LatentCBF: A Control Barrier Function in Latent Space for Safe Control

Somnath Sendhil Kumar[1], Qin Lin[2] and John M. Dolan[3]

***Abstract*— Safe control is crucial for safety-critical autonomous systems that are deployed in dynamic and uncertain environments. Quadratic-programming-control-barrier-function (QP-CBF) is becoming a popular tool for safe controller synthesis. Traditional QP-CBF relies on explicit knowledge of the system dynamics and access to all states, which are not always available in practice. We propose LatentCBF (LCBF), a control barrier function defined in the latent space, which only needs an agent's observations, not full states. The transformation from observations to latent space is established by a Lipschitz network-based AutoEncoder. In addition, the system dynamics and control barrier functions are all learned in the latent space. We demonstrate the efficiency, safety, and robustness of LCBFs in simulation for quadrotors and cars.**

I. INTRODUCTION

Many safety-critical robotic systems are deployed in dynamic, uncertain, and even adversarial environments, such as autonomous cars, delivery drones, etc. To provably guarantee the safety of these robots is challenging. Quadratic programming-control-barrier-function (QP-CBF), which solves an optimization problem with safety as a hard constraint online, is becoming a popular tool for safe controller synthesis. Thanks to the forward invariant set property, the trajectories of a dynamic system can be assured to stay in a safe set over an infinite time horizon [1], [2].

Unlike model-based optimal control schemes such as QP-CBF, reinforcement learning (RL) is a direct adaptive optimal control for nonlinear systems [3]. That is to say, RL does not require system identification for a prior model first before the controller design. Instead, RL is expected to adaptively compensate for model uncertainty if a reward function is appropriately designed.

This work presents a unified control schema that employs a data-driven CBF, neural dynamics, and a reinforcement learning policy all in a latent space. The data-driven CBF is a Lipschitz neural network [4] which is a Lipschitz-continuous function required for a robust and safe CBF. We use a latent space instead of a conventional state space because the state space is not always observable in practice, especially for many robotic systems using sensors or images as inputs.

The contributions of our paper can be summarized as follows:

- To the best of our knowledge, we are the first to propose a unified control architecture defined in latent space, which employs a CBF built using a Lipschitz neural network, a dynamics model using a control affine Lipschitz neural network, and a neural policy.
- Domain adaptation for the autoencoder architecture ensures that the latent space is always relevant and can contain features necessary for the barrier function, the dynamics model, and the policy. This has been demonstrated in many dynamic systems from quadrotors to cars in our experiments.

The rest of this paper is organized as follows. Related work is discussed in Section II. We elaborate on the proposed LatentCBF in Section III. The experimental results are presented in Section IV. Concluding remarks and future work are in Section V.

II. RELATED WORKS

In this section, we will briefly review the most important related works in two main categories.

A. Neural Network-based Barrier Functions

1) Barrier functions for safety certificates: [5] proposes a neural barrier function to represent the safe set. It helps to verify if the agent is in the safe or unsafe set. However, unlike CBF, safety certificates are not used for controller synthesis. Alongside the CBF, a reinforcement learning policy is also employed to take action for data collection and inference but uses already defined system dynamics.

2) Barrier functions for controller synthesis: [6], [7], [8] ues neural network-based barrier functions. However, the dynamics models are known. [8] proposes *BarrierNet*, which requires an available dynamics model and pre-defined Hessian and linear cost matrices for the QP. Our approach can learn system dynamics and optimal Hessian and linear cost matrices for the QP.

[6] proposes *rCBLF-QP*, which learns Hessian and linear matrices for the QP via explorations. The major limitation is that the typical definition of a barrier function requires state information that is not typically available in real-world scenarios. Our approach only requires the observation space.

We compare our approach with *rCBLF-QP* and *BarrierNet* in Section IV.

B. Differentiable QP solver

[9] formulates how a QP-based optimization can be added as a differentiable layer as the last layer of a neural network. It proposes to learn the Hessian and linear cost matrices with

[1]Somnath Sendhil Kumar is with the Department of Electrical Engineering, IIT(BHU) Varanasi, India somnath.sendhilk.eee19@itbhu.ac.in

[2]Qin Lin is with the Electrical Engineering and Computer Science Department, Cleveland State University q.lin80@csuohio.edu

[3]John M. Dolan is with the Robotics Institute, Carnegie Mellon University jdolan@andrew.cmu.edu

a cost function by backpropagating through the network. [8] is a similar work but uses a high-order CBF (HOCBF). Its limitation, again, is the requirement of fully available state information.

III. Latent Control barrier Function

A. Preliminary

The basis of our formulation is the latent space learned using a *Lipschitz AutoEncoder*. We define the following *encoder* $E(.)$ and *decoder* $D(.)$:

$$\begin{aligned} z &= E(x) \\ \hat{x} &= D(z) \end{aligned} \tag{1}$$

where $E : X \to Z$, $D : Z \to \hat{X}$. The original observation space X and $\hat{X}$ are both $\in \mathbb{R}^D$. The latent space $Z \in \mathbb{R}^d$. The latent space is used by individual modules of our framework, which will be explained in the upcoming sections. The major difference between the encoder used in our work and a typical encoder is that our encoder is based on a Lipschitz neural network such that

$$\begin{aligned} |E(x_1) - E(x_2)| &\le C * |x_1 - x_2| \\ |z_1 - z_2| &\le C * |x_1 - x_2| \end{aligned} \tag{2}$$

where $z_1, z_2 \sim Z$, $x_1, x_2 \sim X$, and C is a Lipschitz constant. More details on the architecture and the construction of our autoencoder are given in Sec. III-C.

We assume a general nonlinear control-affine system in the learned latent space as follows:

$$\dot{z} = f(z, \theta_f) + g(z, \theta_g)u. \tag{3}$$

where the drift term $f : \mathbb{R}^d \to \mathbb{R}^d$ and the affine control input term $g : \mathbb{R}^d \to \mathbb{R}^{d \times q}$, $u \in \mathbb{R}^q$. This assumption is made based on the convergence of the encoder with the L_{dyn} explained in Sec. III-D. $f(.)$ and $g(.)$ are respectively parameterized by θ_f and θ_g using neural networks and learned in a pipeline explained in Sec. III-D.

A control barrier function in the latent space is defined using a Lipschitz neural network $B(z, \theta_B) : \mathbb{R}^d \to \mathbb{R}$. The safety can be guaranteed in an invariant set by establishing the following constraint [2]:

$$\dot{B}(z) > -\alpha B(z) \tag{4}$$

where α is a Lipschitz class $\mathcal{K}$ function, which can be chosen as a user-defined constant as its simplest form [1]. Eq. (4) is expanded as follows:

$$\begin{aligned} &\frac{dB(z,\theta_B)}{dz}\frac{dz}{dt} > -\alpha B(z, \theta_B) \\ &\frac{dB(z,\theta_B)}{dz}(f(z,\theta_f) + g(z,\theta_g)u) > -\alpha B(z, \theta_B) \\ &\frac{dB(z,\theta_B)}{dz}f(z,\theta_f) + \frac{dB(z,\theta_B)}{dz}g(z,\theta_g)u > -\alpha B(z, \theta_B) \\ &L_{f(z,\theta_f)}B(z) + L_{g(z,\theta_g)}B(z)u(t) > -\alpha B(z) \end{aligned} \tag{5}$$

where $L_{f(z,\theta_f)}B(z) = \frac{dB(z,\theta_B)}{dz}f(z,\theta_f)$ and $L_{g(z,\theta_g)}B(z) = \frac{dB(z,\theta_B)}{dz}g(z,\theta_g)$ are Lie derivatives.

Integrating the constraint Eq. (5) into a quadratic program, a QP-CBF-based controller synthesis can be obtained by solving the following QP problem online:

$$\begin{aligned} \min_u \quad & u^T H(z,\theta_H)u + u^T F(z,\theta_F) \\ \text{s.t. } & L_{f(z,\theta_f)}B(z) + L_{g(z,\theta_g)}B(z)u + \alpha B(z) > 0 \\ & u_{min} \le u \le u_{max} \end{aligned} \tag{6}$$

where u_{min} and u_{max} are control limits. $H(.)$ and $F(.)$ are Hessian and linear cost matricies. $H(z, \theta_H) : \mathbb{R}^d \to \mathbb{R}^{q \times q}$ and $F(z, \theta_F) : \mathbb{R}^d \to \mathbb{R}^{q \times 1}$ are matrices and parameterized by θ_H, and θ_F respectively. The architecture and training of $B(z, \theta_B)$, $H(z, \theta_H)$, and $F(z, \theta_F)$ will be explained in Sec. III-E.

The nominal controller in QP is a policy based on a shallow neural network in the latent space, i.e., $\pi(z)$. There are two policies, each designated for an individual phase.

- π_{adapt}: This is used to span over an ample amount of observation space to collect enough data samples for learning the latent space and the barrier function
- $\pi_{optimal}$: This is the optimal policy trained for the reward of the task in the environment.

$$\begin{aligned} \pi &: \mathbb{R}^d \to \mathbb{R}^c \\ u &= \pi(z, \theta_\pi) \end{aligned} \tag{7}$$

More architectural details about the reward and training pipeline are given in Sec. III-F.

B. Overview

The schematic diagram of our proposed framework is illustrated in Fig. 1. The basic flow of the pipeline can be explained by the following steps:

1) The observation is embedded into the latent space using a Lipschitz encoder (see Eq. (1) and the blocks in yellow and blue in Fig. 1).
2) The latent space then is fed into the control pipeline consisting of a policy in Eq. (7), which outputs nominal control signal u (see the purple neural network).
3) The Lipschitz continuous CBF (see Eq. (4) and the green neural network) and the dynamics in Eq. (3) are also defined in the latent space.
4) All of the aforementioned components are sent to the overall optimization in the QP layer (c.f., Eq. (6)) that optimizes the control signal using the Hessian and linear cost matrices. The final optimized control action is u^*.

Note that the pipeline only represents the forward pass; the following sections on individual modules explain their backpropagation. The gradients from all the modules are accumulated in the encoder for learning an optimal representation.

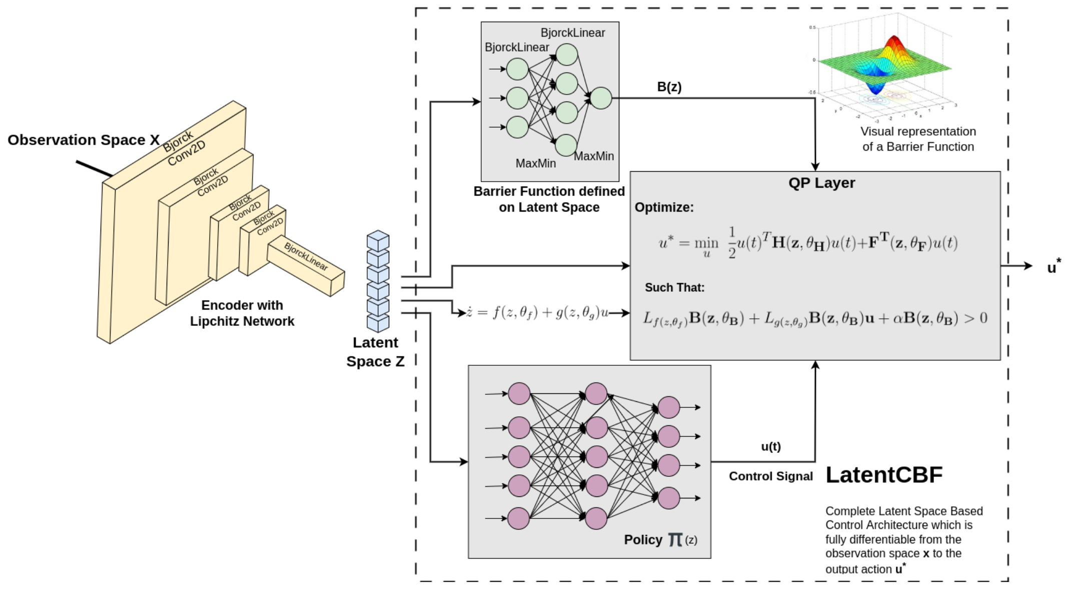
Observation Space X
Bjorck Conv2D
Bjorck Conv2D
Bjorck Conv2D
Bjorck Conv2D
BjorckLinear
Encoder with Lipchitz Network
Latent Space Z
BjorckLinear
BjorckLinear
MaxMin
MaxMin
Barrier Function defined on Latent Space
B(z)
Visual representation of a Barrier Function
QP Layer
Optimize:
$u^* = \min_u \frac{1}{2}u(t)^T\mathbf{H}(\mathbf{z},\theta_\mathbf{H})u(t)+\mathbf{F}^\mathbf{T}(\mathbf{z},\theta_\mathbf{F})u(t)$
Such That:
$L_{f(z,\theta_f)}\mathbf{B}(\mathbf{z},\theta_\mathbf{B}) + L_{g(z,\theta_g)}\mathbf{B}(\mathbf{z},\theta_\mathbf{B})\mathbf{u} + \alpha\mathbf{B}(\mathbf{z},\theta_\mathbf{B}) > 0$
$\dot{z} = f(z,\theta_f) + g(z,\theta_g)u$
$\mathbf{u}^*$
u(t)
Control Signal
Policy π (z)
LatentCBF
Complete Latent Space Based Control Architecture which is fully differentiable from the observation space x to the output action u*

Fig. 1: The illustration of the whole proposed framework. A simple barrier function is visualized with two features i.e., the coordinates as an example.

C. Learning Lipschitz AutoEncoder

The AutoEncoder is an essential component in our framework, since all other components are established in a latent space. To obtain a Lipschitz network-based AutoEncoder [4], we use an architecture similar to [10], which has a classification head in the latent space to backpropagate on the encoder network. Such a design helps the autoencoder in domain adaptation in events of abrupt changes occurring due to training of the control pipeline. The proposed architecture is illustrated in Fig. 2. Following the dashed lines, the loss is backpropagated from all the networks using latent space and gets accumulated into the AutoEncoder.

The AutoEncoder network is trained by optimizing the following reconstruction loss given by

$$L_{recon} = \sum_{i=1}^{n_{batch}} |\hat{x}_i - x_i| \tag{8}$$

The Autoencoder can be flexibly designed for any kind of input, e.g., simple measurement features or even images, by replacing a typical encoder for that modality with specific layers for the Lipschitz network. For example, to get a *Convolutional 2D* layer, we can replace it with the BjorckConv2D [4], and for a *Linear Dense layer*, we can choose BjorckLinear [4]. We have limited options for activations, the most popular choice being *MaxMin* layer [4].

D. Learning System Dynamics

To learn $f(.)$ and $g(.)$ parameterized by neural networks, we formulate a regression model. First, we expand Eq. (3) to get the following form:

$$\frac{d(z)}{dx}\frac{dx}{dt} = f(z,\theta_f) + g(z,\theta_g)u \tag{9}$$

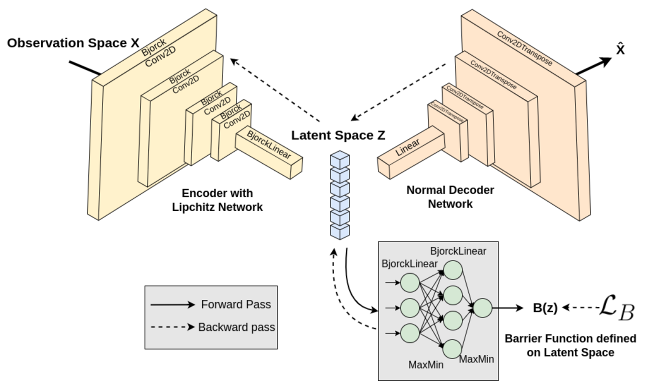

Fig. 2: The illustration of our AutoEncoder design.

where $\frac{d(z)}{dx}$ is the gradient computed in the backward pass of the encoder E (see Eq. (1)). For a given observation x_0, $\frac{dz}{dx}|_{x_0} = \frac{d(E(x))}{dx}|_{x_0}$ is easily calculated by backpropagation. $\frac{dx}{dt}|_{x_0}$ can be computed from the numeric difference between two consecutive observations. Hence the estimated values $\frac{dz}{dt}_{estim}$ will serve as labels in supervised learning with the following loss:

$$L_{dyn} = \sum_{i=1}^{n_{batch}} \left| \frac{dz}{dt}_{estim,i} - f(z,\theta_f)_i - g(z,\theta_g)_i u_i \right| \tag{10}$$

E. Learning Barrier Network

We construct a CBF parameterized by a Lipschitz network called *Barrier Network*. In our experiments, we use four Bjorck Linear Layers and the MaxMin activation function.

To learn the cost matrices $H(z,\theta_H)$ and $F(z,\theta_F)$ in Eq. (6), we use a policy distillation technique [11], where the parameters are updated based on a loss function $l(.)$ measuring the similarity between the output of the latentCBF and expert trajectories from $\pi_{optimal}$ (see (7)), which satisfies $Z_{traj} \in Z_{safe}$.

$$\theta = \arg\max_{\theta} \mathbb{E}[l(\pi_{optimal}(z), u^*)] \tag{11}$$

The gradient for this loss can be computed using the technique in [9]:

$$\begin{bmatrix} d_u \\ d_\lambda \end{bmatrix} = \begin{bmatrix} H & G^T D(\lambda^*) \\ G & D(Gu^* - h) \end{bmatrix}^{-1} \begin{bmatrix} (\frac{\partial l}{\partial u^*})^T \\ 0 \end{bmatrix} \tag{12}$$

where $G = -L_{g(z,\theta_g)}B(z)$, $h = L_{f(z,\theta_f)}B(z) + \alpha B(z)$, $D(\cdot)$ creates a diagonal matrix, and λ is the dual variable for the QP formulation. Eq. (6) can be rewritten correspondingly into a more compact form as follows:

$$\begin{aligned} \min_u \quad & u^T H u + F^T u \\ \text{s.t.} \quad & Gu < h \end{aligned} \tag{13}$$

Once we obtain d_u and d_λ from Eq. (12), we can optimize parameterized $H(z, \theta_H)$ and $F(z, \theta_F)$by

$$\nabla_H l = \frac{1}{2}(d_u u^{*T} + u^* d_u^T) \tag{14}$$

$$\nabla_F l = d_u \tag{15}$$

Now we discuss how to obtain an optimal barrier function. A straightforward mechanism to design the reward $r(x, u)$ is: having a positive reward for reaching the goal, a negative reward for colliding with obstacles or completely deviating from the objective, and otherwise zero.

With such a reward design, we can have an annotating algorithm that identifies whether a particular state is safe or unsafe. For any $x \in X$, after the transformation to latent space under $E : X \to Z$, the agent's trajectory X_{traj} becomes Z_{traj} in the latent space. Hence for $z \sim Z_{traj}$

$$z \in \begin{cases} Z_{safe}, & \text{if } \underset{x(t)\sim X_{traj}}{\overset{x_{term}}{\mathbb{E}}} [\sum r(x(t), u(t))] - p_+ \geq 0 \\ Z_{unsafe}, & \text{if } \underset{x(t)\sim X_{traj}}{\overset{x_{term}}{\mathbb{E}}} [\sum r(x(t), u(t))] - p_- < 0 \end{cases} \tag{16}$$

where $\underset{x(t)\sim X_{traj}}{\overset{x_{term}}{\mathbb{E}}} [\sum r(x(t), u(t))]$ is the expected return or cumulative reward signal along the trajectory X_{traj}, x_{term} is the terminal state of an episode or trajectory, p_+ is the minimum threshold to classify a trajectory safe, and p_- is the maximum threshold to classify a trajectory to be unsafe. The values of p_+ and p_- are hyperparameters. In our experiments we set $p_+ = 0.6 * r_{max}$ and $p_- = 0.2 * r_{min}$, where

$$r_{max} = \max_X r(x(t), u(t)); \quad r_{min} = \min_X r(x(t), u(t))$$

can be obtained from the environment.

With this information and data collected from the sampling phase using π_{adapt}, we have $x \in X_{traj}$. We have the following loss function for training $B(z, \theta_B)$:

$$L_B = \sum_{z\in Z_{safe}} |1 - B(z, \theta_B)| + \sum_{z\in Z_{unsafe}} |B(z, \theta_B) + 1| \tag{17}$$

F. Policy Training

We employ a policy gradient-based reinforcement learning algorithm for training the nominal controller $\pi_{optimal}$.

- π_{adapt}: This is mainly for data collection for training the latent space and the barrier function.
- $\pi_{optimal}$: The policy is entirely trained with off-policy data as the action space is modified by the LCBF. We use DDPG [12], as it works well with both on-policy and off-policy training.

The training is divided into two phases, one with π_{adapt} before reaching the convergence point of the latent space and the barrier function. After this, we switch to phase two with $\pi_{optimal}$ to learn a policy safely with the control barrier function.

IV. Experiments

The observable space is not limited to any specific modality; with a minor modification to the encoder and decoder architecture, any modality can be accommodated. In our experiments, we use images or a stack of images as the observation input to the encoder system, along with sensor data when available. In some test cases, we only use sensor data when vision sensors are not available. To keep the comparison fair across all the approaches, we use a neural network that estimates the system parameters from images and the sensor information. These networks are trained separately for the particular environment and task to act as a state estimator wherever needed by the algorithms we compare. We compare our algorithm with rCLBF-QP [6] and BarrierNet [8] under the same experiments mentioned in these two papers for consistency. For implementing rCLBF-QP [6], we utilized the official implementation, which is made publicly available by the authors. For the state estimation and the vision module, we employed a similar network used in [5], a follow-up paper from the same authors. For BarrierNet [8] implementation, we replicated the same system by ourselves. For the Vision model, we employed the exactly same structure of modules as specified in [7], which is also a follow-up paper from the same authors. Both approaches use predefined dynamics of the system, a predefined barrier function, and a cost function. While these require ground truth from the system or a vision module to extract this information, we compare the number of episodes required to train such vision modules for state estimation to the number of episodes required for LCBF to learn the latent space and barrier function. We use experience from the same exploratory policy without modifying the control signal with these algorithms. Here a model is said to be converged when the test loss of these state estimators is less than a given threshold. As the models are regressing the system values, a threshold of 10^{-3} was experimentally chosen, giving satisfactory prediction results. Our approach defined convergence when the reconstruction loss was less than 10^{-4}. We also compare the safety rate of all three algorithms in different environments. The comparison of tracking errors is summarized in Table I.

TABLE I: Tracking error comparison

Task[2]	Approach	# episodes[1]	$\lvert x - x_{goal} \rvert$
Car trajectory tracking Kinematic model (only sensor data)	rCLBF-QP	136	0.8621
	BarrierNet	148	**0.6432**
	LatentCBF (ours)	**92**	0.7165
Car trajectory tracking Sideslip model (only sensor data)	rCLBF-QP	132	0.9439
	BarrierNet	152	**0.6957**
	LatentCBF (ours)	**92**	0.7165
3D Quadrotor (only sensor data)	rCLBF-QP	104	0.5321
	BarrierNet	84	0.6417
	LatentCBF (ours)	**72**	**0.4967**

TABLE II: Safety rate comparison

Environment[3]	Controller	# episodes[1]	Safety Rate
2D Quadrotor with obstacles (image input of the environment)	rCLBF-QP	224	78%
	BarrierNet	192	81%
	LCBF	**108**	**86%**
3D Quadrotor with an obstacle (image and sensor data as input)	rCLBF-QP	272	98%
	BarrierNet	208	100%
	LCBF	**144**	**100%**

All the tasks are individually explained in the subsection below. Along with the description of the environment, we also present a Visualization of the Barrier Function learned $B(z)$, by applying t-SNE [13] on the observation space to embed into a 2D space corresponding to which $B(z)$ is plotted in a 3D surface. Furthermore, the safe and unsafe samples are also plotted to visualize in which region they lie according to the $B(z)$.

A. *Car Trajectory tracking*

This task is originally from [6]. More details about the task and the environment can be found in the paper. For this task, we had to train a Lyapunov function for rCBLF-QP and train the Hessian and the linear cost matrix for the BarrierNet. The task is straightforward to trace the trajectory given to the ego vehicle. As shown in Table I, BarrierNet is the best performer in terms of the tracking error, but our approach's convergence is the fastest to learn the barrier function and the latent space. We can analyze the learned barrier function in Fig. 3a. Adequate separation between the safe and unsafe sets can be observed in the left contour sub-figure. Almost all unsafe samples (red dots) fall into the black zone (unsafe set). All safe samples (green dots) are in the colorful zone. The corresponding 3D smooth surface is visualized in the right sub-figure. The task is conducted using two different vehicle dynamic models, i.e., the kinematic model without and with sideslip. Consistent conclusions can be drawn from these two experiments.

[2]For trajectory tracking, we compute the maximum tracking error over the trajectory

[1]Individual epoch contains four episodes hence the number of episodes is multiple of 4.

[3]For the quadrotor, we compute % of trials reaching the goal with tolerance $\delta = 0.25$ without collision

B. *3D Quadrotor without obstacle*

This task is again goal tracking but for a 3D quadcopter. Readers are referred to [14] for more details about the experiment setup. Again, only sensor data are available. A Lyapunov function for rCBLF-QP, cost matrices for Barriernet, and Latent Space are trained. In such a task with high uncertainty in the model dynamics, our approach outperforms others with the least number of episodes required for convergence and the smallest tracking error. As illustrated in Fig. 3b, it is clear that the learned latent space is able to represent the safe space within a double-yolk-egg-like zone.

C. *2D Quadrotor with obstacles*

The task is for a reach-and-avoid problem of a quadcopter in 2D space with obstacles [6], [15]. The robot's observation is from a 2D image. The obstacles are marked with a single color throughout the experiment. For the rCBLF-QP, we learn a system extractor to locate the planar quadcopter in the 2D axis and learn the Lyapunov function by simply annotating the region with an obstacle as unsafe. For BarrierNet, we define the same system extractor for location and define the cost of superpositioning rectangles for the obstacles. With the increase in complexity of the input space, these state estimations and barrier function learning limit the system on the information it captures. These require a high number of iterations for convergence. However, our approach still outperforms others, as illustrated in Fig. 3c, where the safe and unsafe regions are well separated.

D. *3D Quadrotor with obstacles*

The setup details for this experiment can be found in [8], [16]. The task again is for a reach-and-avoid problem of a quadcopter but in 3D space with obstacles. The PyBullet gym environment offers us the ground truth for comparison and depth images as the quadrotor's observations. It is challenging for all the algorithms, as the input space dimension is high. For both the rCBLF-QP and BarrierNet, a state estimator had to be trained to estimate the position of the drone relative to the rounded cube object using the RGBD image and the sensor data. Later, the Lyapunov function is also trained for rCBLF-QP. For the BarrierNet, the cost function is defined in the paper [8], and hence learning the Hessian and the linear cost matrix happens in a few iterations to give a nearly optimal solution. Our approach outperforms both BarrierNet and rCBLF-QP controllers regarding the safety rate and the required number of episodes for training. Also, in Fig. 3d, we can observe that the barrier function is a non-convex function in the embedded space from t-SNE, but the high correlation of the safe and unsafe samples to the safe and unsafe region signifies that the barrier function is a convex function in the latent space Z, as $B(z)$ is constructed using a Lipschitz network.

Note that all the baseline approaches have been run with default parameters for network and optimization layers specified in their respective papers.

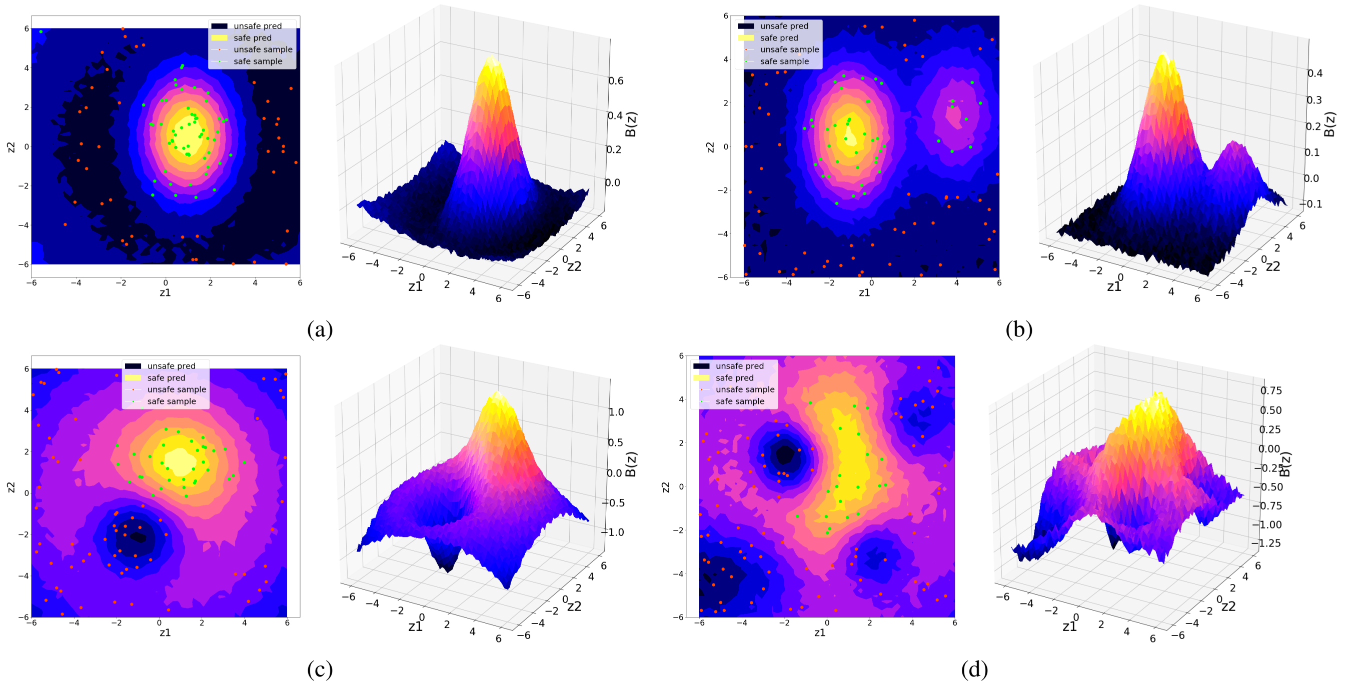

Fig. 3: Visualization of $B(z)$ for the different experiments and projections of safe and unsafe samples on contour and 3D surface view. (a) Car Trajectory tracking experiment; (b) 3D quadrotor without obstacles; (c) 2D quadrotor with obstacles; (d) 3D quadrotor with obstacles

V. CONCLUSIONS

In this paper, we propose a novel framework utilizing LatentCBF, a control barrier function defined in latent space for safe control. The significance is that only observations of robots are required instead of access to the state space, which is not always available in practice. In our unified framework, we deal with AutoEncoder learning, system dynamics learning, and Barrier network learning all in latent space. Owing to the flexibility of representation learning using AutoEncoder, we expect our framework to be more general for a wide range of robotic systems. We have demonstrated the efficacy of our approach for car-like robots and quadrotors. In the future, we plan to test more types of robots in high-fidelity simulators and real testbeds.

REFERENCES

[1] 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.

[2] 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, 2016.

[3] R. S. Sutton, A. G. Barto, and R. J. Williams, “Reinforcement learning is direct adaptive optimal control,” *IEEE control systems magazine*, vol. 12, no. 2, pp. 19–22, 1992.

[4] C. Anil, J. Lucas, and R. Grosse, “Sorting out lipschitz function approximation,” in *International Conference on Machine Learning*. PMLR, 2019, pp. 291–301.

[5] C. Dawson, B. Lowenkamp, D. Goff, and C. Fan, “Learning safe, generalizable perception-based hybrid control with certificates,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1904–1911, 2022.

[6] C. Dawson, Z. Qin, S. Gao, and C. Fan, “Safe nonlinear control using robust neural lyapunov-barrier functions,” in *5th Annual Conference on Robot Learning*. PMLR, 2022, pp. 1724–1735.

[7] W. Xiao, T.-H. Wang, M. Chahine, A. Amini, R. Hasani, and D. Rus, “Differentiable control barrier functions for vision-based end-to-end autonomous driving,” *arXiv preprint arXiv:2203.02401*, 2022.

[8] W. Xiao, R. Hasani, X. Li, and D. Rus, “Barriernet: A safety-guaranteed layer for neural networks,” *arXiv preprint arXiv:2111.11277*, 2021.

[9] B. Amos and J. Z. Kolter, “Optnet: Differentiable optimization as a layer in neural networks,” in *International Conference on Machine Learning*. PMLR, 2017, pp. 136–145.

[10] M. Ghifary, W. B. Kleijn, M. Zhang, D. Balduzzi, and W. Li, “Deep reconstruction-classification networks for unsupervised domain adaptation,” in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*. Springer, 2016, pp. 597–613.

[11] 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)*, 2020, pp. 3717–3724.

[12] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.

[13] L. van der Maaten and G. Hinton, “Viualizing data using t-sne,” *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 11 2008.

[14] 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,” *IEEE Transactions on Robotics*, 2023.

[15] C. Ho, K. Shih, J. S. Grover, C. Liu, and S. Scherer, ““provably safe” in the wild: Testing control barrier functions on a vision-based quadrotor in an outdoor environment,” in *2nd RSS Workshop on Robust Autonomy: Tools for Safety in Real-World Uncertain Environments (RSS 2020)*, 2020.

[16] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, and A. P. Schoellig, “Learning to fly—a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control,” in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 7512–7519.