Case Studies for Computing Density of Reachable States for Safe Autonomous Motion Planning

Yue Meng[1(⊠)], Zeng Qiu[2], Md Tawhid Bin Waez[2], and Chuchu Fan[1]

[1] Massachusetts Institute of Technology, Cambridge, USA
mengyue@mit.edu
[2] Ford Motor Company, Dearborn, USA

Abstract. Density of the reachable states can help understand the risk of safety-critical systems, especially in situations when worst-case reachability is too conservative. Recent work provides a data-driven approach to compute the density distribution of autonomous systems' forward reachable states online. In this paper, we study the use of such approach in combination with model predictive control for verifiable safe path planning under uncertainties. We first use the learned density distribution to compute the risk of collision online. If such risk exceeds the acceptable threshold, our method will plan for a new path around the previous trajectory, with the risk of collision below the threshold. Our method is well-suited to handle systems with uncertainties and complicated dynamics as our data-driven approach does not need an analytical form of the systems' dynamics and can estimate forward state density with an arbitrary initial distribution of uncertainties. We design two challenging scenarios (autonomous driving and hovercraft control) for safe motion planning in environments with obstacles under system uncertainties. We first show that our density estimation approach can reach a similar accuracy as the Monte-Carlo-based method while using only 0.01X training samples. By leveraging the estimated risk, our algorithm achieves the highest success rate in goal reaching when enforcing the safety rate above 0.99.

Keywords: Reachability analysis · State density estimation · Online planning · Liouville Theorem · Neural network

1 Introduction

Verifying and enforcing the safety of the controlled systems is crucial for applications such as air collision avoidance systems [28], space exploration [32], and autonomous vehicles. It is still a challenging problem to perform online verification and controller synthesis for high-dimensional autonomous systems involving complicated dynamics and uncertainties because of the scalability issue in verification and the absence of the analytical form to describe system trajectories.

Reachability analysis is one of the main techniques used for rigorously validating the system's safeness [17,19,26,27,53] and controller synthesis [21,33,40,50].

In reachability analysis, one computes the reachable set, defined as the set of states where the system (with the control inputs) can be driven to from the initial conditions, under the system dynamics and physical constraints. Take the aircraft collision avoidance system as an example: the system safety can be guaranteed if all the future space that the airplane can reach (under physical constraints) will not overlap with obstacles. However, computing the reachable states is proved to be undecidable in general (e.g., polynomial dynamical systems with degrees larger than 2) [24] and is also empirically time-consuming, limiting applications to simple dynamics (e.g., linear systems) or low-dimension systems.

Besides, using worst-case reachability for safety analysis will usually return a binary result ("yes" or "no"), regardless of the initial state distribution and the uncertainty in the systems. The focus on the "worst-case" makes the corresponding reachability-based planning methods "conservative" or "infeasible" when the initial state has a large uncertainty. Consider a robot navigating in an environment with obstacles and state uncertainty - when a collision is inevitable in the worst case (though the worst case is a rare event), the planning algorithm will fail to return any safety-guaranteed control policies but to let the robot stop. Hence in those cases, we need a way to quantify the risk/probability of the undesired event (e.g., collision) happening and guide controller designs.

In this paper, we present a probabilistic and reachability-based planning framework for safety-critical applications. Inspired by [42], we first learn the system flow maps and the density evolution by solving the Liouville partial differential equation (PDE) using Neural Networks from collected trajectory data. Instead of using the exact reachability analysis tool [54] for reachable states probability estimation, we use Barycentric interpolation [25], which can handle more complicated systems (dimension > 4) and sharply reduces the processing time compared to [42]. In addition, by picking different numbers of sampled points, our algorithm can flexibly control the trade-off between estimation efficiency and accuracy. Leveraging this density estimation technique, our planning framework (illustrated in Fig. 1) verifies the safety of the system trajectory via a segment-by-segment checking. If one segment becomes unsafe, we perturb around the reference trajectory to find a safe alternative, and plan for the rest of the trajectory. The process repeats until all segments are enforced to be safe.

We conduct experiments on two challenging scenarios (autonomous car and hovercraft control with uncertainties). Our estimated reachable states density distribution is informative as it reflects the contraction behavior of the controllers and highlights the places that the system is more likely to reach. Quantitatively, compared to Monte Carlo density estimation, our approach can achieve a similar accuracy while only using 0.01X training samples. We test our density-based planning algorithm in 20 randomly generated testing environments (for each system), where we achieve the highest success rate in goal reaching with high safety rate (measured by one minus the collision rate) compared to other baselines.

Our contributions are: (1) we are the first group to study the use of learned reachability density in safe motion planning to ensure probabilistic safety for complicated autonomous systems, (2) our approach can estimate state density

and conduct safe planning for systems with nonlinear dynamics, state uncertainty, and disturbances, and (3) we design both qualitative and quantitative experiments for two challenging systems to validate our algorithm being accurate, data-efficient and achieving the best overall performance for the goal reaching success rate and safety.[1]

2   Related Work

Reachability analysis has been a powerful tool for system verification. The related literature has been extensively studied in [2,10,12,37]. However, few of those have been tackling the problem of calculating reachable set density distribution. Hamilton Jacobian PDE has been used to derive the exact reachable sets in [6,10,43], but this approach does not compute the density. Many data-driven methods can compute reachable sets with probabilistic guarantees using scenario optimization [14,56], convex shapes [7,35,36], support vector machines [3,48], and nonparametric methods [13,51]. However, they cannot estimate state density distribution. [20] estimates human policy distribution using a probabilistic model but requires state discretization. [39] uses the Liouville equation to maximize the backward reachable set for only polynomial system dynamics. In [1], the authors discretize the system to Markov Chains (MC) and perform probabilistic analysis on MC. This approach is computation-heavy for online safety checks.

Recently, with Neural Networks (NN) development, there has been a growing interest in studying worst-case reachability for NN [30,31,42,54,55,57] or NN-controlled systems [17,19,26,27,53]. Among those, [42] leverages the exact reachability method [54] and the Liouville Theorem to perform reachability analysis and reachable set density distribution. This approach finds the probability density function transport equation by solving the Liouville PDE using NN. It shows high accuracy in density estimation compared with histogram, kernel density estimation [11], and Sigmoidal Gaussian Cox Processes methods [15]. Hence, we choose this approach to verify the autonomous systems' safety and to conduct safe motion planning.

There have been various motion planning techniques for autonomous systems, and we refer the interested readers to these surveys [22,23,44]. Most approaches use sampling-based algorithms [29], state lattice planners [41,46], continuous optimization [8,47], and deep neural networks [9,58]. Reachable sets have also been used for safe motion planning for autonomous systems [21,33,40,50]. However, worst-case reachability-based methods only treat reachability as a binary "yes" or "no" problem without considering the density distribution of the reachable states. This boolean reachability setting makes the reachability-based motion planner conservative when the collision is inevitable in the worst case (but only happens at a very low probability) thus the system cannot reach the goal state. In this paper, we integrate the density-based reachability estimation method in [42] with model predictive control to improve the goal reaching success rate while enforcing the systems' safety in high probability.

---

[1] The code is available at https://github.com/mengyuest/density_planner.

3   Problem Formulation

Consider a controlled system $\dot{q} = f(q, u)$ where $q \in \mathcal{Q} \subseteq \mathbb{R}^d$ denotes the system state (e.g., position and heading) and $u \in \mathcal{U} \subseteq \mathbb{R}^z$ denotes the control inputs (e.g., thrust and angular velocity). For a given control policy $\pi : \mathcal{Q} \to \mathcal{U}$, the system becomes an autonomous system $\dot{q} = f(q, \pi(q)) = f_\pi(q)$ that the future state q_t at time t will only depends on the initial state q_0. We assume the initial state $q_0 \in \mathcal{Q}_0 \subseteq \mathcal{Q}$. Then, the forward reachable set at time t is defined as:

$$\mathcal{Q}_t = \{q_t \mid q_0 \in \mathcal{Q}_0, \dot{q} = f_\pi(q)\} \tag{1}$$

Assume the initial state q_0 follows a distribution $\mathcal{D}$ with the support $\mathcal{Q}_0$. Given obstacles $\{\mathcal{O}_i \subseteq \mathbb{R}^p\}_{i=1}^M$ in the environment, we aim to compute the probability for colliding with obstacles and the forward probabilistic reachability defined below:

Definition 1 (Collision probability estimation). *Given a system $\dot{q} = f_\pi(q)$ with initial state distribution $\mathcal{D}$, compute the probability for states colliding with an obstacle $\mathcal{O}$ at time t: $P_t(\mathcal{O}) = Prob\{q_0 \sim \mathcal{D}, \dot{q} = f_\pi(q), \Pi(q_t) \in \mathcal{O}\}$ where $\Pi : \mathcal{Q} \to \mathbb{R}^p$ projects the system state to the space that the obstacle $\mathcal{O}$ resides.*

Definition 2 (Forward probabilistic reachability estimation). *Given a system $\dot{q} = f_\pi(q)$ with initial state distribution $\mathcal{D}$, for each time step t, estimate the forward reachable set $\mathcal{Q}_t$ and the probability distribution $\{(\mathcal{A}_i, P_t(\mathcal{A}_i))\}_{i=1}^{N_t}$. Here $\mathcal{A}_1, ..., \mathcal{A}_{N_t}$ is a non-overlapping partition for $\mathcal{Q}_t$, i.e., $\mathcal{A}_i \cap \mathcal{A}_j = \varnothing, \forall i \neq j$, $\bigcup_{i=1}^{N_t} \mathcal{A}_i = \mathcal{Q}_t$.*

Assume π is a tracking controller: $\pi(q) = u(q, q^{ref})$ with a reference trajectory $\{q_t^{ref}\}_{t=1}^T$ of length T first generated from a high-level planner with commands $U^{ref} = \{u_t^{ref}\}_{t=1}^T$. Define the *total collision risk*:

$$\text{Prob(colliding)} = P_c(U^{ref}) = \sum_{t=1}^T \sum_{i=1}^M P_t(\mathcal{O}_i) \tag{2}$$

We are interested in the following problem:

Definition 3 (Safety verification and planning problem). *Given a system $\dot{q} = f_\pi(q)$ with initial state distribution $\mathcal{D}$ and reference control commands U^{ref}, verify the total collision risk $P_c(U^{ref}) \leq \gamma$, where γ is a tolerant collision risk threshold. If not, plan a new command $\tilde{U}^{ref}$ to ensure $P_c(\tilde{U}^{ref}) \leq \gamma$*

In this paper, the details about the dynamic systems $\dot{q} = f_\pi(q)$ and controllers $\pi(q) = u(q, q^{ref})$ are listed in the A and B.

4   Technical Approaches

Inspired by [42], we design a sample-based approach to compute the reachability and the density distribution for the system described in the previous section and further leverage these results for trajectory planning for autonomous systems.

4.1 Data-driven Reachability and Density Estimation

Our framework is built on top of a recently published density-based reachability analysis method [42]. From the collected trajectory data, [42] learns the system flow map and the state density concentration function jointly, guided by the fact that the state density evolution follows the Liouville partial differential equation (PDE). With the set-based reachability analysis tools RPM [54], they can estimate the bound for the reachable set probability distribution.[2]

For the autonomous system defined in Sect. 3, we denote the density function $\rho : \mathcal{Q} \times \mathbb{R} \to \mathbb{R}^{\geq 0}$ which measures how states distribute in the state space at a specific time step. The density function is completely determined by the underlying dynamics f_π and the initial density map $\rho_0 : \mathcal{Q} \to \mathbb{R}^{\geq 0}$ according to the Liouville PDE [16].

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \cdot f_\pi) = 0, \quad \rho(q, 0) = \rho_0(q) \tag{3}$$

We define the flow map $\Phi : \mathcal{Q} \times \mathbb{R} \to \mathcal{Q}$ such that $\Phi(q_0, t)$ is the state at time t starting from q_0 at time 0. The density along the trajectory $\Phi(q_0, t)$ is an univariate function of t, i.e., $\rho(t) = \rho(\Phi(q_0, t), t)$. If we consider the augmented system with states $[q, \rho]$, from Eq. 3 we can get the dynamics of the augmented system:

$$\begin{bmatrix} \dot{q} \\ \dot{\rho} \end{bmatrix} = \begin{bmatrix} f_\pi(q) \\ -\nabla \cdot f_\pi(q)\rho \end{bmatrix} \tag{4}$$

To compute the state and the density at time T from the initial condition $[q_0, \rho_0(q_0)]$, one can solve the Eq. 4 and the solution at time T will give the desired density value. To accelerate the computation process for a large number of initial points, we use neural networks to estimate the density $\rho(q, t)$ and the flow map $\Phi(q, t)$. Details for the network training are introduced in Sect. 3 of [42].

4.2 Reach Set Probability Estimation

As mentioned in [42], when the system state is high (≥ 4), it is either infeasible (due to the numerical issue in computing for polyhedra) or too time-consuming to generate RPM results for probability estimation. The state dimension will become 7–10 for a 2D car control or a 3D hovercraft control problem after including the reference control inputs. If we use other worst-case reachability analysis tools such as [17] to compute the probability, the reachable set will be too conservative and the planner will not return a feasible solution (other than stop) because the reachable states will occupy the whole state space regardless of the choice of the reference controls. Therefore, we use a sample-based approach to estimate the probability of the reachable sets, as introduced in the following.

To estimate the probability in Prob. 1, we first uniformly sample initial states $\{q_0^i\}_{i=1}^{N_s}$ from the support of the distribution ρ_0 and use the method in Sect. 4.1

---

[2] For details about computing the probability of the reachable state, we refer the interested readers to [42](Appendix B).

to estimate the future states and the corresponding densities at time t denoted as $\{(q_t^i, \rho_t(q_t^i))\}_{i=1}^{N_s}$. We approximate the forward reachable set $\mathcal{Q}_t$ defined in Eq. 1 as the convex hull of $\{q_t^i\}_{i=1}^{N_s}$, and denote it as $\mathcal{CH}_t$. Then, based on $\{(q_t^i, \rho_t(q_t^i))\}_{i=1}^{N_s}$, we use the linear interpolation to estimate the density distribution $\hat{\rho}_t(\cdot)$ at time t. Finally, we uniformly sample points q_s within the convex hull $\mathcal{CH}_t$, and the probability for the system reaching $\mathcal{A}$ can be computed as:

$$\text{Prob}(q_t \in \mathcal{A}) \approx \frac{\sum_{q_s \sim \mathcal{CH}_t} \mathbb{1}\{q_s \in \mathcal{A}\}\hat{\rho}_t(q_s)}{\sum_{q_s \sim \mathcal{CH}_t} \hat{\rho}_t(q_s)} \tag{5}$$

Here are some remarks for our approach. The probabilistic guarantee about estimation accuracy is provided in [42][Appendix A]. Besides, our approach will return probability zero if the ground truth probability of reaching $\mathcal{A}$ is zero. Moreover, compared to the set-based approach RPM, which has poor scalability because of the number of polyhedral cells growing exponentially to the system state dimension, our sample-based approach is fast, and the runtime can be controlled by selecting different numbers of sampled points as a trade-off between efficiency and accuracy.
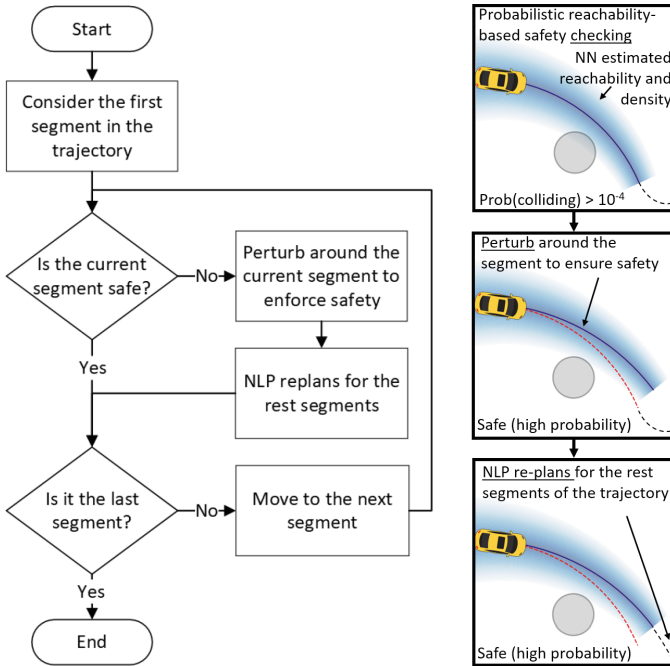
(a) Flow chart of the algorithm   (b) Planning process

Fig. 1. The safe planning algorithm.

4.3   Motion Planning Based on Reachability Analysis

After estimating the reachable states and density for the autonomous system under a reference trajectory, we utilize the results to plan feasible trajectories to ensure the collision probability is under a tolerable threshold.

In this paper, the reference trajectory is generated using nonlinear programming (NLP). Given the origin state q_{origin}, the destination state q_{dest}, the physical constraints $u_{min} \leq u \leq u_{max}$ and M obstacles $\{(x_i^o, y_i^o)\}_{i=0}^{M-1}$ (with radius $\{r_i\}_{i=0}^{M-1}$) in the environment, discrete time duration Δ_t and total number of timesteps T, we solve an NLP using CasADI [5] to generate a reference trajectory, which consists of N trajectory segments $\xi_0, ..., \xi_{N-1}$ (each segment ξ_j has length L and is generated by $q_{j \cdot L}$ and u_j). The details about this nonlinear optimization formulation can be found in Appendix C

Then for each segment ξ_j, with the uncertainty and disturbances considered, we use the approach in Sect. 4.2 to estimate the system's reachable states as well as their density. If the total collision risk defined in Eq. 2 is below a predefined threshold (10^{-4} in our case), we call the current trajectory "safe". Otherwise, we call the trajectory "unsafe" and adjust for the current trajectory segment. Notice that the traditional reachability-based planning is just a special case when we set this threshold to 0.

To ensure fast computation for the planning, we use the perturbation method to sample candidate trajectory segments around this "unsafe" trajectory segment ξ_j (by adding Δu to the reference control commands) and again use the method in Sect. 4.2 to verify whether the candidate is "unsafe", until we find one segment $\tilde{\xi}_j$ that is "safe", and then we conduct the NLP starting from the endpoint of the segment $\tilde{\xi}_j$. We repeat this process until all the trajectory segments are validated to be "safe". The whole process is summarized in Algorithm 1.

Given enough sampled points with guaranteed correctness in approximating state density and forward reachable set, the algorithm is sound because the produced control inputs will always ensure the system is "safe". However, our algorithm is not complete because: (1) in general, the nonlinear programming is not always feasible, and (2) the perturbation method might not be able to find a feasible solution around the "unsafe" trajectory. The first point can be addressed by introducing slack variables to relax for the safety and goal-reaching constraints. The second point can be tackled by increasing the tolerance probability threshold of collision.

5   Experiments

We investigate our approach in autonomous driving and hovercraft navigation applications under the following setup: given an environment with an origin point, a destination region, and obstacles, the goal for the agent at the origin point is to reach the destination while avoiding all the obstacles. Notice that this is a very general setup to encode the real-world driving scenarios because: (1) the road boundaries and other irregular-shaped obstacles can be represented

---

Algorithm 1. Reachability-based Planning Algorithm

Input: Origin S_0, destination S_N, NLP constraints
Output: Reference trajectories $\xi_0, \xi_1, \cdots, \xi_{N-1}$

1: $i \leftarrow 0$
2: **while** $i < N$ **do**
3:     Generate segments $\xi_i, ..., \xi_{N-1}$ from S_i to S_N using NLP
4:     **for** $j = i : N$ **do**
5:         Use the method in Sect. 4.2 to check whether the trajectory segment ξ_j is "safe".
6:         **if** The trajectory is "safe" **then**
7:             Continue
8:         **else**
9:             Perturb the segment ξ_j to search for a possible "safe" segment $\tilde{\xi}_j$ (goes from S_j to close to S_{j+1}).
10:             $\xi_j \leftarrow \tilde{\xi}_j$
11:             $S_{j+1} \leftarrow \tilde{S}_{j+1}$
12:         **end if**
13:     **end for**   ▷ By far, $S_0 \rightarrow S_{j+1}$ is "safe"
14:     $i \leftarrow j + 1$ ▷ Next step will inspect $S_{j+1} \rightarrow \cdots S_N$
15: **end while**

---

by using a set of obstacles, and (2) other road participants (pedestrians, other driving cars) can be modeled as moving obstacles. Here we consider only the center of mass of the car/hovercraft in rendering reachable sets and planning (we can bloat the radius of the obstacle to take the car/hovercraft length and width into account). In Sect. 5.1, we evaluate the reachability and density for the system under a fixed reference trajectory. In Sect. 5.2, we leverage the reachability and density result to do trajectory re-planning when the system is "unsafe".

We collect 50,000 trajectories from the simulator, with randomly sampled initial states, reference trajectories, and disturbances. Each trajectory has 50 timesteps with a duration of 0.02s at each time step. Then, we select 40,000 for the training set and 10,000 for the evaluation set and train a neural network for estimating the future states and the density evolution mentioned in [42]. We use a fully connected ReLU-based neural network with 3 hidden layers and 128 hidden units in each layer. We train the neural network for 500k epochs, using stochastic gradient descent with a batch size of 256 and a learning rate of 0.1. The code is implemented in PyTorch [45], and the training takes less than an hour on an NVidia RTX 2080Ti GPU.

5.1   Reachable States and Density Estimation

In this section, we first conceptually show how our approach of estimating reachable states and density can benefit safety-critical applications. As depicted in

Fig. 4, a car plans to move to the destination (the red arrow) while avoiding all the obstacles on the road. The initial state of the car (X,Y position, and heading angle) and the disturbance follow a Gaussian distribution. The high-level motion planner has already generated a reference trajectory (the blue line in Fig. 4), with the uncertainty owing to the initial state estimation error and the disturbance. We will show that our approach can estimate the state density distribution and reachable state accurately and can help to certify that the planned reference trajectory is not colliding with obstacles in high probability.

Visualizations of the Estimated Reachability and Density Heatmap. Using the method introduced in Sect. 4.2, we can first estimate the tracking error density distribution and marginalize it to the 2D XY-plane to get the probability heatmaps (as shown in Fig. 2(a)–(d)). Then we can transform it to the reference trajectory and check whether it has an intersection with the obstacles in the environment (as shown in Fig 2(e)).

(a) t=0.0s   (b) t=0.3s   (c) t=0.8s   (d) t=1.0s

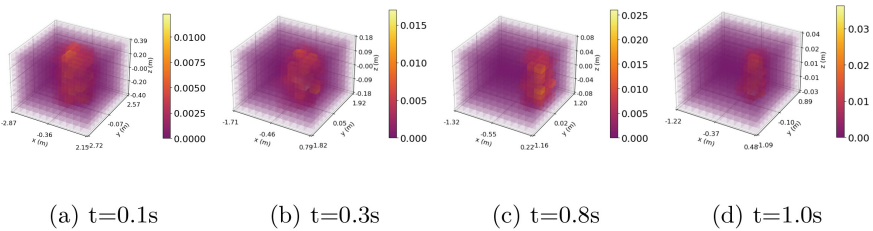(e) Density distribution on trajectories.   (f) Simulated trajectories.
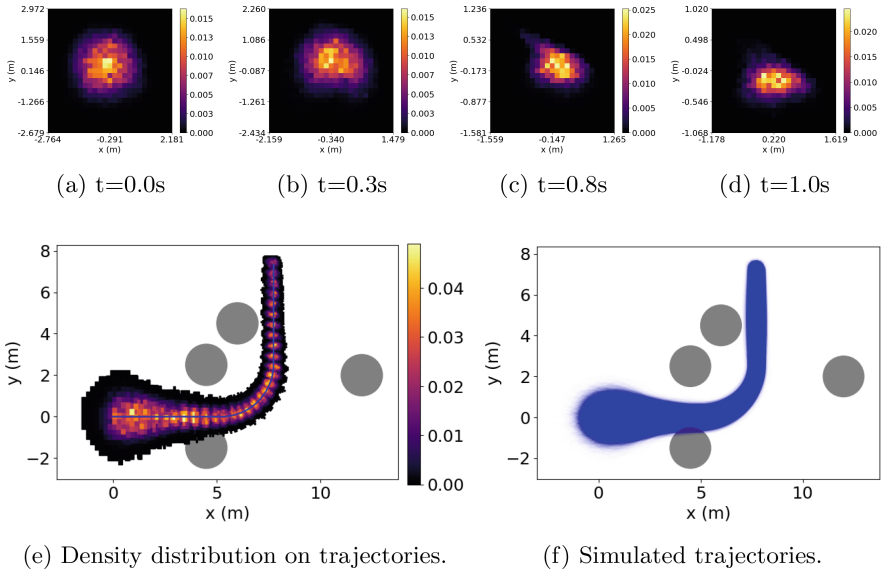
Fig. 2. Estimated density (for states in (a)–(d) and along the trajectory in (e)) for the car model. The states are shown to concentrate on reference trajectory (blue line in (e)), and the collision risk is very low. (Color figure online)

To verify the correctness of our estimated reachable states and density, we also sample a large number of states from the initial state distribution and use the ODE to simulate actual car trajectories, as shown in Fig. 2(b). Comparing Fig. 2(a) with Fig. 2(b), we find out in both cases that the vehicle will have a collision with the bottom obstacle. In addition, our density result also shows that

the risk of the collision is very low (Prob(colliding)$\leq 10^{-4}$ as shown in Sect. 5.1), which is reasonable because the majority of the states will be converging to the reference trajectory (as indicated from Fig. 2(a)–(d)). Only a few outlier trajectories will intersect with the obstacle. We also conduct this experiment with the Hovercraft system (3D scenarios), where the results in Fig. 3 reflect similar contraction behaviors, and the probability of colliding with the obstacles is very low (thus, we do not need to do planning in this stage).

Visually, our results are more informative than the pure reachability analysis because ours reflects the tracking controller's contraction behavior and illustrates that the colliding event is in very low probability. The following subsection will further quantify this probability and compare it with a traditional probability estimation method.


(a) t=0.1s   (b) t=0.3s   (c) t=0.8s   (d) t=1.0s
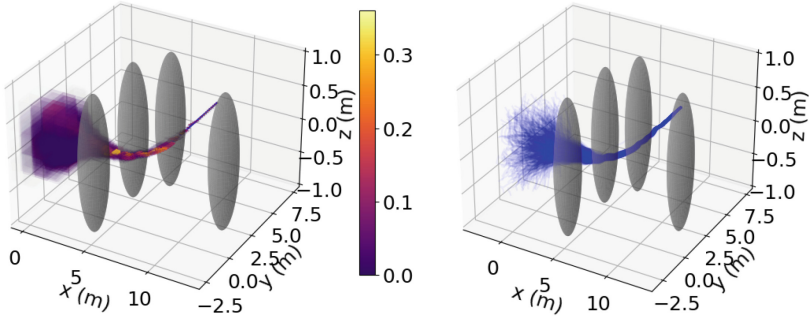
(e) Density distribution on trajectories.   (f) Simulated trajectories.

Fig. 3. Estimated density (for states in (a)–(d) and along the trajectory in (e)) for the hovercraft model. The states are shown to concentrate on reference trajectory (blue line in (e)), and the collision probability is very low. (Color figure online)

Comparison with Monte-Carlo Based Probability Estimation. Our visualization result in Sect. 5.1 reflects that under some initial conditions, the vehicle might hit the obstacle. Although Fig. 2 shows that the likelihood of the clash is very low, we want to quantify the risk of the collision to benefit future
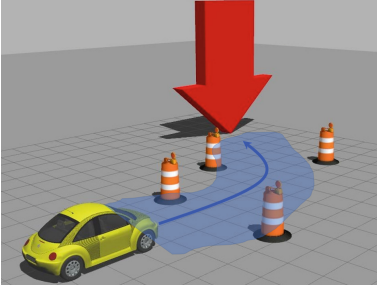
Fig. 4. A vehicle plans to reach the red arrow while avoiding all the obstacles. Blue region shows the reference trajectory and uncertainty. (Color figure online)
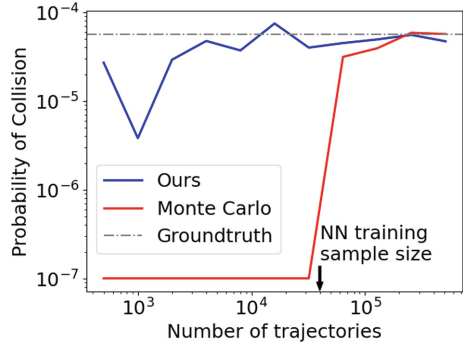
Fig. 5. Estimation of collision probability with respect to sample size.

decisions (e.g., choosing the policy with the lowest collision probability or with the lowest value at risk (VaR) [38]). However, it is intractable to derive the ground-truth probability of collision for general non-linear systems. Therefore, we compare our estimation result with the Monte Carlo approximation (this is done by generating a considerable amount of simulations and counts for the frequency of the collision.

We try different numbers of samples (from 500 to 512000) and compare our approach to the Monte Carlo estimation. As shown in Fig. 5, the groundtruth probability of collision (where we approximated by sampling 5×10^7 trajectories and compute the collision rate) is approximately 5×10^{-5}. The Monte Carlo approach fails to predict meaningful probability results until increasing the sample size to 64000. In contrast, our approach can give a non-trivial probability estimation using only 500 examples, less than 0.01X the samples needed for the Monte Carlo approach.

The black vertical arrow in Fig. 5 corresponds to the 40000 sample size (the number of samples we have used offline for our neural network training). The corresponding result is already as stable as the Monte Carlo approach which requires more than 64000 samples. Furthermore, our approach can be adapted to any initial condition for the same car dynamic system without retraining or fine-tuning, making it possible for downstream tasks like online planning, as introduced in the following section.

In terms of the computation time (for 512000 points), our approach requires 2.3X amount of time (31.8 s) as needed for the Monte Carlo method (14.1 s), mainly because that the Delaunay Triangulation method [34] used for state-space partition in the linear interpolation has the complexity of $O(N^{\lfloor d/2 \rfloor})$ [52] for N data points in the d-dimension system. Thus the run time will grow about quadratically to the number of sample points in our case (state dimension=4).

With 1000 sampled points (as used in the rest of the experiments), our method takes ∼2 s, which is acceptable. One alternative solution to accelerate the computation is to use Nearest Neighbor interpolation for density estimation.

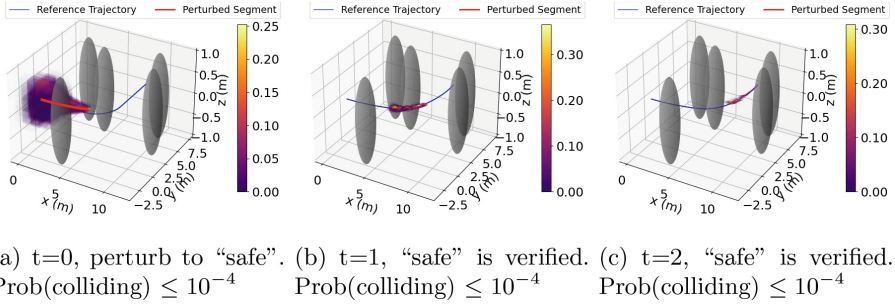## 5.2   Online Planning via Reachable Set Density Estimation

When the probability of collision is higher than the threshold (10^{-4} in our experiment setting), we need to use the planning algorithm to ensure the safety of the autonomous system under uncertainty and disturbance. We first show how the planning algorithm works in Sect. 5.2, and then quantitative assessment of our algorithm is conducted in Sect. 5.2.

Demonstration for an Example. We conduct experiments to demonstrate how our proposed reachability-based planning framework works for autonomous driving cars and hovercraft applications. In Fig. 6, a car is moving from left to right of the map while avoiding collisions with obstacles. After checking for the first segment's safety (Fig. 6(a)), the algorithm finds out the probability of the collision is higher than 10^{-4}. Thus it starts to plan for the first segment (the red line in Fig. 6(b)) around the reference trajectory (the blue line in Fig. 6(b)). Moreover, after it updates the reference trajectory using the NLP solver (the blue line in Fig. 6(c)) based on the perturbed segment, the algorithm detects the next segment as "unsafe." Hence it plans again to enforce the collision probability is below 10^{-4} (Fig. 6(c)). Another example in a 3D scenario is shown in Fig. 7, where the hovercraft perturbs for the first segment and then later verifies that the next two segments are all "safe", hence the whole trajectory in Fig. 7(c) is "safe."
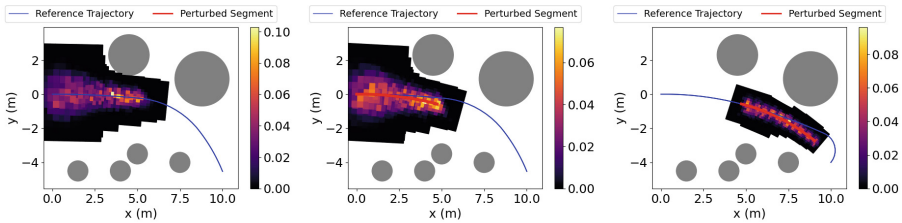
(a) t=0, "unsafe" detected.   (b) t=0, perturb to "safe".   (c) t=1, perturb to "safe".
Prob(colliding) $\geq 10^{-4}$ Prob(colliding) $\leq 10^{-4}$ Prob(colliding) $\leq 10^{-4}$

Fig. 6. Demonstration for the re-planning algorithm for the 2D car experiment. For each trajectory segment, if the probability of collision is higher than a predefined threshold, we denote the trajectory segment as "unsafe", otherwise, we denote the trajectory segment as "safe". (Color figure online)

(a) t=0, perturb to "safe". (b) t=1, "safe" is verified. (c) t=2, "safe" is verified.
Prob(colliding) $\leq 10^{-4}$ Prob(colliding) $\leq 10^{-4}$ Prob(colliding) $\leq 10^{-4}$

Fig. 7. Demonstration for the re-planning algorithm for the 3D hovercraft experiment. For each trajectory segment, if the probability of collision is higher than a predefined threshold, we denote the trajectory segment as "unsafe", otherwise, we denote the trajectory segment as "safe".

Evaluation of Trajectory Planning Performance. To illustrate the advantage of our approach in enforcing the system safety, we design 20 testing environments with randomly placed obstacles (while ensuring the initial reference trajectory is feasible from the NLP solver) for the autonomous car system and the hovercraft control system. We compare our framework with several baseline methods: the "original" approach just uses the initial reference trajectory (without any planning process), the "$d =$?" approach denotes the distance-based planning methods with the safety distance threshold set as d (the larger d is, the more conservative and safer the algorithm will be, at the cost of infeasible solutions), the "reach" approach uses estimated reachable tube computed from the sampled convex hull $\mathcal{CH}$ introduced in Sect. 4.2 to do safe planning.

We measure the performances of different methods using two metrics: feasibility and safety. *Feasibility* is defined as the frequency that the algorithm can return a plausible solution (but might be unsafe) to reach the designed destination. Over all feasible solutions, *safety* is defined as the expected collision probability. Intuitively, the feasibility measures the successful rate of the goal reaching, and the "safety" measures how "reliable" the planner is.

As shown in Fig. 8(a)(b) for the car experiments, compared to the "reach" method (which just uses reachable tubes to do planning) and a distance-based planning method ("$d = 1.0$"), our approach can achieve a similar safety rate, while having 0.29–0.76 higher feasibility (due to less conservative planning). Though we are 0.059 less in feasibility than the "original" and the "d = 0.1" baselines, they lead to 3772X–4574X collision rates than ours (due to our approach being less aggressive in planning). Compared to the "$d = 0.2$" method, we are 0.06 better in feasibility (0.94 vs 0.88) while being only 0.2X in collision rate (0.000025 vs 0.000125), we also get a lower collision rate . A similar trend can also be observed from the hovercraft experiment (Fig. 8(c)(d)). Hence, our approach achieves the best overall performance by considering feasibility and safety.
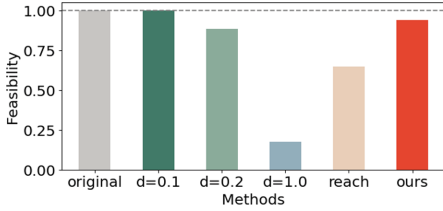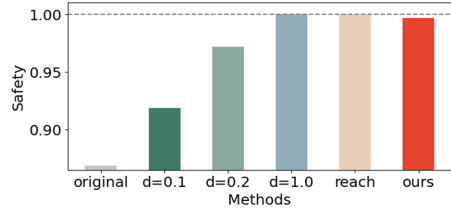
(a) Car model, feasibility comparisons
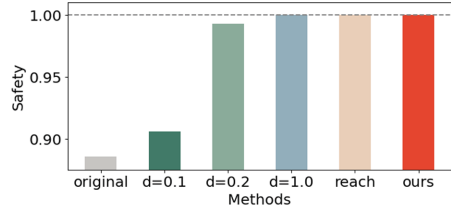
(b) Car model, safety comparisons
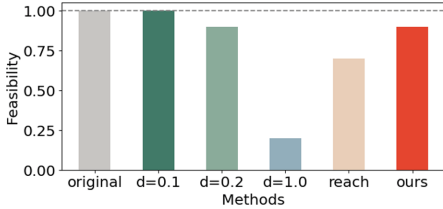
(c) Hovercraft, feasibility comparisons

(d) Hovercraft, safety comparisons

Fig. 8. Feasibility and safety comparisons. Our method achieves the best trade-off between the feasibility and the safety, with close to 100% safety and 0.2X–4X improvement in feasibility comparing to high-safety methods. Here "original" uses nonlinear programming for planning , "$d =$?" denotes the distance-based re-planning with the safety distance d, "reach" uses estimated reachable tube to do re-planning, and "ours" leverages both the reachable tube and the corresponding density to do re-planning. More details can be found in Sect. 5.2

5.3   Discussions

While our approach can accurately estimate the collision probability and the motion planner using our estimated density can achieve the highest goal reaching rate compared to other baselines when enforcing the safety rate above 0.99, we admit there are assumptions and limitations in our method.

First we assume the neural network can learn a perfect state dynamic and state density evolution, which is not always satisfied due to the model capacity and the complexity of the system. The proof in [42][Appendix A] shows the generalization error bound for this learning framework, which indicates one possible remedy is to collect more training trajectories.

Besides, we assume the sampled trajectories from the simulator are following the same distribution as for the real world trajectories. This assumption might create biases in the density concentration function and the flow map estimation. One way to resolve this issue is to further fine-tuning the neural network using the real world data at the inference stage.

The first limitation of our approach is lacking guarantee for the convergence of the planning algorithm. The success planning rate of our method depends on the perturbation range (how far the control policy can deviate from the reference policy) and the perturbation resolution (the minimum difference between two

candidate policies). There are also optimization-based methods, such as stochastic gradient descent [4], that can converge with probabilistic guarantee derived from the Robbins-Siegmund theorem [49]. Using optimization-based method for density-based planning is left to our future work.

The second limitation of our planning framework is the computation time. This is mainly due to our risk computation step in Eq. 2, as mentioned in Sect. 5.1. Although our proposed probability computation method can handle higher dimension systems than [42], the complexity of the Delaunay Triangulation process in our framework grows in the power of $\lfloor d/2 \rfloor$ for a d-dimensional system. In practice, one can use less number of sample points to reduce the computation time. Another alternative is to use other interpolation methods (e.g., Nearest Neighbor interpolation) for d-dimensional space.

6   Conclusion

We propose a data-driven framework for probabilistic verification and safe motion planning for autonomous systems. Our approach can accurately estimate collision risk, using only 0.01X training samples compared to the Monte Carlo method. We conduct experiments for autonomous driving and hovercraft control, where the car (hovercraft) with state uncertainty and control input disturbances plans to move to the destination while avoiding all the obstacles. We show that our approach can achieve the highest goal reaching rate among all approaches that can enforce the safety rate above 0.99. For future works, we manage to develop verification approaches for cases that consider (1) other road participants' presence and intention and (2) more complicated sensory inputs, such as LiDAR measurements or even raw camera inputs.

Appendix

A   Car Model Dynamic and Controller Designs

The dynamics for the rearwheel kinematic car model [18] is:

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{6}$$

where (x, y) denotes the position of the vehicle's center of mass, θ denotes vehicle's heading angle, and v, ω are the velocity and angular velocity control

inputs. Given a reference trajectory generated from the motion planner $(x^{ref}, y^{ref}, \theta^{ref})^T$, the error for the car model is:

$$
\begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - x^{ref} \\ y - y^{ref} \\ \theta - \theta^{ref} \end{bmatrix} \tag{7}
$$

The Lyapunov-based controller is designed as:

$$
\begin{cases} v = v^{ref} \cos(e_\theta) + k_1 e_x + d_v \\ \omega = \omega^{ref} + v^{ref}(k_2 e_y + k_3 \sin(e_\theta)) + d_\omega \end{cases} \tag{8}
$$

where k_1, k_2, k_3 are the coefficients for the controller and d_v and d_ω are the controller disturbances.

B   Hovercraft Model Dynamic and Controller Designs

The hovercraft is the model tested in 3D scenarios. The dynamics for the hovercraft [18] is:

$$
\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & 0 \\ \sin\theta & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ v_z \\ \omega \end{bmatrix} \tag{9}
$$

where (x, y, z) denotes the 3D position of the hovercraft's center of mass, θ denotes the heading angle of the hovercraft in the xy-plane, v (and ω) denotes the velocity (and angular velocity) in the xy-plane, v_z denotes the velocity along the z-axis. When a reference trajectory $(x^{ref}, y^{ref}, z^{ref}, \theta^{ref})^T$ is introduced, the error for the car model is:

$$
\begin{bmatrix} e_x \\ e_y \\ e_z \\ e_\theta \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 & 0 \\ -\sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - x^{ref} \\ y - y^{ref} \\ z - z^{ref} \\ \theta - \theta^{ref} \end{bmatrix} \tag{10}
$$

The Lyapunov-based controller is designed as:

$$
\begin{cases} v = v^{ref} \cos(e_\theta) + k_1 e_x + d_v \\ v_z = v_z^{ref} + k_4 e_z + d_{v_z} \\ \omega = \omega^{ref} + v^{ref}(k_2 e_y + k_3 \sin(e_\theta)) + d_\omega \end{cases} \tag{11}
$$

where k_1, k_2, k_3, k_4 are the coefficients for the controller and d_v, d_{v_z} and d_ω are the corresponding disturbances.

C   Nonlinear Programming for Controller Synthesize

The goal of this section is to find a control sequence $\{u_j\}_{j=0}^{N-1}$ for the car (or the hovercraft, we use "robot" to represent them in the following context) starting from $q_{origin} \in \mathbb{R}^d$ to reach the goal state $q_{dest} \in \mathbb{R}^d$ in T time steps, while satisfying the physical constraints and avoiding colliding with the surrounding obstacles (M obstacles in total) in the environment. We use the forward Euler method to compute the ODE $\dot{q} = f(q, u)$, with each time step duration as Δt. Each control input u_j will last for $L = \lceil \frac{T}{N} \rceil$ steps. For the physical constraints, we set up the maximum and minimum allowed value for the control inputs as $u_{max}, u_{min} \in \mathbb{R}^z$. We represent the obstacles as circles (and spheres in 3D scenarios). The i-th obstacle has a center position $\bar{q}_i^o \in \mathbb{R}^2$ ($\bar{q}_i^o \in \mathbb{R}^3$ in 3D scenarios) and a radius $r_i \in \mathbb{R}^+$ (we use $\bar{q}_j$ to represent the robot position at time j, to distinguish with the full robot state q_j). We formulate the optimization process as followed:

$$\min_{u_{0:N-1}} \quad \sum_{i=0}^{M-1} \sum_{j=0}^{T} \gamma_{i,j}^2$$

$$\text{s.t.} \quad q_0 = q_{origin}$$
$$q_T = q_{dest}$$
$$u_{min} \leq u_j \leq u_{max}, \forall j = 0...N-1$$
$$q_{j\cdot L+k+1} = q_{j\cdot L+k} + f(q_{j\cdot L+k}, u_j)\Delta t, \forall j = 0...N-1, \forall k = 0...L-1$$
$$|\bar{q}_j - \bar{q}_i^o|^2 + \gamma_{i,j}^2 \geq r_i{}^2, \forall i = 0..., M-1, \forall j = 0, ..., T$$

$$(12)$$

where the first two constraints make sure the robot starts from the initial point and will reach the goal point, the third and forth constraints enforce the physical constraints and the robot dynamic, and the last constraint ensure the robot will not hit obstacles at any time. For feasibility issues, slack variables $\gamma_{i,j}$ are introduced to relax the collision avoidance constraint (the robot safety will be checked and ensured during the online planning process after this optimization process).

References

1. Abate, A.: Probabilistic Reachability for Stochastic Hybrid Systems: Theory, Computations, and Applications. University of California, Berkeley (2007)
2. Agha, G., Palmskog, K.: A survey of statistical model checking. ACM Trans. Model. Comput. Simul. (TOMACS) **28**(1), 1–39 (2018)
3. Allen, R.E., Clark, A.A., Starek, J.A., Pavone, M.: A machine learning approach for real-time reachability analysis. In: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2202–2208. IEEE (2014)
4. Amari, S.: Backpropagation and stochastic gradient descent method. Neurocomputing **5**(4–5), 185–196 (1993)

5. Andersson, J.A.E., Gillis, J., Horn, G., Rawlings, J.B., Diehl, M.: Casadi: a software framework for nonlinear optimization and optimal control. Math. Program. Comput. **11**(1), 1–36, 2019

6. Bansal, S., Tomlin, C.: Deepreach: A deep learning approach to high-dimensional reachability. arXiv preprint arXiv:2011.02082 (2020)

7. Berndt, A., Alanwar, A., Johansson, K.H., Sandberg, H.: Data-driven set-based estimation using matrix zonotopes with set containment guarantees. arXiv preprint arXiv:2101.10784 (2021)

8. Chen, J., Liu, C., Tomizuka, M.: Foad: fast optimization-based autonomous driving motion planner. In: 2018 Annual American Control Conference (ACC), pp. 4725–4732. IEEE (2018)

9. Chen, L., Xuemin, H., Tian, W., Wang, H., Cao, D., Wang, F.-Y.: Parallel planning: a new motion planning framework for autonomous driving. IEEE/CAA J. Autom. Sin. **6**(1), 236–246 (2018)

10. Chen, M., Tomlin, C.J.: Hamilton-jacobi reachability: some recent theoretical advances and applications in unmanned airspace management. Ann. Rev. Control Robot. Autonom. Syst. **1**, 333–358 (2018)

11. Chen, Y., Ahmadi, M., Ames, A.D.: Optimal safe controller synthesis: a density function approach. In: 2020 American Control Conference (ACC), pp. 5407–5412. IEEE (2020)

12. International competition on verifying continuous and hybrid systems. https://cps-vo.org/group/ARCH/FriendlyCompetition. Accessed 18 June 2021

13. Devonport, A., Arcak, M.: Data-driven reachable set computation using adaptive gaussian process classification and Monte Carlo methods. In: 2020 American Control Conference (ACC), pp. 2629–2634. IEEE (2020)

14. Devonport, A., Arcak, M.: Estimating reachable sets with scenario optimization. In: Learning for Dynamics and Control, pp. 75–84. PMLR (2020)

15. Donner, C., Opper, M.: Efficient bayesian inference of sigmoidal gaussian cox processes. https://doi.org/10.14279/depositonce-8398 (2018)

16. Ehrendorfer, M.: The liouville equation and prediction of forecast skill. In: Predictability and Nonlinear Modelling in Natural Sciences and Economics, pp. 29–44. Springer (1994)

17. Everett, M., Habibi, G., Jonathan, P.: How Efficient reachability analysis of closed-loop systems with neural network controllers. arXiv preprint arXiv:2101.01815 (2021)

18. Fan, C., Miller, K., Mitra, S.: Fast and guaranteed safe controller synthesis for nonlinear vehicle models. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 629–652. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_31

19. Fan, J., Huang, C., Chen, X., Li, W., Zhu, Q.: ReachNN*: a tool for reachability analysis of neural-network controlled systems. In: Hung, D.V., Sokolsky, O. (eds.) ATVA 2020. LNCS, vol. 12302, pp. 537–542. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59152-6_30

20. Fridovich-Keil, D., et al.: Confidence-aware motion prediction for real-time collision avoidance1. Int. J. Robot. Res. **39**(2–3), 250–265 (2020)

21. Gerdts, M., Xausa, I.: Avoidance trajectories using reachable sets and parametric sensitivity analysis. In: Hömberg, D., Tröltzsch, F. (eds.) CSMO 2011. IAICT, vol. 391, pp. 491–500. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36062-6_49

22. Goerzen, C., Kong, Z., Mettler, B.: A survey of motion planning algorithms from the perspective of autonomous uav guidance. J. Intell. Robot. Syst. **57**(1), 65–100 (2010)
23. González, D., Pérez, J., Milanés, V., Nashashibi, F.: A review of motion planning techniques for automated vehicles. IEEE Trans. Intell. Transp. Syst. **17**(4), 1135–1145 (2015)
24. Hainry, E.: Decidability and undecidability in dynamical systems. In: Research Report (2009)
25. Hormann, K.: Barycentric interpolation. In: Approximation Theory XIV: San Antonio 2013, pp. 197–218. Springer (2014)
26. Hu, H., Fazlyab, M., Morari, M., Pappas, G.J.: Reach-sdp: Reachability analysis of closed-loop systems with neural network controllers via semidefinite programming. In: 2020 59th IEEE Conference on Decision and Control (CDC), pp. 5929–5934. IEEE (2020)
27. Ivanov, R., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verisig: verifying safety properties of hybrid systems with neural network controllers. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, pp. 169–178 (2019)
28. Julian, K.D., Kochenderfer, M.D.: Reachability analysis for neural network aircraft collision avoidance systems. J. Guidance Control Dyn. **44**(6), 1132–1142 (2021)
29. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. Int. J. Robot. Res. **30**(7), 846–894 (2011)
30. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 97–117. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_5
31. Katz, G., et al.: The marabou framework for verification and analysis of deep neural networks. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 443–452. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_26
32. Kornfeld, R.P., Prakash, R., Devereaux, A.S., Greco, M.E., Harmon, C.C., Kipp, D.M.: Verification and validation of the mars science laboratory/curiosity rover entry, descent, and landing system. J. Spacecraft Rockets **51**(4), 1251–1269 (2014)
33. Kousik, S., Vaskov, S., Fan, B., Johnson-Roberson, M., Vasudevan, R.: Bridging the gap between safety and real-time performance in receding-horizon trajectory design for mobile robots. Int. J. Robot. Res. **39**(12), 1419–1469 (2020)
34. Lee, D.T., Schachter, B.J.: Two algorithms for constructing a delaunay triangulation. Int. J. Comput. Inf. Sci. **9**(3), 219–242 (1980)
35. Lew, T., Pavone, M.: Sampling-based reachability analysis: A random set theory approach with adversarial sampling. arXiv preprint arXiv:2008.10180 (2020)
36. Liebenwein, L., Baykal, C., Gilitschenski, I., Karaman, S., Rus, D.: Sampling-based approximation algorithms for reachability analysis with provable guarantees. RSS (2018)
37. Liu, C., Arnon, T., Lazarus, C., Strong, C., Barrett, C., Kochenderfer, M.J.: Algorithms for verifying deep neural networks. arXiv preprint arXiv:1903.06758 (2019)
38. Majumdar, A., Pavone, M.: How should a robot assess risk? towards an axiomatic theory of risk in robotics. In: Amato, N.M., Hager, G., Thomas, S., Torres-Torriti, M. (eds.) Robotics Research. SPAR, vol. 10, pp. 75–84. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-28619-4_10
39. Majumdar, A., Vasudevan, R., Tobenkin, M.M., Tedrake, R.: Convex optimization of nonlinear feedback controllers via occupation measures. Int. J. Robot. Res. **33**(9), 1209–1230 (2014)

40. Manzinger, S., Pek, C., Althoff, M.: Using reachable sets for trajectory planning of automated vehicles. IEEE Trans. Intell. Veh. **6**(2), 232–248 (2020)
41. McNaughton, M., Urmson, C., Dolan, J.M., Lee, J.W.: Motion planning for autonomous driving with a conformal spatiotemporal lattice. In: 2011 IEEE International Conference on Robotics and Automation, pp. 4889–4895. IEEE (2011)
42. Meng, Y., Sun, D., Qiu, Z., Waez, M.T.B., Fan, C.: Learning density distribution of reachable states for autonomous systems. arXiv preprint arXiv:2109.06728 (2021)
43. Mitchell, I.M., Bayen, A.M., Tomlin, C.J.: A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games. IEEE Trans. Autom. Control **50**(7), 947–957 (2005)
44. Paden, B Čáp, M., Yong, S.Z., Yershov, D., Frazzoli, E.: A survey of motion planning and control techniques for self-driving urban vehicles. IEEE Trans. Intell. Veh. **1**(1), 33–55 (2016)
45. Paszke, A., et al.: Pytorch: an imperative style, high-performance deep learning library. Adv. Neural Inf. Process. Syst. **32**, 8026–8037 (2019)
46. Pivtoraiko, M., Knepper, R.A., Kelly, A.: Differentially constrained mobile robot motion planning in state lattices. J. Field Robot. **26**(3), 308–333 (2009)
47. Qian, X., Altché, F., Bender, P., Stiller, C., de La Fortelle, A.: Optimal trajectory planning for autonomous driving integrating logical constraints: an miqp perspective. In: 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), pp. 205–210. IEEE (2016)
48. Rasmussen, M., Rieger, J., Webster, K.N.: Approximation of reachable sets using optimal control and support vector machines. J. Comput. Appl. Math. **311**, 68–83 (2017)
49. Robbins, H., Siegmund, D.: A convergence theorem for non negative almost supermartingales and some applications. In: Optimizing Methods in Statistics, pp. 233–257. Elsevier (1971)
50. Shkolnik, A., Walter, M., Tedrake, R.: Reachability-guided sampling for planning under differential constraints. In 2009 IEEE International Conference on Robotics and Automation, pp. 2859–2865. IEEE (2009)
51. Thorpe, A.J., Ortiz, K.R., Oishi, M.M.K.: Data-driven stochastic reachability using hilbert space embeddings. arXiv preprint arXiv:2010.08036 (2020)
52. Toth, C.D., O'Rourke, J., Goodman, J.E.: Handbook of Discrete and Computational Geometry. CRC Press (2017)
53. Tran, H.D., et al.: NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 3–17. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_1
54. Vincent, J.A., Schwager, M.: Reachable polyhedral marching (rpm): A safety verification algorithm for robotic systems with deep neural network components. arXiv preprint arXiv:2011.11609 (2020)
55. Xiang, W., Tran, H.D., Johnson, T.T.: Output reachable set estimation and verification for multilayer neural networks. IEEE Trans. Neural Netw. Learn. Syst. **29**(11), 5777–5783 (2018)
56. Xue, B., Zhang, M., Easwaran, A., Li, Q.: PAC model checking of black-box continuous-time dynamical systems. IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. **39**(11), 3944–3955 (2020)

57. Yang, X., Tran, H.-D., Xiang, W., Johnson, T.: Reachability analysis for feed-forward neural networks using face lattices. arXiv preprint arXiv:2003.01226 (2020)
58. Zeng, W.: End-to-end interpretable neural motion planner. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 8660–8669 (2019)