scheme. In the analysis that follows, we use Matlab's `interpn` command with a cubic interpolant, and hence might expect to be able to show up to third order accuracy. However, third order accuracy for this interpolation scheme theoretically requires that the function to be interpolated have continuous derivatives up to the third, and a bounded fourth derivative. Unfortunately, in the neighborhood of kinks, $v(x)$ will have discontinuities in its first derivative, and therefore does not meet this requirement. These discontinuities could cause large interpolation errors near the kinks. Backing off to linear interpolation would reduce the chances of such large errors near the kinks, but would be unable to show a convergence rate greater than first order. We have therefore chosen to stick with Matlab's cubic interpolation scheme so that we can at least demonstrate superlinear convergence. We are not sure whether the failure to show second order or higher convergence rates in many of the examples is due to interpolation or integration error, but we are investigating alternative interpolation schemes that might be able to distinguish the two error sources.

3 A Level Set Formulation for Backwards Reachable Sets

In this section we formally define the reachable set for a system and formulate a terminal value HJ PDE whose solution describes it. For more details, see [14, 17].

We model our system with the ordinary differential equation

$$\frac{dx}{dt} = \dot{x} = f(x, a, b), \tag{1}$$

where $x \in \mathbb{R}^n$ is our state, $a(\cdot)$ is the input for player I and $b(\cdot)$ is the input for player II.

Assumption 1. The input signals are drawn from the following sets

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

where $\mathcal{A} \subset \mathbb{R}^{n_a}$ and $\mathcal{B} \subset \mathbb{R}^{n_b}$ are compact and $t \in [-T, 0]$ for some $T > 0$. We will consider input signals which agree almost everywhere to be identical.

Assumption 2. The flow field $f : \mathbb{R}^n \times \mathcal{A} \times \mathcal{B} \to \mathbb{R}^n$ is uniformly continuous, bounded, and Lipschitz continuous in x for fixed a and b. Consequently, given a fixed $a(\cdot) \in \mathfrak{A}(t)$, $b(\cdot) \in \mathfrak{B}(t)$ and initial point, there exists a unique trajectory solving (1).

Assumption 3. The *target set* $\mathcal{G}_0 \subset \mathbb{R}^n$ for our reachability problem is closed and can be represented as the zero sublevel set of a bounded and Lipschitz continuous function $g : \mathbb{R}^n \to \mathbb{R}$

$$\mathcal{G}_0 = \{x \in \mathbb{R}^n \mid g(x) \leq 0\}. \tag{2}$$

We assume that player I will try to steer the system away from the target with her input $a(\cdot)$, and player II will try to steer the system towards the target with her input $b(\cdot)$. For readers who prefer a more intuitive understanding of the inputs, consider that in our examples the target set will represent the capture set in a pursuit-evasion game. Our control or evader will then be player I and the pursuer or adversarial disturbance will be player II.

In a general differential game setting, the information that each player knows about the other's decisions can play a major role in the game's outcome. To simplify the exposition in this report, we make an additional assumption about the dynamics.

Assumption 4. The order of play in the differential game makes no difference in its outcome. Mathematically,

$$\max_{a \in \mathcal{A}} \min_{b \in \mathcal{B}} f(x, a, b) = \min_{b \in \mathcal{B}} \max_{a \in \mathcal{A}} f(x, a, b) \tag{3}$$

for all $x \in \mathbb{R}^n$.

Under this assumption, we need not delve too deeply into information patterns; for readers interested in more details, see [17, 3, 2]. Important classes of systems that satisfy (3) are those whose dynamics take any of the forms

$$\begin{aligned}
\dot{x} &= f(x, a), \\
\dot{x} &= f(x, b), \\
\dot{x} &= f_1(x, a) + f_2(x, b).
\end{aligned}$$

The first two are not games at all, and the final case is called *separable*. The examples in sections 5 and 6 correspond to the first case, while those in sections 7 and 8 are separable.

Note that in our formulation of the problem, a trajectory starts at some initial time $t < 0$ and we would like to know if it has passed into or through the target set by time zero. We will sometimes want to discuss the length of time that a trajectory has had to evolve; we adopt the differential game notation $\tau = -t$ to denote this positive quantity.

To solve the backwards reachability problem, we want to determine the *backwards reachable set* $\mathcal{G}(\tau)$ for $\tau \in [0, T]$. Informally, $\mathcal{G}(\tau)$ is the set of states from which there exists input signals for player II that for all input signals of player I will generate trajectories which lead to the target set within time τ. Unfortunately, a formal definition of the backwards reachable set in a two player differential game setting requires introducing a formal concept of strategies, since both players should be allowed to respond to the evolution of the system. Such details are beyond the scope of this report, since they are tangential to the goal of analyzing the accuracy of numerical approximations of the reachable set. Readers interested in the details are referred to [14, 17].

The reachable set can be determined by solving for the viscosity solution of a time-dependent Hamilton-Jacobi equation.

Theorem 1. *Let $v : \mathbb{R}^n \times [-T, 0] \to \mathbb{R}$ be the viscosity solution of the terminal value HJ PDE*

$$\begin{aligned} D_t v(x,t) + \min[0, H(x, D_x v(x,t))] = 0, \qquad & \textit{for } t \in [-T, 0], x \in \mathbb{R}^n; \\ v(x, 0) = g(x), \quad & \textit{for } x \in \mathbb{R}^n; \end{aligned} \tag{4}$$

where

$$H(x,p) = \max_{a \in \mathcal{A}} \min_{b \in \mathcal{B}} p^T f(x,a,b). \tag{5}$$

The zero sublevel set of v describes $\mathcal{G}(\tau)$

$$\mathcal{G}(\tau) = \{x \in \mathbb{R}^n \mid v(x,t) \leq 0\}.$$

The significance of this theorem is that we can harness well developed numerical schemes from the level set literature to compute accurate approximations of $v(x,t)$, and therefore accurate approximations of $\mathcal{G}(\tau)$, for even complicated nonlinear dynamics. For a proof of this theorem, see [14]. A completely different proof of the single player version of this theorem was developed in [10]. We also note that this theorem works for systems which do not satisfy assumption 4, although the definition of reachable set must suitably reflect the information pattern in those games where the pattern matters.

In previous papers we have presented alternative HJ PDE formulations for computing the backwards reachable set. In [27], the Hamiltonian was restricted to negative values only within the target set; unfortunately, the resulting potential for discontinuities in the solution makes accurate numerical implementation difficult. In [16], minimization was performed as a separate, postprocessing step. While this formulation is more efficient, it is more difficult to reason about and may produce incorrect results when the Hamiltonian and/or target set are nonconvex. Consequently, we advocate using the formulation in Theorem 1 for determining reachable sets.

4 Implementing a Level Set Algorithm

Nonlinear PDEs such as the HJ PDE (4) exhibit a number of properties that make their solutions difficult to determine either analytically or numerically; for example, even with smooth initial conditions $g(x)$ and flow field $f(x, a, b)$, the solution of (4) can develop kinks—locations where the derivatives become discontinuous—in finite time. However, since HJ PDEs describe a number of important physical processes, techniques have been developed to find numeric approximations of their solutions. In this section, we review those techniques. Readers familiar with level set algorithms may wish to skip directly to the examples.

Because of the presence of kinks, (4) usually does not have a solution in the classical sense. Instead, we seek the unique, continuous viscosity solution as defined in [6]. A family of algorithms called *level set methods* have been designed specifically to compute approximations to the viscosity solution for time-dependent HJ PDEs with continuous initial conditions and Hamiltonians such as (4). In this section we examine the details of adapting level set methods to the approximation of reachable sets. We assume throughout that the human modeler provides a way of computing the optimization over inputs a and b necessary to compute $H(x, p)$ in (5), and therefore we concentrate on numerically determining the zero level set of the solution v to (4).

4.1 The Numerical Scheme

The goal of our implementation is to compute with as much accuracy as possible an implicit surface function for the boundary of the reachable set. The accuracy of the derivative approximations described below is measured in terms of the order of their local truncation errors: on a grid with spacing Δx, an order r method for approximating a function u with a numerically computed $\hat{u}$ has error $\|u - \hat{u}\| = \mathcal{O}(\Delta x^r)$. In general, we will call any scheme with order two or greater ($r \geq 2$) a *high order* scheme.

Because our state space is $\mathbb{R}^n$, we compute an approximation of the value of $v(x,t)$ at the nodes of a fixed Cartesian grid in $\mathbb{R}^n \times [T, 0]$. Within (4), there are three terms that must be evaluated: the spatial derivative $D_x v(x,t)$, the Hamiltonian $H(x,p)$ and the time derivative $D_t v(x,t)$. One of the appealing properties of level set methods is that we can separately choose techniques for approximating each of these terms at each node using values of v at the node and its neighbors.

Spatial Derivative: Traditional finite difference approximations of order r for the spatial derivative of a function represented on a grid assume that the function and at least its first $r-1$ derivatives are continuous. Clearly this property will not hold in the presence of the kinks in $v(x,t)$. Nevertheless, convergent numerical approximations of $D_x v(x,t)$ were developed shortly after viscosity solutions were first proposed [7]. In our code, we use either a basic first order accurate scheme [19, 24] or a weighted, essentially non-oscillatory fifth order accurate approximation [20, 18].

A key feature of all of these schemes is their use of directional approximations. Consider approximating $D_x v(x,t)$ for $x \in \mathbb{R}$ (so $n = 1$). At a grid point x_i, there exists a *left* approximation $D_x^- v$ and a *right* approximation $D_x^+ v$; the usual first order accurate version would be

$$\begin{aligned} D_x^- v(x_i, t) &= \frac{v(x_i, t) - v(x_{i-1}, t)}{x_i - x_{i-1}}, \\ D_x^+ v(x_i, t) &= \frac{v(x_{i+1}, t) - v(x_i, t)}{x_{i+1} - x_i}. \end{aligned}$$

Achieving higher order accuracy requires the use of values from more than a grid point's immediate neighbors and, as mentioned above, assumes continuity of higher derivatives. The assumption will fail near kinks, and as a result the solution will become oscillatory and unstable. *Essentially non-oscillatory* (ENO) schemes compute several different approximations to the left and right, and then choose to use only the least oscillatory. A *weighted essentially non-oscillatory* (WENO) scheme takes advantage of all the approximations in smooth regions of the solution to increase the order of accuracy, but reverts to ENO near kinks. Our fifth order accurate WENO scheme uses three neighbors on each side to compute a node's left and right approximations to $D_x v(x,t)$. Extension to multidimensional spaces ($n > 1$) is conceptually trivial, since the approximation of $D_x v(x,t)$ can be computed separately for each dimension.

It should be noted that none of these finite difference schemes is likely to achieve better than first order accuracy in the immediate vicinity of a kink, because the first derivative does not exist at such a point. The added complexity of the schemes adds accuracy only away from these points; a property which is sometimes called *high resolution* to distinguish it from true high order accuracy. In the examples that follow, we will see that high resolution methods like WENO are worth the added complexity because the fully first order accurate schemes show significant failures on reachable sets with sharp features.

Hamiltonian: In general, we use the well studied *Lax-Friedrichs* (LF) approximation

$$\hat{H}(x, p^+, p^-) \triangleq H\left(x, \tfrac{p^- + p^+}{2}\right) - \tfrac{1}{2}\alpha^T(p^+ - p^-), \tag{6}$$

where p^+ and p^- are respectively the right and left approximations of p and $H(x,p)$ is given by (5). The second term in this approximation is a high order numerical dissipation added to damp out spurious oscillations in the solution. The components of the vector $\alpha \in \mathbb{R}^n$ depend on the partial derivatives of H with respect to its second argument

$$\alpha_i = \max_{p \in \mathcal{I}} \left| \frac{\partial H}{\partial p_i} \right| \tag{7}$$

where $\mathcal{I}$ is a hypercube containing all the values that the vector p takes on over the computational domain (see [20] for details). We can understand this dissipative term as being analogous to the Laplacian term $\epsilon \Delta v$ in the vanishing viscosity version $D_t v + H(x, D_x v) = \epsilon \Delta v$ of the HJ PDE. Too much dissipation will excessively smooth the approximate solution (rounding off what should be sharp corners in the reachable set), while too little will lead to numerical instability. The amount chosen by (7) is sufficient to guarantee stability and experimentally appears not to be overly dissipative.

For some of the examples studied below, it is possible to determine from the dynamics $f(x, a, b)$ and the terminal conditions $g(x)$ optimal input feedback policies $a(x)\colon \mathbb{R}^n \to \mathcal{A}$ and $b(x)\colon \mathbb{R}^n \to \mathcal{B}$* before beginning computations, and hence derive a constant convective flow field $\dot{x} = f(x) = f(x, a(x), b(x))$. For purely convective systems, a less dissipative approximation of the Hamiltonian uses only upwind derivatives [18]

$$\hat{H}(x, p^+, p^-) \triangleq H(x, p^*), \tag{8}$$

where the upwind costate vector $p^* \in \mathbb{R}^n$ is defined componentwise

$$p_i^* = \begin{cases} p_i^+, & \text{if } f_i(x) < 0; \\ p_i^-, & \text{if } f_i(x) > 0; \\ 0, & \text{otherwise.} \end{cases}$$

Because (8) contains no dissipative term like the α term in (6), we are more likely to be able to resolve sharp features of reachable sets using the upwind Hamiltonian. Unfortunately, we can only use (8) when the upwind direction is well defined, as is the case in purely convective flows. We are currently investigating other numerical Hamiltonians—such as Local Lax-Friedrichs and Roe with entropy fix [20]—whose generality and dissipation fall somewhere between those of (6) and (8).

Time Derivative: We appeal to the method of lines to treat the time derivative of (4). From (6) or (8) we can determine $\hat{H}$ at any node, and so we can treat the value of v at that node as the solution to the *Ordinary Differential Equation* (ODE) $D_t v + \min[0, \hat{H}] = 0$. Among the many numerical ODE solvers that exist, the explict Runge-Kutta (RK) schemes are particularly easy to implement. Like any explicit solver for time-dependent PDEs, the timestep Δt that can be taken by our RK integrator is restricted by the Courant-Friedrichs-Lewy (CFL) condition to be some flow speed dependent multiple of the spatial grid size Δx. In fact, applying standard RK schemes to the solution of HJ PDEs will lead to instability unless Δt is proportional to Δx^2, a restriction that would greatly increase computational cost for a

*Such feedback policies cannot be determined in advance for most systems, because the optimal inputs in (5) depend on the costate p. In these isolated examples, the costate evolves in predictable ways from the terminal conditions and we can thus find feedback policies that satisfy the assumptions placed on the inputs.

fixed time interval $[-T, 0]$. Therefore, we use *Total Variation Diminishing* (TVD) RK schemes [25, 18], which will not introduce oscillations into the solution when Δt is proportional to Δx. We have implemented first (which is just forward Euler), second and third order accurate TVD RK schemes. Because of the CFL condition, the timestep is usually much smaller than the grid spacing; consequently, it is possible to use less accurate methods in time than in space without a noticable degradation in solution quality.

4.2 Practical Details

After the numerical scheme is chosen, a number of practical details must be determined in order to produce reasonable approximations to reachable sets.

Initial Conditions: We assume that the modeler can provide a signed distance function representation $g(x)$ for $\mathcal{G}_0$. For many basic geometric shapes, such a function can be constructed manually. A sphere of radius r is given by $g(x) = \|x\|_2 - r$. The halfspace defined by a plane with outward normal $\hat{n}$ passing through point q is $g(x) = \hat{n}^T(x - q)$. Signed distance functions for a torus (section 6.1) and a cylinder (section 8.1) are given in the examples below.

To progress from basic to constructive geometry, we use the negation, minimum and maximum operators to create approximate signed distance functions for complements, unions and intersections respectively. For example, if sets $\mathcal{G}_1$ and $\mathcal{G}_2$ are represented by signed distance functions $g_1(x)$ and $g_2(x)$ then

$$\begin{aligned}
\mathcal{G}_3 = \mathcal{G}_1^{\complement} &\text{ is represented by } g_3(x) = -g_1(x), \\
\mathcal{G}_4 = \mathcal{G}_1 \cup \mathcal{G}_2 &\text{ is represented by } g_4(x) = \min[g_1(x), g_2(x)], \\
\mathcal{G}_5 = \mathcal{G}_1 \cap \mathcal{G}_2 &\text{ is represented by } g_5(x) = \max[g_1(x), g_2(x)], \\
\mathcal{G}_6 = \mathcal{G}_1 \backslash \mathcal{G}_2 &\text{ is represented by } g_6(x) = \max[g_1(x), -g_2(x)].
\end{aligned}$$

While the resulting functions are only approximately signed distance, they can be turned into true signed distance functions by applying reinitialization to them (see below).

Boundary Conditions: The HJ PDE (4) that we are trying to solve is defined over all of $\mathbb{R}^n$, and hence has no physical boundary. Unfortunately, we can numerically approximate the solution only on a finite domain, so we must introduce boundaries and enforce some form of boundary conditions. For periodic dimensions we choose our computational domain to include one complete period and enforce periodic boundary conditions; for example, the relative heading x_3 in section 8.

For nonperiodic dimensions, our goal is to keep any artificially enforced boundary conditions from degrading the accuracy of our reachable set approximation. The first step in achieving this goal is to keep the reachable set's boundary away from the boundary of the computational domain. Unfortunately, unless the boundary conditions are chosen carefully, they can introduce oscillations that destroy accuracy everywhere. To avoid this problem, we implement linear extrapolation away from zero along the boundary of our computational domain [18]. The simple idea is to linearly extrapolate the values of the level set function at grid cells beyond the computational domain. However, we do not want this extrapolation to imply a nonexistant zero level set interface lying somewhere beyond the domain. Consequently, we make sure that the slope of the linear extrapolation always points away from the zero level set. Consider the case in one dimension where we wish to determine a value for a node x_0 lying outside the computational domain next to real nodes x_1 and x_2. The extrapolation is then

$$v(x_0) = v(x_1) + \operatorname{sign}(v(x_1))|v(x_1) - v(x_2)|.$$

The combination of signum and absolute value functions guarantees that if the computational domain's boundary is outside the reachable set, then the level set function will increase in value in directions leading outside the domain. The converse would be true if the domain's boundary was inside the reachable set, although this case does not occur in the examples below. In the case $v(x_1) > 0$, this choice is mathematically equivalent to

$$v(x_0) = \max(2v(x_1) - v(x_2), v(x_2)),$$

which gives some intuition about why this scheme does not introduce spurious oscillations (use min instead of max if $v(x_1) < 0$). Experimentally we have determined that this choice of boundary condition remains stable even on inflow boundaries, whereas regular linear extrapolation frequently does not.

Reinitialization: Level set authorities often discuss the advantages of having a signed distance function ($\|D_x v\| = 1$) instead of merely an implicit surface function representation of the interface being tracked [18, 24]. However, even if the modeler provides a signed distance function for the terminal conditions $g(x)$, evolution according to (4) can quickly distort the level set function by causing its gradient to become steep in some regions and shallow in others. When the gradient becomes very steep or very shallow, the quality of gradient approximations can degrade, which in turn reduces the accuracy of the level set scheme. In order to avoid this outcome, we can periodically halt the regular computation in our algorithm and reconstruct a proper signed distance respresentation of the level set function, a process called *reinitialization*.

When computing the evolution of an interface with a level set method, reinitialization is justified because we are only interested in the location of the zero level set of the level set function v. Therefore, we can modify its value away from this level set as much as necessary to ensure that $\|D_x v\| = 1$. The two most commonly quoted disadvantages of reinitialization are its added computational cost and the fact that in practice it will inevitably cause slight perturbations of the zero level set. An additional disadvantage arises in reachability applications if the value of the implicit surface function has meaning away from the zero level set. For example, in the soft walls project [9], we are designing controllers to keep systems from entering the backwards reachable set [4]. These controllers make use of gradient and value information from the true solution to (4); information that would be corrupted by reinitialization.

Consequently, the decision whether to reinitialize or not will depend on the level set algorithm used, the complexity of the system dynamics and the goal of the reachability analysis. For the examples discussed in the remainder of this report, we found that reinitialization was unnecessary when running our dense level set algorithm in Matlab. On the other hand, the localized algorithm discussed below and running in our C++ implementation requires regular reinitialization. Since it may be useful for some problems, we briefly discuss reinitialization procedures.

Because of its wide use in level set methods, several different techniques for reinitialization have been developed (see, for example, [5, 24, 18]). At various times, we have used two distinct methods. For low order accurate but fast reinitialization, we use fast marching methods (FMM) [29, 23]. These schemes are first order accurate, but need to visit each reinitialized grid point only once when building the signed distance function. However, starting an FMM sweep can be a challenge, because nodes adjacent to the interface must have known values (often requiring explicit construction of the interface).

The second reinitialization method executes a few discrete timesteps of a solver for the PDE [26, 8]

$$\begin{aligned} D_{\tilde{t}}\tilde{v}(x,\tilde{t}) &= \operatorname{sign}(\tilde{v}(x,\tilde{t}))(1 - \|D_x \tilde{v}(x,\tilde{t})\|), \\ \tilde{v}(x,0) &= v(x,t). \end{aligned} \tag{9}$$

Note that this PDE is run in an auxiliary timeframe $\tilde{t}$. If it were run to convergence, then $\|D_x \tilde{v}\| = 1$, but in practice we run one to ten discrete timesteps to some $\tilde{t}_f$, and then reset $v(x,t) = \tilde{v}(x,\tilde{t}_f)$ to get $\|D_x v\| \approx 1$. Analytically, $\operatorname{sign}(\tilde{v}) = 0$ on the zero level set of v, and so that level set should never move. In practice, we need to use a smoothed sign function, such as

$$\operatorname{sign}(v) = \frac{v}{\sqrt{v^2 + \Delta x^2}},$$

to avoid moving the zero level set too much. The advantages of this scheme are that it can be started directly from the current implicit surface function v, and that we can use high resolution techniques to approximate the gradient and time derivative and thereby achieve increased accuracy. A fast but accurate Gudonov solver for (9) is available [8, section A.3], so there is no need to use a dissipative Lax-Friedrichs approximation. The disadvantage of this scheme is its speed, which can be considerably slower than an FMM when high resolution approximations are employed.

Localizing Computation: The HJ PDE (4) describes the evolution of v in all of $\mathbb{R}^n$; however, in some cases we are only interested in its zero level set. Consequently, we can restrict our effort to grid nodes near the boundary between positive and negative values of v. In the level set literature this idea has been variously called *local level set* [21] or *narrowbanding* [5, 1]. We have implemented a new variant of this method in the C++ version of our code, and typically restrict our effort to within three to six nodes on each side of the interface.

Because the boundary of the reachable set is of one dimension less than the state space, considerable savings are available for two and three dimensional problems. If the number of nodes in each dimension is n (proportional to Δx^{-1}) and the dimension d, the total number of nodes is $\mathcal{O}(n^d)$; the CFL condition on timestep means that total computational cost for a fixed time interval $[T, 0]$ is $\mathcal{O}(n^{d+1})$. With local level sets, we reduce computational costs back down to $\mathcal{O}(n^d)$, and we have seen this cost behavior experimentally. The disadvantages to this scheme include the need for reinitialization (typically 25%–50% of execution time) and the lack of values for the level set function away from the interface; values which could be useful in some applications.

4.3 Example Run Details

The computations performed to calculate the reachable sets in the examples below share some common features. The "high order" algorithm uses the WENO spatial derivative approximation and third order TVD RK in time, while the "low order" algorithm uses first order spatial and time approximations. When choosing timestep size, we conservatively took a CFL number of 0.5, meaning that we took steps half as large as the CFL restriction suggests would be stable. The results section of each example specifies whether the Lax-Friedrichs (6) or upwinded (8) numerical Hamiltonian was used. The initial conditions in every example can be determined analytically. The computational domain is chosen to contain the final backwards reachable set (as given by the analytically determined $\mathcal{C}$) plus some buffer space, and linear extrapolation away from zero is used to enforce boundary conditions in every nonperiodic dimension. Since the time to convergence of the reachable set $\tau_{\max}$ is known for every example, we solve the HJ PDE out to some small multiple (typically 1.1) of $\tau_{\max}$ rather than rely on a numerical convergence test which might produce different stopping times for different grid resolutions. We solve the PDE over the entire computational domain at each timestep, and perform no reinitializations during the run.

The previous comments apply to our Matlab implementation, which was used for all of the examples except where noted. Our C++ implementation currently features a WENO spatial derivative approximation, second order accurate TVD RK, a CFL number of 0.9, localized computation and extensive reinitialization after every timestep. We are in the process of rebuilding this version to increase its flexibility.

5 The Ice-Cream Cone in Two Dimensions

We start with a two dimensional example featuring a single kink in order to demonstrate the technique in a simple setting.

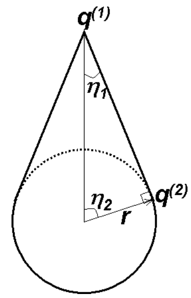


Figure 2: The ice-cream cone. The dashed portion of the circle is that part of the target which is not on the boundary of the reachable set

5.1 The Problem

The dynamics are

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = f(x,a) = \begin{bmatrix} a \\ -\gamma \end{bmatrix},$$

where $\gamma \geq 0$ is a constant and scalar input $a \in \mathcal{A} = [-\alpha, +\alpha]$ is used to steer the system away from the target set

$$\mathcal{G}_0 = \{x \in \mathbb{R}^2 \mid \|x\|_2 \leq r\},$$

which is a circle of radius r centered at the origin. Converting into Hamilton-Jacobi form, the terminal condition function is

$$g(x) = \sqrt{x_1^2 + x_2^2} - r,$$

and the Hamiltonian is

$$\begin{aligned} H(x,p) &= \max_{a \in \mathcal{A}} \left[p^T f(x,a) \right], \\ &= \max_{a \in [-\alpha,+\alpha]} \left[p_1 a + p_2(-\gamma) \right], \\ &= \alpha |p_1| - \gamma p_2. \end{aligned} \tag{10}$$

The resulting reachable set, shown in figure 2 is an inverted ice-cream cone, with the target set at the bottom and an inverted cone on the top. Defining

η_1, η_2, $q^{(1)}$ and $q^{(2)}$ as in figure 2, we can trigonometrically determine

$$\tan \eta_1 = \frac{\alpha}{\gamma}, \qquad \eta_2 = \frac{\pi}{2} - \eta_1,$$
$$q^{(1)} = \begin{bmatrix} 0 \\ \frac{r}{\sin \eta_1} \end{bmatrix} = \frac{r\sqrt{\alpha^2 + \gamma^2}}{\alpha} \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$
$$q^{(2)} = \begin{bmatrix} r \sin(\pi/2 - \eta_2) \\ r \cos(\pi/2 - \eta_2) \end{bmatrix} = \frac{r}{\sqrt{\alpha^2 + \gamma^2}} \begin{bmatrix} \alpha \\ \gamma \end{bmatrix},$$

From r, $q^{(1)}$, $q^{(2)}$ and the point symmetric to q_2 on the other side of the circle, it is straightforward to generate a collection of points lying on the surface of the reachable set. It is also easy to show that the reachable set converges in the time it takes a trajectory to go from a point at the top of the target circle to the tip of the cone q_1. Letting $\tau_{\max}$ be the time to convergence

$$\tau_{\max} = q_2^{(1)} - r = r\left(\sqrt{1 + \frac{\gamma^2}{\alpha^2}} - 1\right). \tag{11}$$

For these dynamics and this particular terminal condition, we can generate an equivalent constant flow field. Let $\dot{x}_i = f_i(x)$ where

$$f_1(x) = \begin{cases} +\alpha, & x_1 > 0 \text{ and } x_2 > q_2^{(2)}; \\ -\alpha, & x_1 < 0 \text{ and } x_2 > q_2^{(2)}; \\ 0, & x_1 = 0 \text{ or } x_2 \leq q_2^{(2)}. \end{cases} \tag{12}$$
$$f_2(x) = \begin{cases} -1, & x_2 > q_2^{(2)}; \\ 0, & x_2 \leq q_2^{(2)}. \end{cases}$$

The problem as described contains an obvious directional bias, since the kink in the horizontal direction lines up with the x_2 axis. This alignment leads to a suspicious level of accuracy for the algorithm, especially in maximum error. To make the results more general, we solve the system in a coordinate frame $z \in \mathbb{R}^2$ rotated counterclockwise by angle μ

$$z = R(\mu)x,$$
$$R(\mu) = \begin{bmatrix} \cos\mu & \sin\mu \\ -\sin\mu & \cos\mu \end{bmatrix}.$$

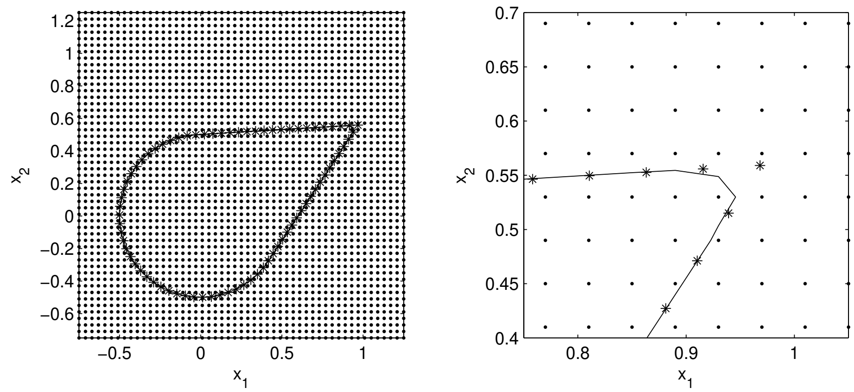


Figure 3: Level set solution (solid line) and $M = 76$ points in $\mathcal{C}$ (stars). The level set was calculated on the coarsest grid $N = 50$ (shown as gray dots). Left: entire grid. Right: zoomed view of kink point in reachable set, showing the location of greatest error.

5.2 The Results

We solve the ice-cream cone example in rotated coordinates for $\mu = -\pi/3$ on an uniform Euclidean grid covering the domain $[-0.75, +1.25]^2$. Figure 3 shows the results on the coarsest grid in comparison with a very sparse $\mathcal{C}$. The error analyses below are performed with a $\mathcal{C}$ of size $M = 4030$ (corresponding to an approximate point spacing of 0.001 along the boundary of the reachable set).

Because it is easier to code and quicker to run, we calculate the backwards reachable set using an appropriately rotated version of the constant flow field (12) rather than the equivalent Hamiltonian (10). In later versions of this report, we intend to analyze the effect of this choice.

Figure 4 shows the results for grid sizes ranging from $N = 50$ to $N = 800$. Both the low order accurate and high order accurate schemes were tested. To get a qualitative feel for convergence rate, lines equivalent to first and second order accuracy are shown. Since only the slopes of these lines matter, they were arbitrarily aligned vertically with the average error of the high order scheme on the coarsest grid.

From this plot we can see that for this example the algorithm is somewhere between first and second order accurate in average and root mean square

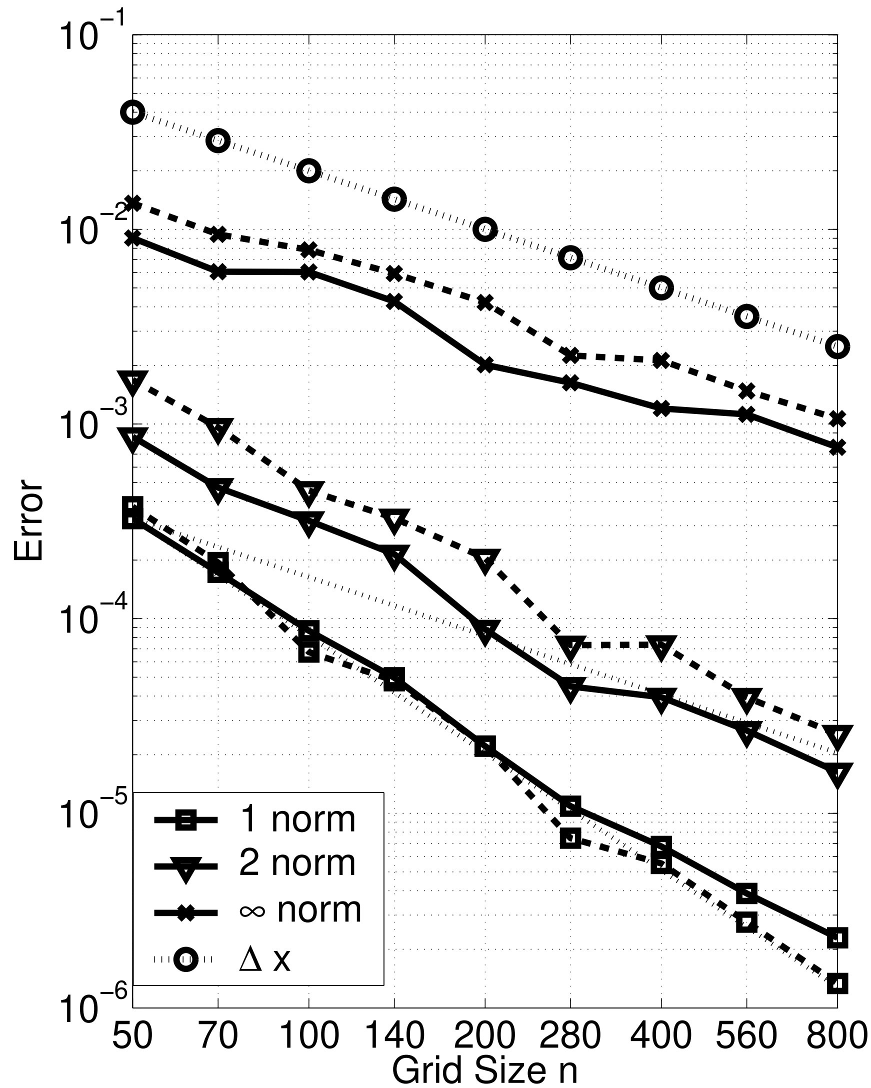


Figure 4: Experimental convergence rates for the two dimensional ice-cream cone example. Average (□), root mean square (▽) and maximum (×) errors for the high order accurate algorithm (solid lines) and low order accurate algorithm (dashed lines) are shown. For comparison the grid cell spacing Δx (○) is included, as are lines equivalent to first order (upper dotted line) and second order (lower dotted line) convergence rates.

norm	type	scale	Grid Size N 50	100	200	400	800
1	error	10^{-4}	3.267	0.867	0.221	0.067	0.023
	ratio	1		3.77	3.92	3.28	2.94
	/ Δx	10^{-2}	0.817	0.433	0.221	0.135	0.092
2	error	10^{-4}	8.602	3.184	0.882	0.395	0.163
	ratio	1		2.70	3.61	2.23	2.43
	/ Δx	10^{-2}	2.151	1.592	0.882	0.789	0.651
∞	error	10^{-3}	9.034	6.043	2.014	1.201	0.760
	ratio	1		1.50	3.00	1.68	1.58
	/ Δx	1	0.226	0.302	0.201	0.240	0.304

Table 1: Quantitative error for various grid sizes on the ice-cream cone example, using the high order accurate algorithm. The "ratio" rows show the convergence rate (first order would have ratio 2, second order would have ratio 4). The "/Δx" rows show error size as a fraction of grid cell size.

error, and nearly first order accurate in maximum error. Equally important, the maximum error is significantly smaller than a grid cell, and the average error is less than 1% of a grid cell. There is little difference between the high order and low order algorithms, so in this case the much faster low order algorithm would be sufficient. We shall see that this fact is not true for more challenging examples in three dimensions. Table 1 shows the error numbers quantitatively for the high order accurate algorithm on selected grid sizes. Notice that the grid size doubles between columns, so a first order accurate algorithm would have ratio 2 and a second order accurate algorithm would have ratio 4.

All error interpolation in this report is performed using Matlab's `interpn` command, generally with the `cubic` option. Figure 5 compares the difference in error for the high order accurate level set scheme using either the `cubic` option (solid lines) or the `linear` option (dashed lines). If it is operating properly, the cubic scheme should show a significantly different convergence rate that the linear scheme. The fact that the two schemes show nearly the same convergence rate leads us to believe that the cubic interpolation scheme is having trouble with the kinked solution. Although we omit this interpolation comparison for the remaining examples, the results are similar. Hence, we are seeking a more suitable interpolation scheme in the hopes of showing higher order accuracy.

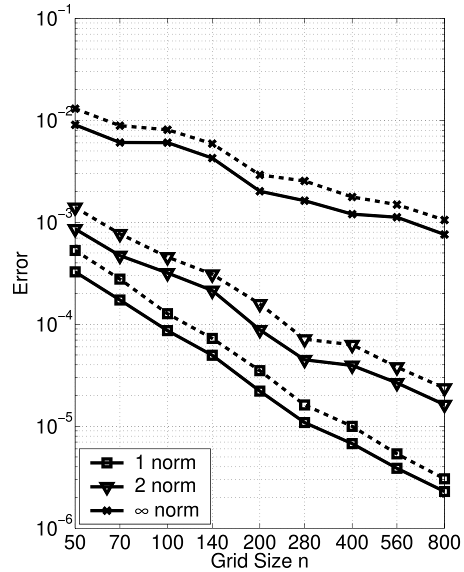


Figure 5: Comparing error for cubic error interpolation (solid lines) and linear error interpolation (dashed lines) for the ice-cream cone example.

6 The Ice-Cream Cone Torus in Three Dimensions

We extend the two dimensional ice-cream cone into three dimensions. The boundary of the two dimensional version already features curves, straight lines, and sharp points to challenge the accuracy of a level set implementation. In our extension to three dimensions we add a hole and a two dimensional kink curve (which is not grid aligned). Although this example does not include a sharp point, the next does.

6.1 The Problem

To define the dynamics, we transform the coordinates $x \in \mathbb{R}^3$ into cylindrical coordinates

$$z = \begin{bmatrix} z_\rho \\ z_\theta \\ z_3 \end{bmatrix} = \begin{bmatrix} \sqrt(x_1^2 + x_2^2) \\ \tan^{-1}(x_2/x_1) \\ x_3 \end{bmatrix} \in \mathbb{R}^+ \times [0, 2\pi[\times \mathbb{R} \triangleq \mathbb{S}.$$

We then apply a cylindrically symmetric version of the ice-cream cone example

$$\frac{d}{dt}\begin{bmatrix} z_\rho \\ z_\theta \\ z_3 \end{bmatrix} = f(z,a) = \begin{bmatrix} a \\ 0 \\ -\gamma \end{bmatrix},$$

where $\gamma \geq 0$ is a constant and scalar input $a \in \mathcal{A} = [-\alpha, +\alpha]$ is used to steer the system away from the target set

$$\mathcal{G}_0 = \left\{ z \in \mathbb{S} \;\middle|\; \sqrt{(z_\rho - d)^2 + z_3^2} \leq r \right\},$$

which is a torus with radius d and cross sectional radius r, centered at the origin and lying in the $x_3 = 0$ plane. Converting into Hamilton-Jacobi form, the terminal condition function is

$$g(z) = \sqrt{(z_\rho - d)^2 + z_3^2} - r,$$

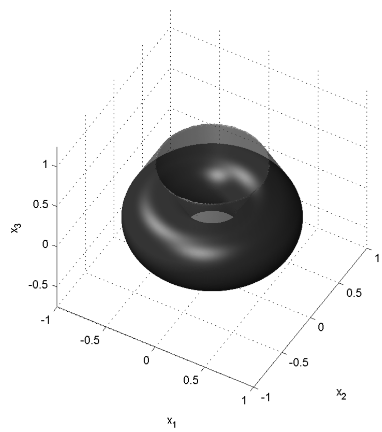
and the Hamiltonian is

$$\begin{aligned} H(z,p) &= \max_{a \in \mathcal{A}} \left[p^T f(z,a) \right], \\ &= \max_{z \in [-\alpha, +\alpha]} \left[p_\rho a + p_3(-\gamma) \right], \\ &= \alpha |p_\rho| - \gamma p_3. \end{aligned}$$

To visualize the resulting reachable set, shift the two dimensional ice-cream cone d units to the right horizontally, and then rotate it about the vertical axis to form a three dimensional shape. The point kink at the top of the two dimensional reachable set becomes a circular kink in the three dimensional set.

In a manner similar to the previous example, we can generate an equivalent constant flow field for these dynamics and this particular terminal condition.

$$\begin{aligned} f_\rho &= \begin{cases} +\alpha, & z_\rho > d \text{ and } z_3 > q_3^{(2)}; \\ -\alpha, & z_\rho < d \text{ and } z_3 > q_3^{(2)}; \\ 0, & z_\rho = d \text{ or } z_3 \leq q_3^{(2)}. \end{cases} \\ f_\theta &= 0. \\ f_3 &= \begin{cases} -1, & z_3 > q_3^{(2)}; \\ 0, & z_3 \leq q_3^{(2)}. \end{cases} \end{aligned} \tag{13}$$



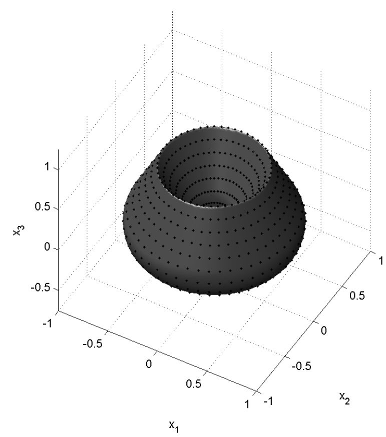


Figure 6: Left: target set (solid donut) and backwards reachable set (transparent) for the ice-cream cone torus in three dimensions on the $N = 159$ grid. Right: backwards reachable set (solid) and $M = 1020$ points in $\mathcal{C}$ (dots).

where the point $q_3^{(2)}$ is the vertical coordinate of the point $q^{(2)}$ defined as in the previous section (adjusted for the cylindrical coordinate system)

$$q^{(2)} = \begin{bmatrix} d \\ 0 \\ 0 \end{bmatrix} + \frac{r}{\sqrt{\alpha^2 + \gamma^2}} \begin{bmatrix} \alpha \\ 0 \\ \gamma \end{bmatrix}.$$

The time of convergence $\tau_{\max}$ is exactly that given in (11).

Because the kink of this example is already a circle and hence cannot be aligned with the computational Euclidean grid, we do not bother to rotate the dynamics.

6.2 The Results

We solve the ice-cream torus example on a uniform Euclidean grid covering the domain $[-1, +1]^2 \times [-0.75, +1.25]$. Figure 6 shows the target set and backwards reachable set on the finest grid, in comparison with a very sparse $\mathcal{C}$. The error analyses below are performed with a $\mathcal{C}$ of size $M = 482880$ (corresponding to an approximate point spacing of 0.005 along the boundary

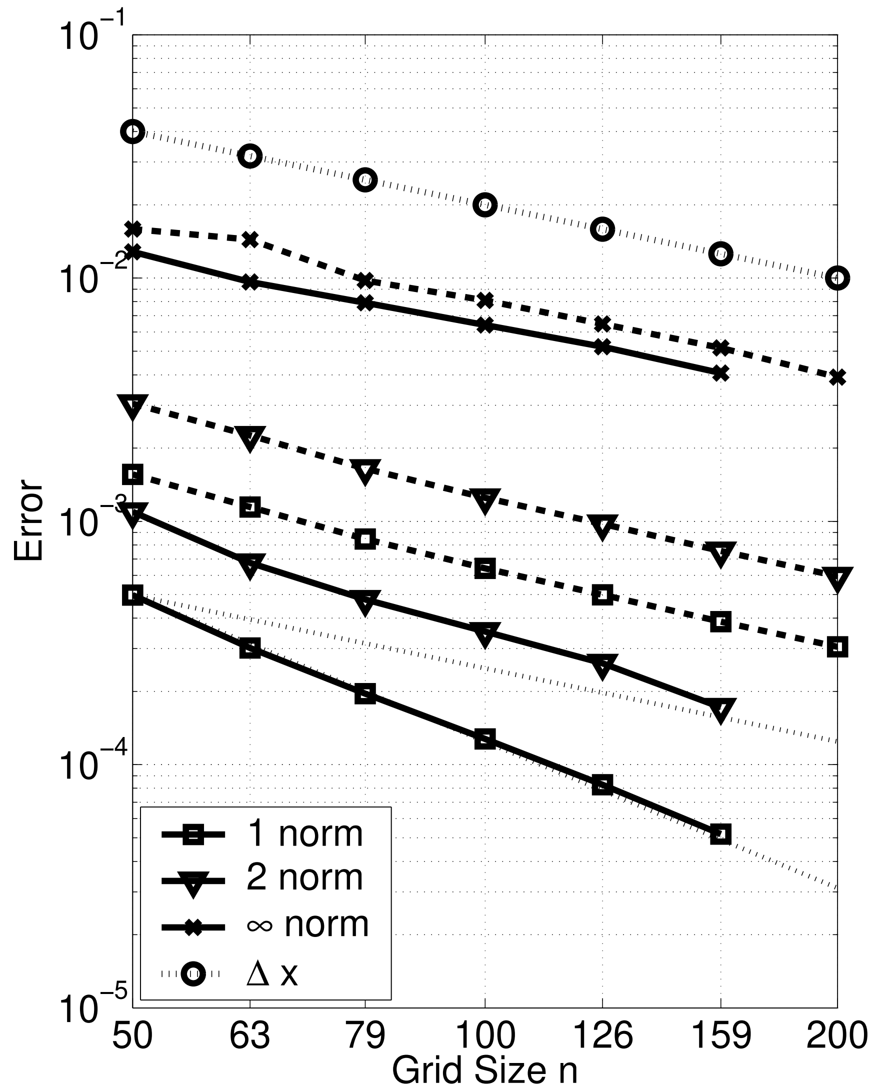


Figure 7: Experimental convergence rates for the three dimensional ice-cream torus example. Average (□), root mean square (▽) and maximum (×) errors for the high order accurate algorithm (solid lines) and low order accurate algorithm (dashed lines) are shown. For comparison the grid cell spacing Δx (○) is included, as are lines equivalent to first order (upper dotted line) and second order (lower dotted line) convergence rates.

norm	type	scale	Grid Size N					
			50	63	79	100	126	159
1	error	10^{-4}	4.975	3.016	1.957	1.276	0.827	0.518
	ratio	1		1.65	1.54	1.53	1.54	1.60
	/ Δx	10^{-2}	1.244	0.950	0.773	0.638	0.521	0.412
2	error	10^{-4}	10.940	6.750	4.774	3.525	2.608	1.718
	ratio	1		1.62	1.41	1.35	1.35	1.52
	/ Δx	10^{-2}	2.735	2.126	1.886	1.763	1.643	1.366
∞	error	10^{-3}	12.818	9.634	7.908	6.410	5.220	4.070
	ratio	1		1.33	1.22	1.23	1.23	1.28
	/ Δx	1	0.320	0.303	0.312	0.321	0.329	0.324

Table 2: Quantitative error for various grid sizes on the ice-cream torus example, using the high order accurate algorithm. The "ratio" rows show the convergence rate (first order would have ratio 1.26, second order would have ratio 1.59). The "$/\Delta x$" rows show error size as a fraction of grid cell size.

of the reachable set). We calculate the backwards reachable set using the constant flow field (13).

Figure 7 shows the results for grid sizes ranging from $N = 50$ to $N = 200$. The physical memory of our machine (1 MB) was unable to solve this problem with the high order accurate scheme for grids larger than 159^3, and with the low order accurate scheme for grids larger than 200^3.

From this plot we can see that for this example the algorithm is second order accurate in average error, somewhere between first and second order accurate in root mean square error, and slightly better than first order accurate in maximum error. The maximum error is significantly smaller than a grid cell, and the average error is 1% or less of a grid cell. The main difference between the results of this example and those of the last is the noticeably degraded accuracy of the low order scheme in average and root mean square error; however, the maximum errors of the two schemes are similar. Since the low order scheme is much faster, the choice of scheme would depend on which type of error is more significant to the application. Table 2 shows the error numbers quantitatively for the high order accurate algorithm.

7 The Axe

The previous two examples contained kink formation, but no rarefaction regions. In this section, we consider a backwards reachable set which looks like an axe head† and is formed by a flow field that is converging in one dimension while diverging in another. The resulting reachable set contains not only a kink line (the axe's edge), but also two sharp points (the top and bottom of the axe's edge) to confound our interface algorithm.

7.1 The Problem

The dynamics are straightforward

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = f(x,a) = \begin{bmatrix} -1 \\ a \\ b \end{bmatrix},$$

where scalar input $a \in \mathcal{A} = [-\alpha, +\alpha]$ is used to steer the system away from the target set $\mathcal{G}_0$ and scalar input $b \in \mathcal{B} = [-\beta, +\beta]$ is used to steer the system towards $\mathcal{G}_0$. In order to make it easy to compute points on the surface of the reachable set, $\mathcal{G}_0$ is chosen to be a six sided polyhedron with a trapezoidal projection into the x_1–x_3 plane and a rectangular projection into the x_1–x_2 plane. The eight corners of $\mathcal{G}_0$ are given by

$$q^{(1)} = \begin{bmatrix} \underline{x}_1 \\ \underline{x}_2 \\ \underline{x}_3 \end{bmatrix} \quad q^{(2)} = \begin{bmatrix} \underline{x}_1 \\ \overline{x}_2 \\ \underline{x}_3 \end{bmatrix} \quad q^{(3)} = \begin{bmatrix} \underline{x}_1 \\ \overline{x}_2 \\ \overline{x}_3 \end{bmatrix} \quad q^{(4)} = \begin{bmatrix} \underline{x}_1 \\ \underline{x}_2 \\ \overline{x}_3 \end{bmatrix}$$

$$q^{(5)} = \begin{bmatrix} \overline{x}_1 \\ \underline{x}_2 \\ \underline{\underline{x}}_3 \end{bmatrix} \quad q^{(6)} = \begin{bmatrix} \overline{x}_1 \\ \overline{x}_2 \\ \underline{\underline{x}}_3 \end{bmatrix} \quad q^{(7)} = \begin{bmatrix} \overline{x}_1 \\ \overline{x}_2 \\ \overline{\overline{x}}_3 \end{bmatrix} \quad q^{(8)} = \begin{bmatrix} \overline{x}_1 \\ \underline{x}_2 \\ \overline{\overline{x}}_3 \end{bmatrix}$$

where $\underline{x}_i$ and $\overline{x}_i$ are problem parameters specifying the size of the target set and

$$\underline{\underline{x}}_3 = \underline{x}_3 - \beta(\overline{x}_1 - \underline{x}_1) \qquad \overline{\overline{x}}_3 = \overline{x}_3 + \beta(\overline{x}_1 - \underline{x}_2)$$

are chosen to make $\mathcal{C}$ easy to find. From $\{q^{(i)}\}_{i=1}^{8}$, it is easy to find planes describing the six faces of $\mathcal{G}_0$. Since $\mathcal{G}_0$ is the intersection of the six halfspaces defined by these planes, the terminal condition function $g(x)$ can be

†In the spirit of the now century old Stanford-Berkeley cross bay rivalry.

constructed by taking the maximum of the (simple to evaluate) implicit surface functions for each of these halfspaces. We omit the algebraic details of this process. The Hamiltonian is

$$
\begin{aligned}
H(x,p) &= \max_{a\in\mathcal{A}} \min_{b\in\mathcal{B}} \left[p^T f(x,a) \right], \\
&= \max_{a\in[-\alpha,+\alpha]} \min_{b\in[-\beta,+\beta]} \left[p_1(-1) + p_2 a + p_3 b \right], \\
&= -p_1 + \alpha|p_2| - \beta|p_3|.
\end{aligned} \tag{14}
$$

There is a constant flow field equivalent to this target set and Hamiltonian combination, but we do not derive its rather complicated form here.

To visualize the resulting reachable set, take the target polygon $\mathcal{G}_0$ and attach a four sided polygon to its largest rectangular face (the face whose corners are $\{q^{(i)}\}_{i=5}^{8}$). The initial polygon forms the back side of the axe, while the second polygon is the blade and edge. This second polygon has two trapezoidal faces which form the sides of the blade and meet at the axe's edge. The other two faces of this polygon are triangular, with bases running along $\mathcal{G}_0$ and the tips forming the sharp points at the top and bottom of the axe's edge. These triangular faces have the same normal as the sloped faces of $\mathcal{G}_0$.

In forming $\mathcal{C}$, it is easy to find points on the five faces of $\mathcal{G}_0$ that are also part of the reachable set. To find points on the blade portion of the reachable set, it suffices to know $\{q^{(i)}\}_{i=5}^{8}$ and the additional two points at the top and bottom of the axe's edge

$$
q^{(9)} = \begin{bmatrix} \overline{x}_1 + \frac{1}{2\alpha}(\overline{x}_2 - \underline{x}_2) \\ \frac{1}{2}(\overline{x}_2 + \underline{x}_2) \\ \underline{\underline{x}}_3 - \frac{\beta}{2\alpha}(\overline{x}_2 - \underline{x}_2) \end{bmatrix} \qquad q^{(10)} = \begin{bmatrix} \overline{x}_1 + \frac{1}{2\alpha}(\overline{x}_2 - \underline{x}_2) \\ \frac{1}{2}(\overline{x}_2 + \underline{x}_2) \\ \overline{\overline{x}}_3 + \frac{\beta}{2\alpha}(\overline{x}_2 - \underline{x}_2) \end{bmatrix}
$$

It is also straightforward to determine that the convergence time $\tau_{\max} = \frac{1}{2\alpha}(\overline{x}_2 - \underline{x}_2)$.

The backwards reachable set as described above contains many axis aligned features. To make our convergence analysis more general, we solve the problem in the rotated coordinate frame $z = R_2(\mu_2)R_3(\mu_3)x \in \mathbb{R}^3$, where $R_i(\mu_i)$ rotates by angle μ_i counterclockwise about the i^{th} coordinate axis with the linear transforms

$$
R_2(\mu_2) = \begin{bmatrix} \cos\mu_2 & 0 & \sin\mu_2 \\ 0 & 1 & 0 \\ -\sin\mu_2 & 0 & \cos\mu_2 \end{bmatrix}, \qquad R_3(\mu_3) = \begin{bmatrix} \cos\mu_3 & \sin\mu_3 & 0 \\ -\sin\mu_3 & \cos\mu_3 & 0 \\ 0 & 0 & 1 \end{bmatrix}.
$$

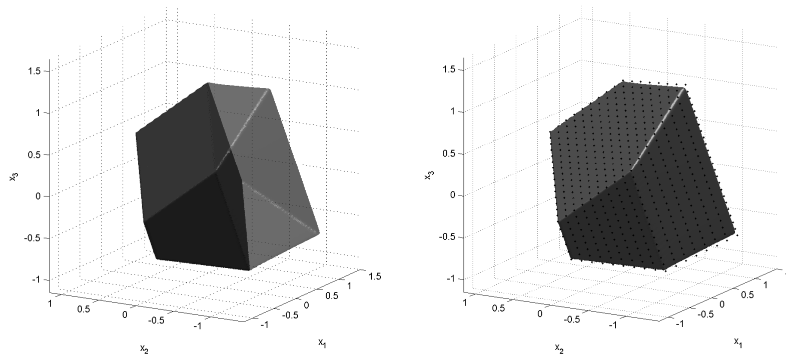


Figure 8: Left: target set (solid polygon) and backwards reachable set (transparent) for the axe example on the $N = 159$ grid. Right: backwards reachable set (solid) and $M = 912$ points in $\mathcal{C}$ (dots).

7.2 The Results

We solve the axe example on a Euclidean grid covering the domain

$$[-1.20, +1.50] \times [-1.45, +1.15] \times [-1.15, +1.65].$$

For grid size N, there are $\lceil (2.7/2.8)(N+1) \rceil$ grid points in the z_1 dimension, $\lceil (2.6/2.8)(N+1) \rceil$ grid points in the z_2 dimension, and $(N+1)$ grid points in the z_3 dimension (to get approximately equal Δz in all three dimensions). The parameters for the example were chosen as

$$\underline{x} = \begin{bmatrix} -0.5 \\ -0.8 \\ -0.3 \end{bmatrix} \quad \alpha = 1.0 \quad \rho_2 = +\frac{\pi}{6}$$

$$\overline{x} = \begin{bmatrix} +0.5 \\ +0.8 \\ +0.3 \end{bmatrix} \quad \beta = 0.5 \quad \rho_3 = -\frac{\pi}{3}$$

Figure 8 shows the target set and backwards reachable set on the finest grid, in comparison with a very sparse $\mathcal{C}$. The error analyses below are performed with a $\mathcal{C}$ of size $M = 430046$ (corresponding to an approximate point spacing of 0.005 along the boundary of the reachable set). We calculate the backwards reachable set from the general Hamiltonian (14).

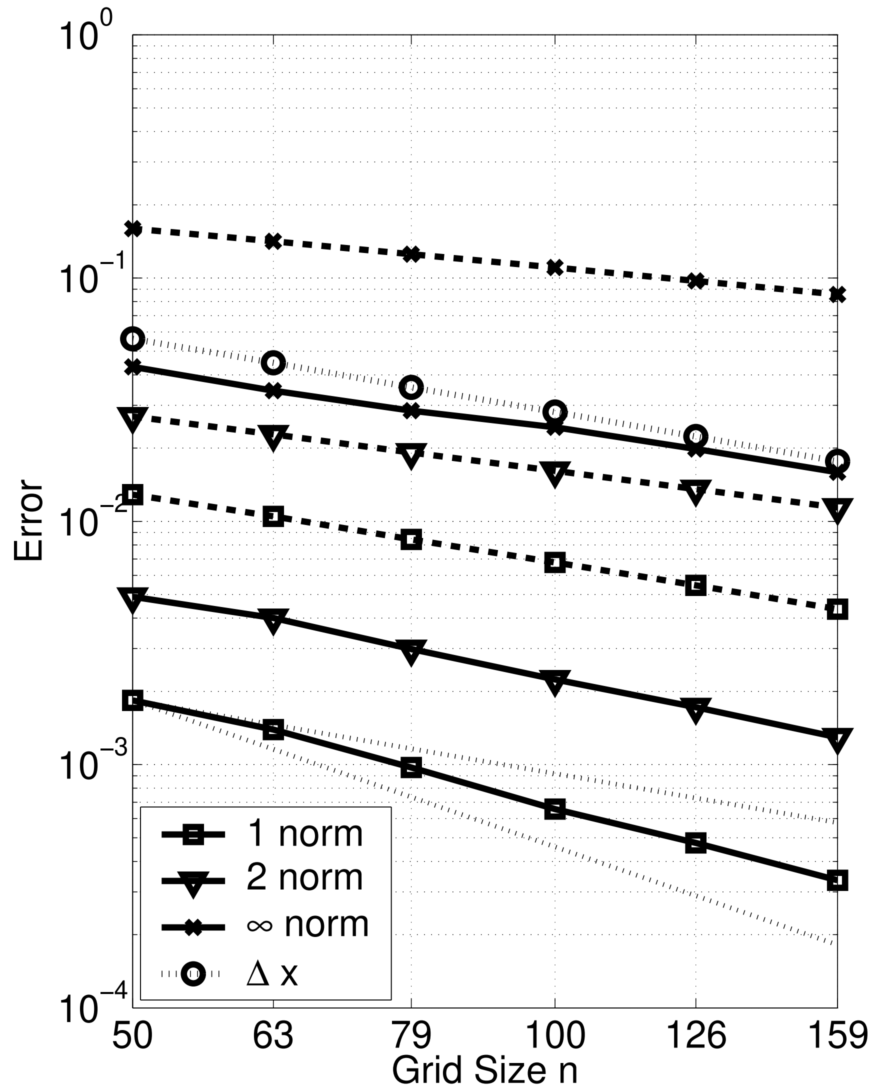


Figure 9: Experimental convergence rates for the axe example. Average (□), root mean square (▽) and maximum (×) errors for the high order accurate algorithm (solid lines) and low order accurate algorithm (dashed lines) are shown. For comparison the grid cell spacing Δx (○) is included, as are lines equivalent to first order (upper dotted line) and second order (lower dotted line) convergence rates.

norm	type	scale	Grid Size N 50	63	79	100	126	159
1	error	10^{-3}	1.835	1.394	0.973	0.657	0.477	0.335
	ratio	1		1.32	1.43	1.48	1.38	1.42
	/ Δx	10^{-2}	3.263	3.109	2.739	2.335	2.137	1.897
2	error	10^{-3}	4.896	4.005	2.984	2.237	1.721	1.295
	ratio	1		1.22	1.34	1.33	1.30	1.33
	/ Δx	10^{-2}	8.704	8.935	8.398	7.955	7.711	7.336
∞	error	10^{-2}	4.314	3.442	2.850	2.435	1.983	1.590
	ratio	1		1.25	1.21	1.17	1.23	1.25
	/ Δx	1	0.767	0.768	0.802	0.866	0.889	0.901

Table 3: Quantitative error for various grid sizes on the axe example, using the high order accurate algorithm. The “ratio” rows show the convergence rate (first order would have ratio 1.26, second order would have ratio 1.59). The “$/\Delta x$” rows show error size as a fraction of grid cell size.

Figure 9 shows the results for grid sizes ranging from $N = 50$ to $N = 159$. The physical memory of our machine was unable to solve this problem for grids larger than $N = 159$ using either scheme.

From this plot we can see that for this example the high order accurate algorithm is somewhere between first and second order accurate in average and root mean square error, and slightly worse than first order accurate in maximum error. The maximum error is smaller than a grid cell, and the average error is 4% or less of a grid cell. Table 3 shows the error numbers quantitatively for the high order accurate algorithm.

The low order accurate algorithm fares much worse, achieving at most first order accuracy and giving errors that are many times those produced by the high order accurate method. The major problem is that the low order scheme cannot resolve the sharp edges and especially the sharp points of this reachable set, as shown in figure 10. Only if rough results were sought could the low order accurate algorithm be justified for computing sets with such sharp features.

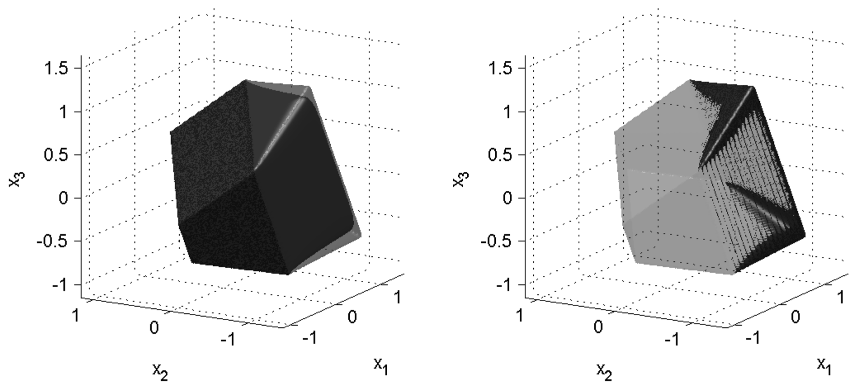


Figure 10: Comparing the low order accurate solution to the high order accurate solution on the $N = 159$ grid. Notice that the error is concentrated around the sharp edge and points of the axe. Right: low order accurate solution (solid) and high order accurate solution (transparent). Left: Difference between the low and high order accurate solutions (solid) and high order accurate solution (transparent).

8 A Collision Avoidance Example

This example features complex three dimensional dynamics with kink formation and rarefaction fronts. From a control perspective, we wish to determine reachability for a three dimensional kinematic model of two adversarial vehicles: the *pursuer* wants to get within a certain distance of the *evader*. In the dynamic game literature this problem is called *the game of two identical cars* [11], and the reachable set corresponds to the set in which the pursuer can capture the evader. Our previous publications [27, 16, 15] have called this problem the *three dimensional aircraft collision avoidance example*.

8.1 The Problem

We model our two vehicles with a commonly used, very simple kinematic system. The state z of each vehicle is represented by a location in the plane

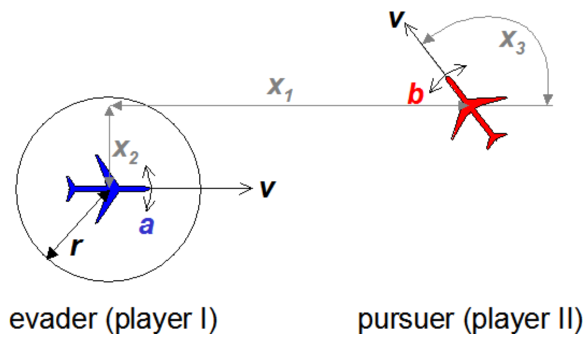


Figure 11: Relative coordinate system for collision example.

and a heading, so $z \in \mathbb{R}^2 \times [0, 2\pi[$. The evolution of these states is governed by the vehicle's forward velocity v and rotational velocity ω

$$\frac{d}{dt}\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} v\cos\psi \\ v\sin\psi \\ \omega \end{bmatrix}.$$

For the purposes of this example, we fix the linear velocities of the vehicles and use the angular velocities as the inputs, so v will be a constant while a and b will correspond to ω.

We say that a *collision* has occurred if the two vehicles come within distance r of one another. Our goal is to determine the set of states from which the pursuer can cause a collision to occur. Translating into reachability terms, $\mathcal{G}_0$ is the set of all states where the two vehicles are within r units of one another, the evader is player I (input a), the pursuer is player II (input b), and the capture set is $\mathcal{G}(\tau)$. Because $\mathcal{G}_0$ depends only on the relative positions of the vehicles, we can simplify the system down to three dimensions by working in relative coordinates $x \in \mathbb{R}^2 \times [0, 2\pi[$. As shown in figure 11, we fix the evader at the origin and facing along the positive x_1 axis. Then the pursuer's relative location and heading are described by the flow field

$$\dot{x} = \frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -v_a + v_b\cos x_3 + a x_2 \\ v_a \sin x_3 - a x_1 \\ b - a \end{bmatrix} = f(x, a, b).$$

We can find the resulting Hamiltonian

$$
\begin{aligned}
H(x,p) &= \max_{a\in\mathcal{A}} \min_{b\in\mathcal{B}} \left[p^T f(x,a,b)\right], \\
&= \max_{a\in[-\alpha,+\alpha]} \min_{b\in[-\beta,+\beta]} \begin{bmatrix} -p_1 v_a + p_1 v_b \cos x_3 + p_2 v_a \sin x_3 \\ \qquad + a(p_1 x_2 - p_2 x_1 - p_3) + b p_3 \end{bmatrix}, \\
&= \begin{pmatrix} -p_1 v_a + p_1 v_b \cos x_3 + p_2 v_a \sin x_3 \\ \qquad + \alpha|p_1 x_2 - p_2 x_1 - p_3| - \beta|p_3| \end{pmatrix},
\end{aligned}
\tag{15}
$$

where the bounds on inputs are $a \in [-\alpha, +\alpha]$ and $b \in [-\beta, +\beta]$.

Since a collision can occur at any relative heading, the target set $\mathcal{G}_0$ depends only on x_1 and x_2 and includes any state within distance r of the planar origin

$$\mathcal{G}_0 = \{x \in R^3 | x_1^2 + x_2^2 < r^2\},$$

which can be converted into signed distance function

$$g(x) = \sqrt{x_1^2 + x_2^2} - r, \tag{16}$$

for our HJ PDE's terminal conditions.

The reachability algorithm we have presented can solve this problem for any choices of parameters. However, if we focus on the case in which the two vehicles' control authority and speed are identical, we can analytically determine points $\mathcal{C}$ on the surface of the backwards reachable set. Our method is based on work in [11], which solves this example with the pursuer at the origin. We have recently recreated these results in Matlab, and then modified them to solve the game with the evader at the origin (it turns out that the two cases are not symmetric). For reasons of space, we will not reiterate the complex procedure involved with computing these points, but rather refer the reader to [13] for details or to [12] for the Matlab code that was used to generate $\mathcal{C}$. In what follows, we choose the parameters

$$
\begin{aligned}
r &= 5, \\
v_a = v_b &= 5, \\
\alpha = \beta &= +1.
\end{aligned}
\tag{17}
$$

During the computation of $\mathcal{C}$ from these parameters, it can be determined that the convergence time $\tau_{\max} \approx 2.6$.

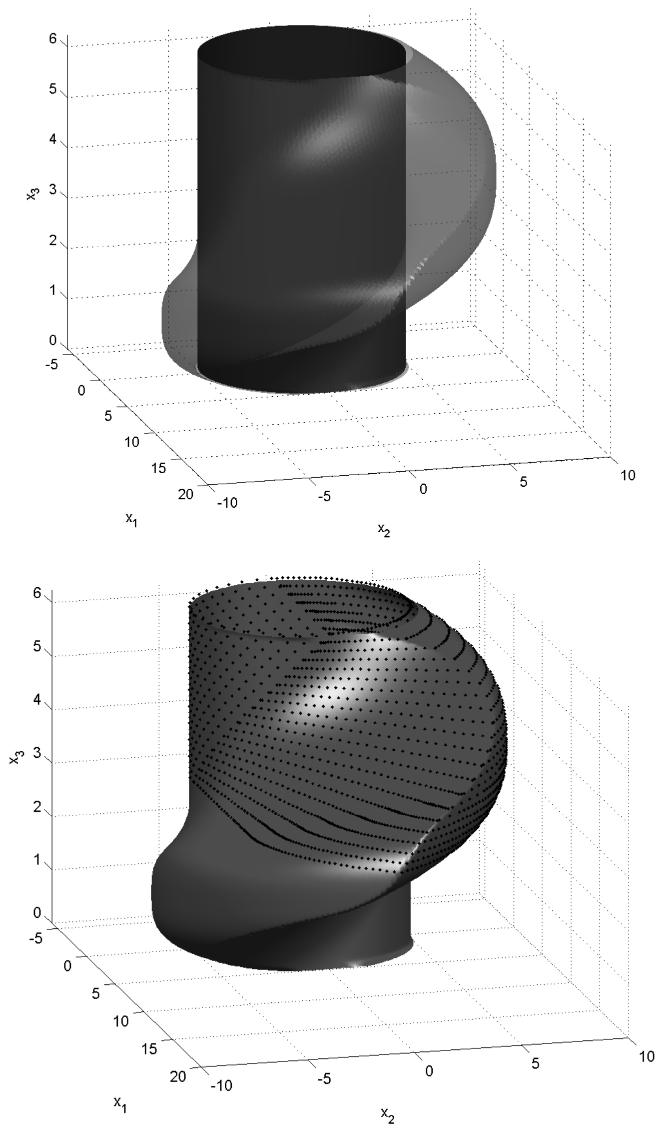


Figure 12: Top: target set (solid cylinder) and backwards reachable set (transparent) for the collision avoidance example in three dimensions on the $N = 126$ grid. Bottom: backwards reachable set (solid) and $M = 2612$ points in $\mathcal{C}$ (dots). Because the backwards reachable set is vertically symmetric about $x_3 = \pi$, we analyze the accuracy of only the top half of the set.

8.2 The Results

We solve the collision avoidance example on a Euclidean grid covering the domain $[-6, +20] \times [-10, +10] \times [0, 2\pi[$, with periodic boundary conditions in the third dimension. For grid size N, there are $N + 1$ grid points in the x_1 dimension, $\lceil (20/26)(N + 1) \rceil$ grid points in the x_2 dimension (to get approximately equal Δx in these two dimensions) and N grid points in the periodic x_3 dimension. Figure 12 shows the target set and backwards reachable set, in comparison with a very sparse $\mathcal{C}$. Animations showing this set growing and spinning are available at [12]. The error analyses below are performed with a $\mathcal{C}$ of size $M = 238597$, and the computation is done directly from the Hamiltonian (15).

Figure 13 shows the results for grid sizes ranging from $N = 50$ to $N = 200$ (at the time of writing, the high order accurate computation had not yet been performed on the finest grid). From this plot we can see that for this example the high order accurate algorithm is nearly second order accurate in average error, first order accurate in root mean square error, and slightly worse than first order accurate in maximum error. The maximum error is approximately a grid cell, and the average error is less than 3% of a grid cell, as shown in table 4.

The low order accurate algorithm fares much worse; it is at best first order accurate in all norms, the average and root mean square error are nearly a grid cell in size, and the maximum error is more than two grid cells. Figure 14 shows clearly that the difference between the low and high order accurate schemes lies in how precisely they capture the sharp features on the front of the bulge in the reachable set.

We have also implemented this example in our C++ version of the level set code. The same parameters (17) and point cloud $\mathcal{C}$ were used for these C++ runs, although they used a slightly different domain and hence grid cell spacing. Figure 15 shows the results for grid sizes ranging from $N = 50$ to $N = 200$. The convergence rate conclusions are similar.

9 Conclusions and Future Work

In this report we have demonstrated that level set algorithms can be used to compute accurate approximations to backwards reachable sets in two and

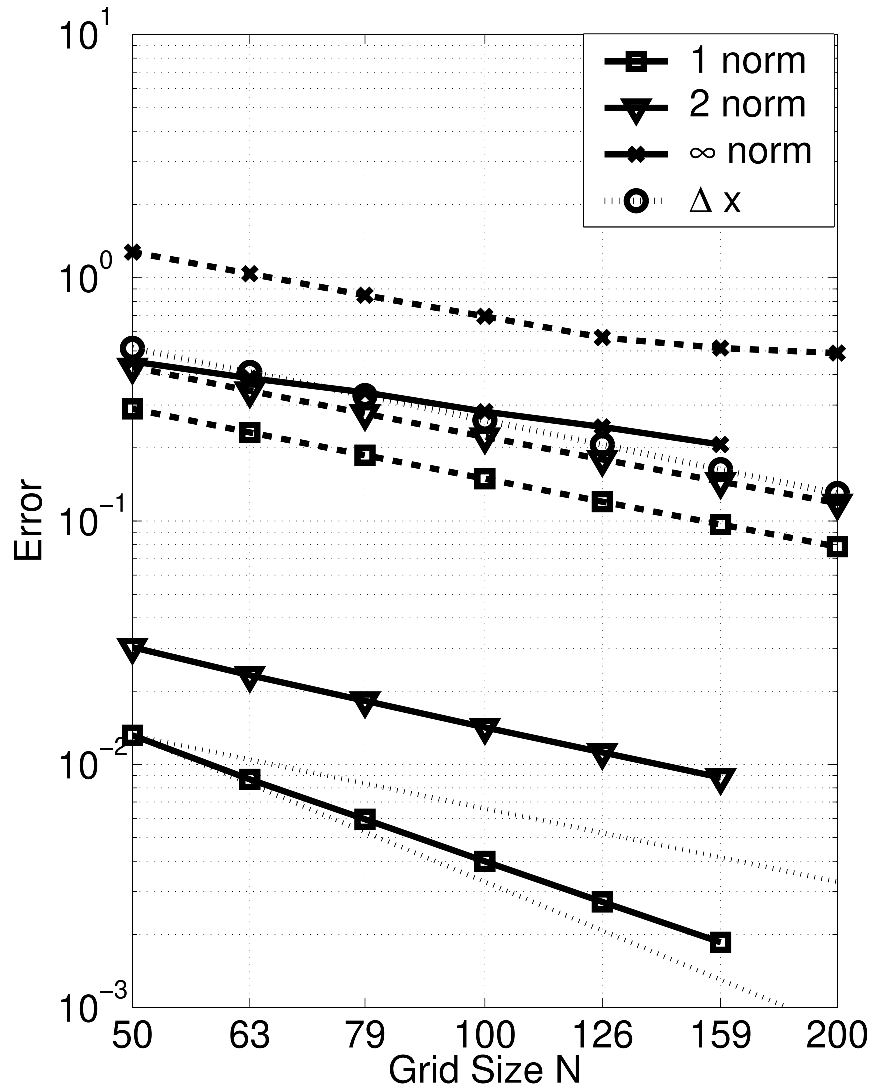


Figure 13: Experimental convergence rates for the collision avoidance example. Average ($\Box$), root mean square ($\triangledown$) and maximum ($\times$) errors for the high order accurate algorithm (solid lines) and low order accurate algorithm (dashed lines) are shown. For comparison the grid cell spacing Δx ($\circ$) is included, as are lines equivalent to first order (upper dotted line) and second order (lower dotted line) convergence rates.

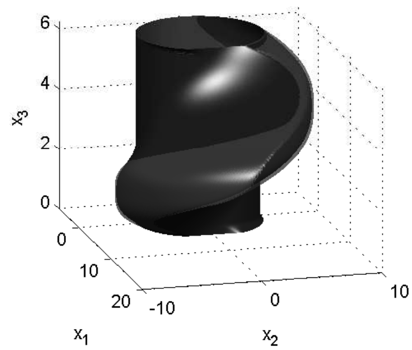


Figure 14: Comparing the low order accurate solution to the high order accurate solution on the $N = 126$ grid. Notice that the error is concentrated around the edge of the bulge in the reachable set. Right: low order accurate solution (solid) and high order accurate solution (transparent). Left: Difference between the low and high order accurate solutions (solid) and high order accurate solution (transparent).

norm	type	scale	Grid Size N 50	63	79	100	126	159
1	error	10^{-2}	1.290	0.867	0.585	0.400	0.272	0.186
	ratio	1		1.49	1.48	1.46	1.47	1.47
	/ Δx	10^{-2}	2.515	2.125	1.783	1.538	1.320	1.14
2	error	10^{-2}	2.984	2.323	1.804	1.420	1.120	0.883
	ratio	1		1.28	1.29	1.27	1.27	1.27
	/ Δx	10^{-2}	5.819	5.692	5.503	5.465	5.434	5.432
∞	error	1	0.450	0.387	0.339	0.282	0.244	0.206
	ratio	1		1.16	1.14	1.20	1.15	1.18
	/ Δx	1	0.878	0.947	1.033	1.084	1.183	1.270

Table 4: Quantitative error for various grid sizes on the collision avoidance example, using the high order accurate algorithm. The "ratio" rows show the convergence rate (first order would have ratio 1.26, second order would have ratio 1.59). The "/Δx" rows show error size as a fraction of grid cell size.

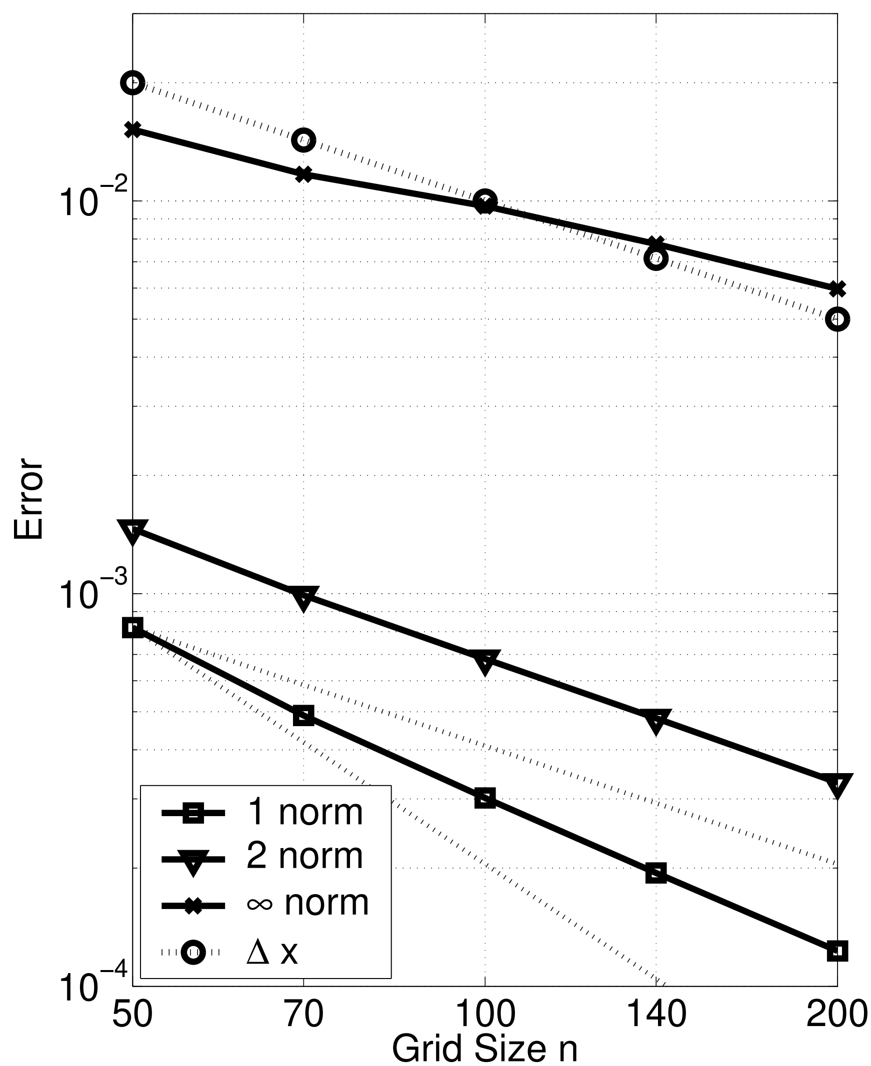


Figure 15: Experimental convergence rates for the collision avoidance example using the C++ implementation. Average (□), root mean square (▽) and maximum (×) errors for only the high order accurate algorithm (solid lines) is shown. For comparison the grid cell spacing Δx (∘) is included, as are lines equivalent to first order (upper dotted line) and second order (lower dotted line) convergence rates.

three dimensions for systems with nonlinear dynamics and multiple inputs. Even for reachable sets displaying sharp features (edges and corners), the algorithms were capable of significantly subgrid accuracy on average, and near grid level accuracy in worst case. An algorithm displaying subgrid accuracy can use a considerably coarser grid to achieve a fixed level of accuracy than would be needed by a grid level accurate algorithm. Because the cost of computation grows so fast as the grid is refined, even a grid twice as coarse can be computed eight to sixteen times faster.

The error analysis technique used here is general, in the sense that it can evaluate the accuracy of any algorithms designed to track interfaces featuring kinks or to solve Hamilton-Jacobi PDEs.

In addition to generalizing our C++ implementation and constructing user interfaces for both the C++ and Matlab versions, we intend to investigate adaptive mesh refinement and particle level set methods to improve accuracy, and to examine the effectiveness of these techniques in higher dimensions.

Acknowledgements: We would like to thank Professors Ronald Fedkiw and Stanley Osher for extensive discussions about the details of numerical schemes for solving the HJ PDE, and Professors Claire Tomlin and Shankar Sastry for their support during the construction of the software that was used.

References

[1] D. Adalsteinsson and J. A. Sethian. A fast level set method for propagating interfaces. *Journal of Computational Physics*, 118:269–277, 1995.

[2] M. Bardi. Some applications of viscosity solutions to optimal control and differential games. In I. Capuzzo-Dolcetta and P. L. Lions, editors, *Viscosity Solutions and Applications*, number 1660 in Lecture Notes in Mathematics, pages 44–97. Springer, 1995.

[3] M. Bardi and I. Capuzzo-Dolcetta. *Optimal Control and Viscosity Solutions of Hamilton-Jacobi-Bellman equations*. Birkhäuser, Boston, 1997.

[4] J. Adam Cataldo. Control algorithms for soft walls. Technical Report UCB/ERL M03/42, Department of Electrical Engineering and Computer Science, University of California, Berkeley, October 2003.

[5] D. Chopp. Computing minimal surfaces via level set curvature flow. *Journal of Computational Physics*, 106:77–91, 1993.

[6] M. G. Crandall, L. C. Evans, and P.-L. Lions. Some properties of viscosity solutions of Hamilton-Jacobi equations. *Transactions of the American Mathematical Society*, 282(2):487–502, 1984.

[7] M. G. Crandall and P.-L. Lions. Two approximations of solutions of Hamilton-Jacobi equations. *Mathematics of Computation*, 43(167):1–19, 1984.

[8] R. Fedkiw, T. Aslam, B. Merriman, and S. Osher. A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the ghost fluid method). *Journal of Computational Physics*, 152:457–492, 1999.

[9] Edward A. Lee. Soft walls: Frequently asked questions. Technical Report UCB/ERL M03/31, Department of Electrical Engineering and Computer Science, University of California, Berkeley, July 2003.

[10] John Lygeros. Reachability, viability and invariance: An approach based on minimum cost optimal control. Technical Report CUED/F-INFENG/TR.444, Department of Engineering, University of Cambridge, 2002.

[11] A. W. Merz. The game of two identical cars. *Journal of Optimization Theory and Applications*, 9(5):324–343, 1972.

[12] `http://cherokee.stanford.edu/~mitchell`.

[13] I. Mitchell. Games of two identical vehicles. Technical Report SUDAAR 740, Department of Aeronautics and Astronautics, Stanford University, Stanford, CA, July 2001.

[14] I. Mitchell, A. Bayen, and C. J. Tomlin. Computing reachable sets for continuous dynamic games using level set methods. Submitted January 2004 to *IEEE Transactions on Automatic Control*.

[15] I. Mitchell, A. Bayen, and C. J. Tomlin. Validating a Hamilton-Jacobi approximation to hybrid system reachable sets. In M. D. Di Benedetto and A. Sangiovanni-Vincentelli, editors, *Hybrid Systems: Computation and Control*, number 2034 in Lecture Notes in Computer Science, pages 418–432. Springer Verlag, 2001.

[16] I. Mitchell and C. Tomlin. Level set methods for computation in hybrid systems. In B. Krogh and N. Lynch, editors, *Hybrid Systems: Computation and Control*, number 1790 in Lecture Notes in Computer Science, pages 310–323. Springer Verlag, 2000.

[17] Ian M. Mitchell. *Application of Level Set Methods to Control and Reachability Problems in Continuous and Hybrid Systems*. PhD thesis, Scientific Computing and Computational Mathematics Program, Stanford University, August 2002.

[18] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer-Verlag, 2002.

[19] S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79:12–49, 1988.

[20] S. Osher and Chi-Wang Shu. High-order essentially nonoscillatory schemes for Hamilton-Jacobi equations. *SIAM Journal on Numerical Analysis*, 28(4):907–922, 1991.

[21] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang. A PDE based fast local level set method. *Journal of Computational Physics*, 165:410–438, 1999.

[22] W. Rider and D. Kothe. A marker particle method for interface tracking. In H. Dwyer, editor, *Proceedings of the* 6^{th} *International Symposium on Computational Fluid Dynamics*, pages 976–981, 1995.

[23] J. A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences, USA*, 93(4):1591–1595, 1996.

[24] J. A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, New York, 1999.

[25] Chi-Wang Shu and Stanley Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes. *Journal of Computational Physics*, 77:439–471, 1988.

[26] M. Sussman, P. Smereka, and S. Osher. An improved level set method for incompressible two-phase flow. *Journal of Computational Physics*, 114:145–159, 1994.

[27] C. Tomlin, J. Lygeros, and S. Sastry. A game theoretic approach to controller design for hybrid systems. *Proceedings of the IEEE*, 88(7):949–970, July 2000.

[28] D. Torres and J. Brackbill. The point-set method: Front tracking without connectivity. *Journal of Computational Physics*, 165:620–644, 2000.

[29] J. N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, AC-40(9):1528–1538, 1995.