2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Learning Neural Control Barrier Functions from Expert Demonstrations using Inverse Constraint Learning

Yuxuan Yang[1] and Hussein Sibai[1]

***Abstract*— Safety is a fundamental requirement for autonomous systems operating in critical domains. Control barrier functions (CBFs) have been used to design safety filters that minimally alter nominal controls for such systems to maintain their safety. Learning neural CBFs has been proposed as a data-driven alternative for their computationally expensive optimization-based synthesis. However, it is often the case that the failure set of states that should be avoided is non-obvious or hard to specify formally, e.g., tailgating in autonomous driving, while a set of expert demonstrations that achieve the task and avoid the failure set is easier to generate. We use inverse constraint learning (ICL) to train a constraint function that classifies the states of the system under consideration to safe, i.e., belong to a controlled forward invariant set that is disjoint from the unspecified failure set, and unsafe ones, i.e., belong to the complement of that set. We then use that function to label a new set of simulated trajectories to train our neural CBF. We empirically evaluate our approach in four different environments, demonstrating that it outperforms existing baselines and achieves comparable performance to a neural CBF trained with the same data but annotated with ground-truth safety labels.**

I. Introduction

Designing safe control policies for autonomous systems has been an ongoing challenge that limits their deployment in critical settings [1], [2], [3]. One approach for maintaining safety is the design of safety filters which adjust safety-agnostic nominal controls online when they are potentially safety-violating [4]. Control Barrier Functions (CBFs) define sets of states that can be kept invariant by choosing controls that satisfy linear inequalities, allowing the design for efficient safety filters in the form of quadratic programs [4], [2]. They have been shown to be useful in maintaining safety in a variety of robotic applications, including bipedal robotic walking [5], adaptive cruise control [6], and space shuttles' docking [7].

When a set of *failure* (or *avoid*) states and the dynamics are known, CBFs can be synthesized using techniques such as Sum-of-Squares (SoS) [8], [9] and Hamilton-Jacobi reachability analysis [10], [11], [12]. However, such correct-by-construction methods suffer from the curse-of-dimensionality. Supervised deep learning approaches have been proposed to train neural CBFs, which are easier to design and more scalable, at the expense of losing correctness guarantees [1], [13], [14], [15]. Such approaches require labeled datasets of safe and unsafe states, and obtaining such labels is often not trivial because of non-obvious controlled forward invariant sets that are disjoint from failure sets, e.g., in the cases of systems with input constraints [16], systems with complex observation-based failure sets that are hard to specify mathematically [17], or unknown failure states that the expert generating the demonstrations is implicitly avoiding.

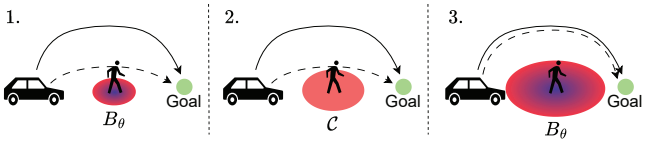


Fig. 1: The framework for training an ICL-CBF B_θ, where the solid arrows represent the expert trajectories and the dashed ones represent the ones generated using a CBF-QP policy consisting of π_{ref} and B_θ. $\mathcal{C}$ is the constraint represented by the learned constraint function $\hat{c}_\phi$. First, we train B_θ on trajectory data sampled with π_{ref} and labeled with $\hat{c}_\phi$, and then simulate the system using it to filter out unsafe actions. Then, we compute the difference between the sampled trajectories and their corresponding expert ones and update $\hat{c}_\phi$ accordingly. Third, we retrain B_θ using a new set of sampled trajectories that are labeled using the new $\hat{c}_\phi$, simulate again, return to the second step, and repeat until convergence.

We consider the setting where a safety-agnostic reference controller is available along with the system dynamics and a set of expert demonstrations that follow the reference controller while maintaining safety. Inverse constraint reinforcement learning (ICRL) [18], [19] or inverse constraint learning (ICL) [20], [21], [22] consider the similar setting of a known reward function along with a simulator and a set of expert demonstrations. Corresponding algorithms infer constraint functions that explain the deviation of the expert trajectories from reward-maximizing behavior. We adopt an ICL approach to learn a constraint function that we use to label a set of sampled trajectories which we then use to train a neural CBF. We call it an ICL-CBF.

We use a variation of MT-ICL [20], an algorithm that uses expert demonstrations from multiple tasks to learn a tight common constraint. Our proposed algorithm iteratively samples a set of trajectories following the reference controller, labels them using the constraint function being trained, trains a neural CBF using the labeled data, samples a new set of trajectories that follow the reference controllers while using a quadratic program based on the trained CBF as a safety filter, and retrains the constraint function to distinguish the sampled trajectories from the expert ones. We visualize the procedure in Figure 1. We also propose a simple heuristic to accelerate training by postponing the training of the neural CBF until the last iteration. At that iteration, the constraint function would

[1]Department of Computer Science and Engineering, Washington University in St. Louis, MO 63130, USA. { y.yuxuan, sibai}@wustl.edu

have converged and is able to provide more accurate labels. We evaluate the ability of learned ICL-CBFs to maintain safety while minimally deviating from the reference controllers in four examples. We show that they outperform two baselines and achieve comparable results to those obtained when the ground-truth safety labels are available.

II. Related work

a) Learning CBFs from expert demonstrations: Few algorithms have been proposed for learning neural CBFs from expert demonstrations [23], [24], [25]. Robey et al. [23] introduced a method to learn neural CBFs solely from expert demonstrations by constructing a safe region encompassing their states, uniformly sampling states around the region's boundary and labeling them as unsafe, and training the neural CBF with these sampled states along with the expert safe states. However, uniform sampling near the boundary region is often inefficient, especially in high-dimensional state spaces. In their follow-up work ROCBF [24], the authors employ the reverse-KNN algorithm to detect the boundary of the safe region and consider the states at that boundary as unsafe, thus avoiding sampling more states. Castañeda et al. [25] proposed training neural CBFs using expert demonstrations to avoid distribution shift calling them in-Distribution Barrier Functions (iDBFs). They train a Gaussian behavior cloning policy using the user-provided demonstrations, then sample actions with low probability under that policy at each state in the demonstrations. At each such state and for each sampled action, they simulate the system and consider the resulting state as out-of-distribution, or equivalently from the view of the neural CBF training, unsafe. Instead, we exploit the knowledge of the reference controller to determine the states that were avoided by the expert due to safety violations, and consider them to be unsafe. Farrel et al. [26] recently proposed an approach to learn neural CBFs from an offline dataset that is a mix of labeled and unlabeled data, without further sampling. It utilizes out-of-distribution detectors for classifying the unlabeled data based on the labeled ones.

b) Inverse constraint learning: Inverse Constraint Learning (ICL) was initially proposed in [18] as an analogue to Inverse Reinforcement Learning (IRL) [27] for constraint inference, but it was limited to discrete settings. Malik et al. [19] proposed ICRL that generalizes ICL to continuous state spaces and model-free settings by learning constraint functions. Kim et al. [20] proposed MT-ICL and demonstrated that ICL can recover tighter constraints in the multi-task setting when the dynamics and the constraints are shared among the tasks. Building on MT-ICL, Qadri et al. [28] proved that when the expert follows a maximum entropy policy or generates trajectories accomplishing multiple tasks sharing the same constraint, the constraint function inferred by an ideal ICL (or MT-ICL) algorithm defines the Backward Reachable Set (BRS) corresponding to some unknown failure set, i.e., the set of states that for which there is no control policy that can prevent the system from eventually reaching the failure set in the worst case [29]. The complement of such BRS is the maximum controlled forward invariant set [30]. Thus, the constraint function obtained by an ideal ICL is a perfect classifier of safe and unsafe states. Based on a learned version of this classifier, we label a new set of sampled trajectories to train our neural CBFs.

III. Preliminaries

We consider nonlinear control-affine systems of the form:

$$\dot{x} = f(x) + g(x)u, \tag{1}$$

where $x(t) \in \mathcal{X} \subseteq \mathbb{R}^n$ is the system's state and $u(t) \in \mathcal{U} \subseteq \mathbb{R}^m$ is the control input at time t. We assume that the action space $\mathcal{U}$ is polyhedral, $f : \mathcal{X} \to \mathbb{R}^n$ and $g : \mathcal{X} \to \mathbb{R}^{n\times m}$ are locally Lipschitz continuous and that system (1) is forward complete.

A. Control Barrier Functions

A CBF for system (1) is defined as follows.

Definition 1 (Zeroing control barrier functions [4])**.** *A continuously differentiable function $B : \mathcal{D} \to \mathbb{R}$ is a* zeroing control barrier function *for system (1) if there exists an extended class-$\mathcal{K}_\infty$ function $\alpha : \mathbb{R} \to \mathbb{R}$, i.e., continuous, strictly increasing, and $\alpha(0) = 0$, such that*

$$\exists u \in \mathcal{U} \text{ s.t. } \dot{B}(x,u) + \alpha(B(x)) \geq 0 \quad \forall x \in \mathcal{D}, \tag{2}$$

where $\dot{B}(x,u) = \nabla B(x)(f(x)+g(x)u)$ is the time derivative of B along the trajectories of (1), and $\mathcal{D} = B_{\geq c} := \{x \mid B(x) \geq -c\}$, for some $c > 0$.

The super-level set $B_{\geq 0}$ can be made *forward invariant* by following a policy that satisfies (2).

Theorem 1 ([4])**.** *Fix a CBF B for system (1) where $\nabla B(x) \neq 0$ for all $x \in B_{=0} := \{x \mid B(x) = 0\}$. Then, any Lipshitz continuous control policy $\pi : \mathcal{X} \to \mathcal{U}$ that satisfies*

$$\pi(x) \in \{u \in \mathcal{U} : \nabla B(x)(f(x)+g(x)u) + \alpha(B(x)) \geq 0\} \tag{3}$$

will result in $B_{\geq 0}$ being a forward invariant set*, i.e., any trajectory of (1) starting in $B_{\geq 0}$ remains inside it $\forall t \geq 0$.*

Then, given a reference controller $\pi_{ref} : \mathcal{D} \to \mathcal{U}$ and a failure set of states that is disjoint from $B_{\geq 0}$, a CBF B for system (1) can be used to design a *safety filter* that corrects π_{ref}'s decisions when violating the invariance of $B_{\geq 0}$, and thus maintaining safety. The safety filter can be constructed by formulating a quadratic program (QP) with the objective of finding a control input that is minimally distant from the reference one while belonging to the set defined in (3) that guarantees the forward invariance of $B_{\geq 0}$ as follows [4]:

$$\begin{aligned} \pi_{safe}(x) := \arg\min_{u\in\mathcal{U}} \|u - \pi_{ref}(x)\|, \\ s.t.\ \nabla B(x)(f(x)+g(x)u) + \alpha(B(x)) \geq 0. \end{aligned} \tag{4}$$

We call π_{safe} a CBF-QP policy.

B. Inverse constraint learning

Compared to Inverse Reinforcement Learning (IRL), which aims to recover the reward function optimized for by an expert policy $\pi_E : \mathcal{D} \to \mathcal{U}$ in unconstrained settings, Inverse Constraint Learning (ICL) aims to recover the constraint

that an expert policy in a constrained setting is satisfying while maximizing a known reward function given a set of trajectories generated by the expert policy. In this paper, we adopt the approach of Multi-Task ICL (MT-ICL) [20]. In MT-ICL, the constraint of a single task is derived as the solution of the following constrained optimization problem:

$$\min_{\pi\in\Pi} J(\pi_E, r) - J(\pi, r) \text{ s.t. } \max_{c\in\mathcal{C}} J(\pi, c) - J(\pi_E, c) \leq 0, \quad (5)$$

where $J(\pi, f)$ represents the value of policy π under the measurement function f, r is the known reward function of states, $\mathcal{C}$ is a set of constraint functions of states, and Π is a set of policies. The optimization process comprises an outer part, which optimizes the constraint $\hat{c}$ as a classifier to maximally distinguish between the states visited by the learned policy $\hat{\pi}_E$ and those visited by the expert policy π_E:

$$\hat{c} = \arg\max_{c\in\mathcal{C}} J(\hat{\pi}_E, c) - J(\pi_E, c). \quad (6)$$

The inner part trains a policy $\hat{\pi}_E$ to maximize the reward given the current learned constraint $\hat{c}$:

$$\hat{\pi}_E = \max_{\pi\in\Pi} J(\pi, r), \quad \text{s.t. } J(\pi, \hat{c}) \leq \kappa, \quad (7)$$

for some user-defined cost threshold κ. The process can be seen as training a constraint that results in the same trajectories as the ones generated by a safe RL policy that maximizes the known reward while satisfying that constraint.

Similarly, the constraint of K tasks can be derived as the solution of the following optimization problem:

$$\begin{aligned} &\min_{\pi^{1:K}\in\Pi} \sum_{i}^{K} J(\pi_E^i, r^i) - J(\pi^i, r^i), \\ &\text{s.t. } \max_{c\in\mathcal{C}} \sum_{i}^{K} J(\pi^i, c) - J(\pi_E^i, c) \leq 0, \end{aligned} \quad (8)$$

where π_E^i, π^i, and r^i are the expert policy, policy, and the reward function of task i. This version can be solved using the same strategy discussed above.

IV. Training neural CBFs using ICL

Qadri et al. [28] proved that the constraint set, i.e., the set of states that are constraint-violating, derived through a single iteration of an *exact entropy-regularized* or multi-task ICL algorithm is the *backward reachable set (BRS)* corresponding to an unknown failure set. The complement of the BRS is the maximal controlled forward invariant set [31], [30].

In this paper, we adapt the ICL procedure to train a neural CBF from expert trajectories. We approximate the expert policy by a CBF-QP policy with a known reference controller and a learned neural CBF. The expert does not need to be a CBF-QP policy. It can be a human or another safe task-achieving policy as long as the provided demonstrations sufficiently cover the state space besides the true constraint set. If the expert deviates from the provided reference controller within the safe set (the complement of the true constraint set), states visited by the CBF-QP policy with an ideal CBF B^* whose sublevel set $B^*_{\geq 0}$ is the true constraint set, but not by the expert demonstrations, would be learned by ICL as part of the constraint set. Thus, the learned constraint set would be a conservative estimate of the true one.

A. Proposed algorithm

Given a learned neural CBF B_θ with parameters θ and the ground-truth reference controller π_{ref}, we define a corresponding CBF-QP policy $\hat{\pi}_E$, per (4), as an approximation of the expert policy π_E. In contrast with (7), we do not need to use a safe RL algorithm to obtain $\hat{\pi}_E$. Instead, we run a QP solver online to get the policy. On the other hand, as in MT-ICL, we sample trajectories using $\hat{\pi}_E$ and solve a similar optimization problem to (6) to find a constraint $\hat{c}_\phi$ that can be used to train B_θ.

Given a set of states visited by the expert trajectories $\mathcal{X}_E$ and a set of states visited by the sampled trajectories $\mathcal{X}_S^c$ using $\hat{\pi}_E$, we train a neural constraint function $\hat{c}_\phi$ parameterized with ϕ by minimizing the following loss:

$$\mathcal{L}_{\hat{c}_\phi} := \sum_{x\in\mathcal{X}_S^c} \|1 - \hat{c}_\phi(x)\|^2 + \sum_{x\in\mathcal{X}_E} \|1 + \hat{c}_\phi(x)\|^2. \quad (9)$$

After that, we sample a new set of trajectories starting from random initial states using π_{ref}. We then partition the set of states in these trajectories, denoted by $\mathcal{X}_S^B$, to safe and unsafe ones: $\mathcal{X}_{safe} = \{x \in \mathcal{X}_S^B \mid \hat{c}(x) < \delta\}$ and $\mathcal{X}_{unsafe} = \{x \in \mathcal{X}_S^B \mid \hat{c}(x) \geq \delta\}$, where $\delta \in \mathbb{R}$ is a hyperparameter. Additionally, we define the set of safe state-action pairs, i.e., the ones where the state is in $\mathcal{X}_{safe}$ and the pair also result in a state in $\mathcal{X}_{safe}$, in the sampled trajectories by $\mathcal{D}_{safe}$. Once we obtain $\mathcal{X}_{safe}$, $\mathcal{X}_{unsafe}$, and $\mathcal{D}_{safe}$, we train a neural CBF B_θ by minimizing the following loss:

$$\begin{aligned} \mathcal{L}_{B_\theta} := {} & w_{safe} \sum_{x_{safe}\in\mathcal{X}_{safe}} \sigma(\varepsilon_{safe} - B_\theta(x_{safe})) + \\ & w_{unsafe} \sum_{x_{unsafe}\in\mathcal{X}_{unsafe}} \sigma(\varepsilon_{unsafe} + B_\theta(x_{unsafe})) \\ & + w_{ascent} \sum_{(x_{safe},u_{safe})\in\mathcal{D}_{safe}} \sigma(\varepsilon_{ascent} - \nabla B_\theta(x_{safe}) \\ & (f(x_{safe}) + g(x_{safe})u_{safe}) - \alpha \cdot B_\theta(x_{safe})), \end{aligned} \quad (10)$$

where $\sigma(\cdot)$ is the ReLU function and α, w_{safe}, w_{unsafe}, w_{ascent}, ε_{safe}, ε_{unsafe}, and ε_{ascent} are non-negative hyperparameters. The training procedure is summarized in Algorithm 1.

Remark 1. *Algorithm 1 can be extended straightforwardly to accept multi-task expert demonstrations and corresponding reference controllers as input as follows: in line 3, it generates a union $\mathcal{X}_S^B$ of sets $\{\mathcal{X}_S^{B,i}\}$ by sampling trajectories using π_{ref}^i for each task i; then in line 8, it generates a union $\mathcal{X}_S^c$ of sets $\{\mathcal{X}_S^{c,i}\}$, one for each task i using its corresponding CBF-QP policy $\hat{\pi}_E^i$ resulting from π_{ref}^i and B_θ; we train B_θ with $\{\mathcal{X}_S^B\}$ in line 5 and update $\hat{c}_\phi$ using $\{\mathcal{X}_S^c\}$ in line 9.*

B. Heuristic for approximating $\hat{\pi}_E$ during training

Retraining the neural CBF B_θ every iteration of Algorithm 1 can be computationally expensive. When the action space is low-dimensional, it might be more efficient to use a grid search to find a safe control during training. For that reason, in our experiments, in the first $N-1$ iterations, we grid the

Algorithm 1: Training ICL-CBFs

Input: reference controller π_{ref}, a set of expert demonstrations $\{\tau_E\}$, and system dynamics.

Initialize $\hat{c}_\phi$ and B_θ as zero functions.
for i *in* $1,\dots,N$ **do**
  Generate $\mathcal{X}_S^B$ by sampling trajectories using π_{ref} starting from random initial states.
  Construct $\mathcal{X}_{safe}$, $\mathcal{X}_{unsafe}$, and $\mathcal{D}_{safe}$ by labeling the sampled trajectories using $\hat{c}_\phi$.
  Train B_θ according to (10).
  if $i = N$ **then**
    break
  Generate $\mathcal{X}_S^c$ by sampling trajectories using $\hat{\pi}_E$ starting from random initial states.
  Update $\hat{c}_\phi(x)$ using and according to (9).

Output: neural control barrier function B_θ.

control space over which we search for a control input that does not violate the constraint $\hat{c}_\phi$ to generate $\hat{\pi}_E$, and only train B_θ in the last iteration.

To generate $\hat{\pi}_E$, we select a ball centered at the origin in the control space $\mathcal{U}$ and grid it according to a user-defined resolution. We also select a sampling time t_s. Then, we define $\hat{\pi}_E$ to be the policy that samples the state x every t_s seconds. At each sampling instant, $\hat{\pi}_E$ iterates over the cells of the grid. For each cell, it computes the distance of its center to $\pi_{ref}(x)$ at the current state. Then, it simulates the system by fixing the control signal to be a constant one that is equal to the center of the cell for t_s seconds. Finally, it selects the closest center to $\pi_{ref}(x)$ that does not lead to a state x' that violates the learned constraint $\hat{c}_\phi$ after t_s seconds, i.e., the case where $\hat{c}_\phi(x') \geq \delta$.

This heuristic allows skipping sampling a new set of trajectories, labeling them using $\hat{c}_\phi$, and training the CBF B_θ at each iteration in Algorithm 1. Once the constraint converges, we can use it to train a neural CBF that can be used to filter unsafe actions during deployment. The neural CBF would then allow the design of the CBF-QP policy that is more computationally efficient to run in real-time. Note that this heuristic is only useful for systems with low-dimensional action spaces. As the action space dimension increases, the original algorithm becomes more efficient.

V. Experimental results

We evaluated our method in four scenarios aiming to address the following research questions (RQs):

- **RQ1:** Do ICL-CBF-based safety filters improve safety while minimally affecting task success?
- **RQ2:** Does the heuristic algorithm for approximating $\hat{\pi}_E$ degrade the quality of B_θ?
- **RQ3:** Should we generate $\mathcal{X}_S^B$ using π_{ref} or $\hat{\pi}_E$?
- **RQ4:** Are the safety labels produced using the learned constraint accurate?
- **RQ5:** How sensitive is the performance of the learned ICL-CBF to the choice of the hyperparameter δ?

A. Setup

The scenarios we consider are: **Single integrator**: driving a simple 2D robot with single integrator dynamics to reach a goal position while avoiding a circular obstacle; **Dubins car**: same as the first scenario but with Dubins car dynamics [32]; **Inverted pendulum [33]**: rotating an inverted pendulum according to a reference controller while avoiding an unsafe set of states; **Quadrotor [34]**: navigating a quadrotor towards a goal position while avoiding colliding with the ground.

We compare our results with those obtained using the ground-truth CBFs (GT) (which we used to generate the expert demonstrations), the neural CBFs trained using ROCBF [24], and the neural CBFs trained using iDBF [25]. We also compare them with those obtained using the neural CBFs trained using a set of sampled trajectories using the reference controllers whose states are labeled as safe or unsafe based on the ground-truth CBF, which we denote by L-CBF (L stands for "labeled"). For all scenarios, we use multi-layer perceptrons (MLPs) to represent the constraint functions and the neural CBFs. We choose the hyperparameter δ to be 0.6 for the single integrator and the quadrotor, 0.4 for the Dubins car, and 0.3 for the inverted pendulum.

B. Implementation Details

We designed CBFs and reference controllers to generate the expert demonstrations. However, generally, the expert policy need not be a CBF-QP policy, as we mentioned earlier.

1) Single integrator: This scenario consists of a 2D robot with single integrator dynamics reaching a specified goal position while avoiding a circular obstacle located at the origin. Given the state $\mathrm{x} = [x, y]^\top$, the robot's dynamics are defined as $\dot{\mathrm{x}} = u = [v_x, v_y]^\top$. The failure set of states is defined as the circle centered at the origin with radius r, i.e., $\{\mathrm{x} \in \mathbb{R}^2 \mid \|\mathrm{x}\|_2 < r\}$, where $r = 1$ is the radius of the obstacle. We sample initial states uniformly at random from the square $[-6,6] \times [-6,6]$ while rejecting those with $\|\mathrm{x}\|_2 < 3$. For each initial state $\mathrm{x}_0 = [x_0, y_0]^\top$, we set the goal state to be $\mathrm{x}_g = [-x_0 + b_x, -y_0 + b_y]^\top$, where b_x, b_y are sampled uniformly at random from the interval $[-1,1]$. We define the reference controller for the goal x_g to be $\pi_{ref}(\mathrm{x}, \mathrm{x}_g) := \frac{\mathrm{x}_g - \mathrm{x}}{\|\mathrm{x}_g - \mathrm{x}\|_2}$. Such a choice of initial and goal states drives the agent to pass close to the origin since they are at opposing ends. Thus, the agent will enter the failure set when following the reference controller if not prevented using a safety filter. To collect expert demonstrations, we used a CBF-QP controller with the same reference controller and the CBF $B_{GT}(\mathrm{x}) = \|\mathrm{x}\|_2 - r$. We considered the agent to have reached the goal when it is within 0.1 distance. The maximum number of time steps in each trajectory is 300 at which it is terminated if it did not reach the goal. We collected 150 trajectories with different initial and goal states for training. The sampling time is set to 0.1 seconds for training and evaluation. For the grid-based $\hat{\pi}_E$, we constructed a 50×50 grid over the square $[-1,1] \times [-1,1]$ in the control space.

2) Inverted pendulum: We followed the same setting used in [8], where the state is $\mathrm{x} = [x, y]$ and the dynamics are

defined as follows: $\dot{\mathrm{x}} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \mathrm{x} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u$. The failure set is the complement of the set $[-0.1, 0.15] \times [-0.3, 0.25]$, the reference controller is $\pi_{ref}(\mathrm{x}) = -K\mathrm{x}$ where $K = [3, 3]$, and the CBF used in the CBF-QP policy that we used to generate the expert demonstrations is $B_{GT}(\mathrm{x}) = k - (\mathrm{x} - \mathrm{x}_g)^\top P(\mathrm{x} - \mathrm{x}_g)$, where $P = \begin{pmatrix} 1.25 & 0.25 \\ 0.25 & 0.25 \end{pmatrix}$, $k = 0.01$, and $\mathrm{x}_g = [0, 0]$. We sampled initial states uniformly at random from the rectangle $[-0.103, 0.148] \times [-0.3, 0.25]$. We considered the agent to have reached the goal (origin) when its norm is less than or equal to 0.1. The maximum number of time steps in each trajectory is 300 at which it is terminated if it did not reach the goal. The sampling time is 0.1 seconds. For the grid-based $\hat{\pi}_E$, we split the interval $[-5, 5]$ into 2500 equal intervals.

3) Dubins car: We followed a similar setting used in [34], where the state is $\mathrm{x} = [x, y, \phi]^\top$ and the system dynamics are: $\dot{\mathrm{x}} = \begin{pmatrix} v\cos(\phi) \\ v\sin(\phi) \\ u \end{pmatrix}$, where $v = 1$ is the fixed speed of the car. The task is to drive the car starting from initial states sampled uniformly at random from the square with sides equal to 0.4 and centered at $(-3, -3)$ for the 2D position and the range $[-0.4\pi, 0.4\pi]$ for the heading angle to the goal position $(x_g, y_g) = (3, 3.5)$, while avoiding the failure set defined by the square centered at the origin with sides equals to 2. We used $\pi_{ref}(\mathrm{x}) := \psi - \phi$ as the reference controller, where $\psi = \arctan(y_g - y, x_g - x)$. We trained a neural CBF using the method proposed in [34] to generate 100 expert trajectories with different initial states. We considered the car to have reached the goal when it is within 0.1 distance. The maximum number of time steps in each trajectory is 200 at which it is terminated if it did not reach the goal. The sampling time is 0.1 seconds. For the grid-based $\hat{\pi}_E$, we split the interval $[-1, 1]$ into 50 equal intervals.

4) Quadrotor: We considered a self-righting 6-state planar quadrotor model used in [34] with a state $\mathrm{x} = [x, v_x, y, v_y, \phi, \rho]^\top$ and dynamics:

$$\dot{\mathrm{x}} = \begin{pmatrix} v_x \\ \frac{-C_D^v v_x}{m} \\ v_y \\ \frac{-C_D^v v_y}{m} - g \\ \rho \\ \frac{-C_D^\phi \rho}{I_{yy}} \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ \frac{-\sin(\phi)}{m} & \frac{-\sin(\phi)}{m} \\ 0 & 0 \\ \frac{\cos(\phi)}{m} & \frac{\cos(\phi)}{m} \\ 0 & 0 \\ -\frac{l}{I_{yy}} & -\frac{l}{I_{yy}} \end{pmatrix} u. \qquad (11)$$

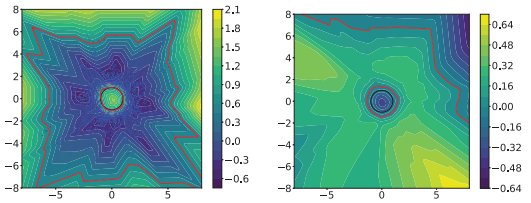
The task is to drive the quadrotor with initial states of the form $\mathrm{x} = [0, 1, 2 + b_y, 0, -\frac{\pi}{2.5} + b_\phi, 0]$, where b_y and b_ϕ are sampled uniformly at random from the ranges $[-0.1, 0.1]$ and $[-0.05, 0.05]$, respectively, to the goal position (6,9), while avoiding the failure set which consists of all the states with $y = 0$, representing the ground. We used the LQR-based controller considered in [34] as the reference controller π_{ref}. We trained a neural CBF using the method proposed in [34] to generate the expert trajectories. We collected 60 trajectories with different initial states for training. We considered the quadrotor to have reached the goal when its within 0.1 distance. The maximum number of time steps in each trajectory is 300 at which it is terminated if it did not reach the goal. The sampling time is set to 0.05 seconds. For the grid-based $\hat{\pi}_E$, we constructed a 100×100 grid over the square $[0, 20] \times [0, 20]$ in the control space.

C. RQ1: Do ICL-CBF-based safety filters improve safety while minimally affecting task success?

We first subjectively analyze the quality of the constraints and the neural CBFs learned using our method for the single integrator case before analyzing their closed-loop performances. We visualize them in Figure 2. In Figure 2a, we can see that the obstacle is correctly identified as part of the constraint set by having corresponding learned constraint function's values greater than δ. Note that because the system can stop at any state by choosing a zero control input, the backward reachable set is equal to the failure set, i.e., the states in the obstacle. That makes a simple distance-based function a valid CBF, which is the one we used to generate the expert trajectories, as we explained in Section V-B. In our case, since the expert trajectories are concentrated around the obstacle, the region of the state space further away is also considered constraint-violating by the learned constraint, as expected. It is evident from Figure 2c that iDBF did not accurately recover the failure set. That is likely because of its strategy for sampling states to label as out-of-distribution (or equivalently, unsafe), which in low-dimensional state spaces and in the presence of dense set of expert trajectories result in overlapping regions labeled both safe and unsafe, significantly affecting the quality of the learned constraint. Although ROCBF successfully delineates the failure set, part of the region visited by the expert trajectories is misclassified as unsafe, as shown in Figure 2d. In contrast, ICL-CBF and L-CBF, as shown in Figures 2b and 2e, respectively, are very close to each other implying that training a neural CBF from the labels generated by a learned constraint using ICL is comparable to training it using the true labels.

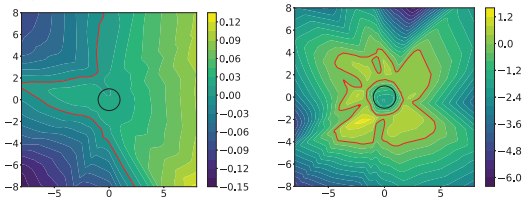
Second, we compare the closed-loop performance of the neural CBFs trained using the different methods based on the collision rates (CR) and success rates (SR) when used as part of the QP-based safety filters. The results are shown in Table I. ICL-CBF achieves the best performance in terms of both metrics in all considered scenarios. For the single integrator one, although ROCBF results in a zero CR, it results in a low SR due to its conservativeness. Compared to L-CBF, ICL-CBF results in a 5.6% decrease in SR in the single integrator scenario and a 20.8% decrease in SR in the quadrotor one. For the inverted pendulum and Dubins car scenarios, ICL-CBF achieves comparable performance to L-CBF. In contrast, iDBF results in high CR in both tasks.

Finally, we visualize a set of trajectories of the inverted pendulum generated using the different safety filters in Figure 3. We sampled a set of initial states for which the reference controller would result in a collision with the absence of a safety filter. The trajectories generated using iDBF and ROCBF reach the goal but enter the failure set. In contrast, both ICL-CBF and L-CBF were able to successfully intervene



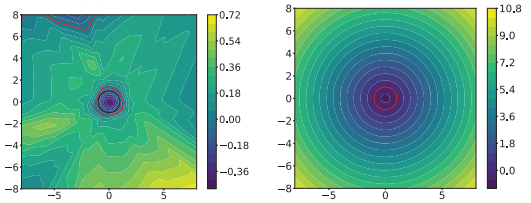
(a) Learned constraint

(b) ICL-CBF



(c) iDBF

(d) ROCBF



(e) L-CBF

(f) Ground truth

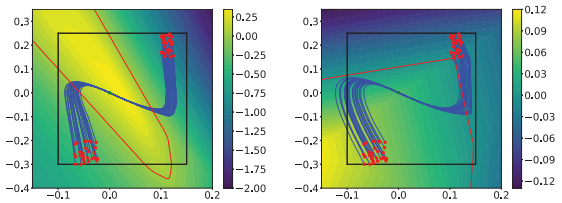
Fig. 2: Values of the learned constraint and neural CBFs in the single-integrator scenario, where the black circles in the middle of the figures represent the obstacle. For (a), we color the boundary of the set $\{x \in \mathbb{R}^2 | \hat{c}_\phi(x) = \delta\}$ in red and the blue points are randomly sampled states from the expert demonstrations. For (b)-(f), the red margins denote the zero level set of each (neural) CBF.

to prevent the system from entering the failure set and attract it towards their sublevel sets which contain the goal.

D. RQ2: Does the heuristic algorithm for approximating $\hat{\pi}_E$ degrade the quality of B_θ?

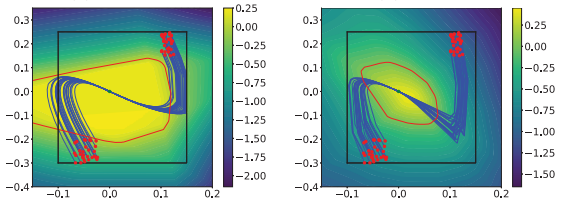
We evaluated the performance of ICL-CBF on the single integrator and quadrotor scenarios with and without using the heuristic for approximating $\hat{\pi}_E$ during training. As shown in Table II, training the ICL-CBF is significantly slower when not using the heuristic. The heuristic results in more noticeable degradation of CR and SR in the quadrotor scenario than the single integrator one. This is likely due to the higher dimensional state space of the quadrotor that requires finer resolution when choosing controls, and that is achieved using quadratic programming better than a brute-force search over coarse grids. The degradation might be an acceptable cost for the gains in training time for small enough dimensional systems. As the required grid size increases, the heuristic becomes slower and less useful.

Moreover, we compared the inference times of the generated CBF-QP policies with those of grid-based policies. They follow the same logic as the ones obtained using the heuristic. They use the learned constraint directly without training the ICL-CBF in the last iteration. We tried using grids for inference with two different resolutions. The first is the same as the one used for training the constraint and the second is a larger one. We generated five trajectories per scenario, each with 1000 time steps using both policies and computed the average trajectory-wise total inference time. The results are shown also in Table II. They show that although using the grid-based policy with a small grid size might decrease the inference time, it often comes at the cost of worse performance, while an ICL-CBF and its corresponding CBF-QP policy significantly outperform them while having small enough inference time, especially in higher dimensions.



(a) ICL-CBF

(b) iDBF



(c) ROCBF

(d) L-CBF

Fig. 3: The neural CBFs learned with different methods along with trajectories of the inverted pendulum starting from sampled initial states and following corresponding CBF-QP policies. The red points are initial states, the green points (at the origin) are the goal states, and the red contours represent the zero level set of each neural CBF.

E. RQ3: Should we generate $\mathcal{X}_S^B$ using π_{ref} or $\hat{\pi}_E$?

Algorithm 1 constructs $\mathcal{X}_S^B$ using π_{ref}, resulting in safe and unsafe trajectories. Recall that if the heuristic for generating $\hat{\pi}_E$ is followed, $\mathcal{X}_S^B$ is only generated once in the N^{th} iteration. If it were to be generated using the heuristic-based $\hat{\pi}_E$ of the $(N-1)^{th}$ iteration, it will likely contain few unsafe states as $\hat{c}_\phi$ would have converged, and that would not result in a balanced $\mathcal{X}_S^B$ for training B_θ. Alternatively, one can construct $\mathcal{X}_S^B$ in the N^{th} iteration as the union of all $\mathcal{X}_S^c$ over all iterations, i.e., by aggregating all the trajectories sampled using $\hat{\pi}_E$ in all $N-1$ iterations. We compare this strategy with the one using π_{ref} in the single integrator scenario. The results, shown in Table III, demonstrate that the B_θ learned from data sampled with π_{ref} outperforms the one trained with the union of the $\mathcal{X}_S^c$ sets in terms of both CR and SR. We hypothesize that the reason is that while training $\hat{c}_\phi$, it is not sufficiently accurate. That results in the distribution of states in the union of the $\mathcal{X}_S^c$ sets different from the one encountered during deployment.

F. RQ4: Are the safety labels produced using the learned constraint accurate?

We subjectively analyze the accuracy of the labels used in the training of different neural CBFs in the inverted pendulum scenario. As discussed earlier, each training method labels the states visited by the expert trajectories and sampled ones, if any, differently. The results are shown in Figure

TABLE I: Success rates (SR) and collision rates (CR) of each method in different scenarios.

Task	Metric	iDBF	ROCBF	ICL-CBF	L-CBF
Single integrator	CR	99.20 ± 0.74	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	0.00 ± 0.00
	SR	0.80 ± 0.74	9.80 ± 2.14	$\mathbf{80.60 \pm 4.41}$	86.20 ± 1.94
Inverted pendulum	CR	2.80 ± 1.17	5.00 ± 1.90	$\mathbf{0.20 \pm 0.40}$	0.60 ± 0.80
	SR	97.20 ± 1.17	95.00 ± 1.90	$\mathbf{99.80 \pm 0.40}$	99.40 ± 0.80
Dubins car	CR	0.00 ± 0.00	69.30 ± 23.30	$\mathbf{1.80 \pm 1.33}$	0.30 ± 0.64
	SR	75.00 ± 25.00	6.40 ± 3.44	$\mathbf{97.60 \pm 1.50}$	99.60 ± 0.80
Quadrotor	CR	65.7 ± 22.14	75.00 ± 25.00	$\mathbf{17.10 \pm 6.64}$	1.50 ± 1.12
	SR	2.8 ± 0.98	0.00 ± 0.00	$\mathbf{77.20 \pm 4.31}$	98.00 ± 1.26

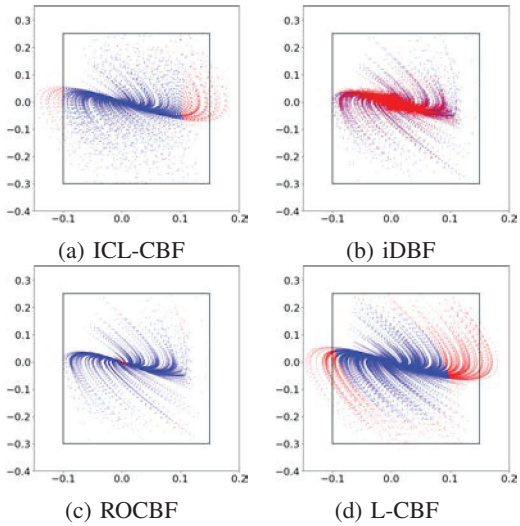
TABLE II: Success rates (SR) and collision rates (CR), training time, and inference time of different methods in the single integrator and quadrotor scenarios.

Task	Method	CR	SR	Training Time	Inference Time
Single integrator	ICL-CBF-based $\hat{\pi}_E$ (without heuristic)	$\mathbf{0.00 \pm 0.00}$	$\mathbf{81.80 \pm 2.32}$	578.11 ± 61.52	3.67 ± 0.18
	ICL-CBF-based $\hat{\pi}_E$ (with heuristic)	$\mathbf{0.00 \pm 0.00}$	80.60 ± 4.41	217.80 ± 2.40	3.77 ± 0.14
	Grid-based $\hat{\pi}_E$ (grid size of 50^2)	$\mathbf{0.00 \pm 0.00}$	79.40 ± 3.67	$\mathbf{189.80 \pm 2.42}$	$\mathbf{0.83 \pm 0.09}$
	Grid-based $\hat{\pi}_E$ (grid size of 250^2)	$\mathbf{0.00 \pm 0.00}$	80.40 ± 6.18	$\mathbf{189.80 \pm 2.42}$	4.52 ± 0.23
Quadrotor	ICL-CBF-based $\hat{\pi}_E$ (without heuristic)	$\mathbf{9.30 \pm 4.12}$	$\mathbf{87.60 \pm 3.44}$	2967.33 ± 148.97	4.15 ± 0.29
	ICL-CBF-based $\hat{\pi}_E$ (with heuristic)	17.10 ± 6.64	77.20 ± 4.31	2369.38 ± 270.49	4.27 ± 0.39
	Grid-based $\hat{\pi}_E$ (grid size of 100^2)	32.10 ± 11.18	57.20 ± 4.12	$\mathbf{2318.28 \pm 271.26}$	$\mathbf{3.52 \pm 0.12}$
	Grid-based $\hat{\pi}_E$ (grid size of 300^2)	24.20 ± 9.00	67.60 ± 4.96	$\mathbf{2318.28 \pm 271.26}$	29.70 ± 4.11

TABLE III: Success rates (SR) and collision rates (CR) of neural CBFs trained using data sampled using different policies in the single integrator scenario.

Data Source	Collision Rate	Success Rate
Sampled using π_{ref}	$\mathbf{0.00 \pm 0.00}$	$\mathbf{80.60 \pm 4.41}$
Union of $\mathcal{X}_S^c$'s	2.20 ± 1.17	78.60 ± 3.50

4, where the red points represent states labeled as unsafe and blue points represent states labeled as safe. The labels of the states generated by iDBF in this low-dimensional state space are inaccurate because the states labeled as unsafe are generated by applying actions to each expert state. When the expert states are close to each other, the states sampled and labeled as unsafe are close to the expert safe states. This issue should be a less of a concern in high-dimensional state spaces. As for the labels generated by ROCBF, only states at the boundary of the region where the expert trajectories reside are classified as unsafe, avoiding having overlapping states with different labels. However, states near the origin were considered boundary points by the reverse-KNN algorithm which lead ROCBF's method to mislabel them as unsafe. The constraint learned by MT-ICL, however, accurately classifies most states inside the failure set as well the states starting from which lead to failure states as unsafe. Its results are the closest to the true labels used by L-CBF shown in Figure 4d.



(a) ICL-CBF (b) iDBF (c) ROCBF (d) L-CBF

Fig. 4: Training data generated with different methods for inverted pendulum, where states within the black margins belong to the predefined safe set. Red points represent unsafe states, while blue points indicate safe states as annotated by each method.

G. RQ5: How sensitive is the performance of the learned ICL-CBF to the choice of the hyperparameter δ?

In our algorithm, δ serves as the threshold we use to partition the set $\mathcal{X}_S^B$. Because the constraint is trained on data produced by this partition, its performance is highly sensitive to the choice of δ. As shown in Figure 5, the performance of ICL-CBFs can deteriorate when δ is poorly chosen. Nevertheless, we found searching for a $\delta \in [0,1]$ results in one with reasonable performance. In practice, one can discretize the interval $[0,1]$ and perform a grid search to find the best one when evaluated at the validation data. In our experiments, $\delta = 0.6$ consistently produced strong results, although further tuning may yield additional gains.

VI. CONCLUSION

In this paper, we address the problem of training neural CBFs from safe expert demonstrations using inverse constraint learning. We use ICL to learn a constraint function and then use it to label the states in newly sampled trajectories as safe or unsafe. Our approach requires a set of expert demonstrations, a potentially unsafe task-achieving reference controller, and the system dynamics. We compare our method

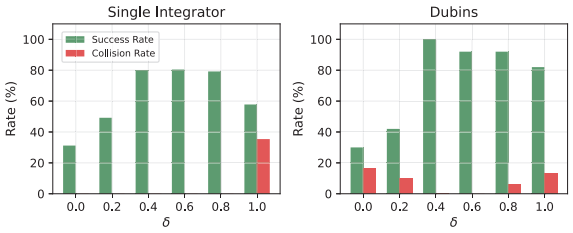


Fig. 5: Collision and success rates as δ varies in the Single integrator and Dubins car scenarios.

against two baseline algorithms as well as against neural CBFs trained using labeled data. Our empirical results validate the effectiveness of our method in generating neural CBFs that improve safety while minimally affecting performance. Our method does not currently guarantee the formal correctness of the learned CBF. We believe that formally verifying its correctness is a promising direction for future work [35].

REFERENCES

[1] C. Dawson, S. Gao, and C. Fan, "Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods for robotics and control," *Trans. Rob.*, vol. 39, no. 3, p. 1749–1767, Jun. 2023. [Online]. Available: https://doi.org/10.1109/TRO.2022.3232542

[2] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *2019 18th European Control Conference (ECC)*, 2019, pp. 3420–3431.

[3] K.-C. Hsu, H. Hu, and J. F. Fisac, "The safety filter: A unified view of safety-critical control in autonomous systems," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 7, no. Volume 7, 2024, pp. 47–72, 2024. [Online]. Available: https://www.annualreviews.org/content/journals/10.1146/annurev-control-071723-102940

[4] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs for safety critical systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 8, pp. 3861–3876, 2017.

[5] S.-C. Hsu, X. Xu, and A. D. Ames, "Control barrier function based quadratic programs with application to bipedal robotic walking," in *2015 American Control Conference (ACC)*, 2015, pp. 4542–4548.

[6] A. D. Ames, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs with application to adaptive cruise control," in *53rd IEEE conference on decision and control*. IEEE, 2014, pp. 6271–6278.

[7] J. Breeden and D. Panagou, "Guaranteed safe spacecraft docking with control barrier functions," *IEEE Control Systems Letters*, vol. 6, pp. 2000–2005, 2021.

[8] A. Clark, "Verification and synthesis of control barrier functions," in *2021 60th IEEE Conference on Decision and Control (CDC)*, 2021, pp. 6105–6112.

[9] L. Wang, D. Han, and M. Egerstedt, "Permissive barrier certificates for safe stabilization using sum-of-squares," *2018 Annual American Control Conference (ACC)*, pp. 585–590, 2018.

[10] S. Tonkens and S. Herbert, "Refining control barrier functions through hamilton-jacobi reachability," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 13 355–13 362.

[11] J. J. Choi, D. Lee, K. Sreenath, C. J. Tomlin, and S. L. Herbert, "Robust control barrier–value functions for safety-critical control," in *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE Press, 2021, p. 6814–6821.

[12] I. Tabbara and H. Sibai, "Learning conservative neural control barrier functions from offline data," *arXiv preprint arXiv:2505.00908*, 2025.

[13] W. Xiao, T.-H. Wang, R. Hasani, M. Chahine, A. Amini, X. Li, and D. Rus, "Barriernet: Differentiable control barrier functions for learning of safe robot control," *IEEE Transactions on Robotics*, vol. 39, no. 3, pp. 2289–2307, 2023.

[14] Z. Qin, D. Sun, and C. Fan, "Sablas: Learning safe control for black-box dynamical systems," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1928–1935, 2022.

[15] D. Tan, F. Acero, R. McCarthy, D. Kanoulas, and Z. Li, "Your value function is a control barrier function: Verification of learned policies using control theory," *arXiv preprint arXiv:2306.04026*, 2023.

[16] O. So, Z. Serlin, M. Mann, J. Gonzales, K. Rutledge, N. Roy, and C. Fan, "How to train your neural control barrier function: Learning safety filters for complex input-constrained systems," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 11 532–11 539.

[17] K. Nakamura, L. Peters, and A. Bajcsy, "Generalizing safety beyond collision-avoidance via latent-space reachability analysis," 2025. [Online]. Available: https://arxiv.org/abs/2502.00935

[18] D. R. Scobee and S. S. Sastry, "Maximum likelihood constraint inference for inverse reinforcement learning," *arXiv preprint arXiv:1909.05477*, 2019.

[19] S. Malik, U. Anwar, A. Aghasi, and A. Ahmed, "Inverse constrained reinforcement learning," in *International conference on machine learning*. PMLR, 2021, pp. 7390–7399.

[20] K. Kim, G. Swamy, Z. Liu, D. Zhao, S. Choudhury, and S. Z. Wu, "Learning shared safety constraints from multi-task demonstrations," *Advances in Neural Information Processing Systems*, vol. 36, pp. 5808–5826, 2023.

[21] G. Chou, D. Berenson, and N. Ozay, "Learning constraints from demonstrations," in *International Workshop on the Algorithmic Foundations of Robotics*. Springer, 2018, pp. 228–245.

[22] K. Leung, S. Veer, E. Schmerling, and M. Pavone, "Learning autonomous vehicle safety concepts from demonstrations," in *2023 American Control Conference (ACC)*. IEEE, 2023, pp. 3193–3200.

[23] A. Robey, H. Hu, L. Lindemann, H. Zhang, D. V. Dimarogonas, S. Tu, and N. Matni, "Learning control barrier functions from expert demonstrations," in *2020 59th IEEE Conference on Decision and Control (CDC)*. Ieee, 2020, pp. 3717–3724.

[24] L. Lindemann, A. Robey, L. Jiang, S. Das, S. Tu, and N. Matni, "Learning robust output control barrier functions from safe expert demonstrations," *IEEE Open Journal of Control Systems*, 2024.

[25] F. Castañeda, H. Nishimura, R. T. McAllister, K. Sreenath, and A. Gaidon, "In-distribution barrier functions: Self-supervised policy filters that avoid out-of-distribution states," in *Proceedings of The 5th Annual Learning for Dynamics and Control Conference*, 2023, pp. 286–299.

[26] H. Yu, S. Farrell, R. Yoshimitsu, Z. Qin, H. I. Christensen, and S. Gao, "Estimating control barriers from offline data," *arXiv preprint arXiv:2503.10641*, 2025.

[27] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey *et al.*, "Maximum entropy inverse reinforcement learning." in *Aaai*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.

[28] M. Qadri, G. Swamy, J. Francis, M. Kaess, and A. Bajcsy, "Your learned constraint is secretly a backward reachable tube," *arXiv preprint arXiv:2501.15618*, 2025.

[29] I. Mitchell, A. Bayen, and C. Tomlin, "A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games," *IEEE Transactions on Automatic Control*, vol. 50, no. 7, pp. 947–957, 2005.

[30] I. Fialho and T. Georgiou, "Worst case analysis of nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 44, no. 6, pp. 1180–1196, 1999.

[31] J. J. Choi, D. Lee, K. Sreenath, C. J. Tomlin, and S. L. Herbert, "Robust control barrier–value functions for safety-critical control," in *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE Press, 2021, p. 6814–6821.

[32] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, no. 3, pp. 497–516, 1957. [Online]. Available: http://www.jstor.org/stable/2372560

[33] A. Clark, "Verification and synthesis of control barrier functions," in *2021 60th IEEE Conference on Decision and Control (CDC)*. Ieee, 2021, pp. 6105–6112.

[34] S. Tonkens and S. Herbert, "Refining control barrier functions through hamilton-jacobi reachability," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 13 355–13 362.

[35] X. Wang, L. Knoedler, F. B. Mathiesen, and J. Alonso-Mora, "Simultaneous synthesis and verification of neural control barrier functions through branch-and-bound verification-in-the-loop training," in *2024 European Control Conference (ECC)*. IEEE, 2024, pp. 571–578.