LatentCBF: Control Barrier Definition on Latent Space

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

Abstract— **Safe execution of policies is crucial for an autonomous system. A popular choice for safety guarantees is CBF(Control Barrier Function). Traditional control barrier functions rely on explicit knowledge of the system dynamics and constraints. Hence We propose LatentCBF (LCBF), a control barrier function definition on the latent space of the agent's observation. LCBFs exploit the properties of deep neural networks to learn a representation of the system used to construct a barrier function. This function acts as a safety constraint that the control signal must satisfy, regardless of the complex system dynamics and uncertainties. By learning this constraint in a latent space, LCBFs have the potential to enable the safe operation of complex systems in a wide range of applications, from autonomous driving to robotics and beyond.**

I. INTRODUCTION

The recent increase in computation power has led to many complex systems that work autonomously. Many control algorithms have been proposed [1], [2], [3], [4] to achieve such a level of autonomy; they are generally classified into model-based optimization and Reinforcement Learning based approaches. While both have different reasons to be used, we can broadly see these approaches with the following characteristics. Model-Based optimizations are computationally expensive for inference but guarantee safety as the system is modeled; hence imposed constraints are met. In the literature, a popular approach is Control Barrier Function(CBF), which enables any controller with safety guarantees. While the Reinforcement learning approach does not guarantee safety, it can fuse multiple modalities and does not require the user to define the system's dynamics and constraints, which mitigates a great deal of details required about the system.

This work presents a hybrid control schema that employs a data-driven CBF, neural dynamics, and a Reinforcement learning policy on a latent space. The data-driven CBF is a Lipschitz neural network [5] which constrains the neural network to be a Lipschitz continuous function which is required for robust and safe CBF definition. Here the latent space is explicitly used over state space as the state space is not observable through the sensors or inputs available, and latent space would represent a real-world scenario. A latent space is used in literature to represent the scene in a more interpretable and usable format and to capture auxiliary information. It helps embed the required information for the policy while also containing the information required for the barrier function to define its safety constraints. The major downfall in real-world applications is the requirement of ground truth for inference and training models that extract these values with an inevitable error and requires integrating different modules for state estimation. Encoding the sensors' inputs into a latent space mitigates the need for such complex systems by infusing the required information into features. The only concern is the non-interpretability of the latent space, i.e., individual features in the latent space cannot be used as any particular system parameter defined in the literature. Hence Defining a Barrier Function and Dynamics on such a latent space will enable us to work with a high-dimensional and complex system.

[1]Somnath Sendhil Kumar is with 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

Hence Our contributions in the paper can be summarized as follows.

- To the best of our knowledge, we are the first to propose a control architecture defined on 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. The CBF is used for modifying the control signal rather than just using it as a safety certificate.
- First, to define LatentCBF, a Lipschitz continuous differential layer is a CBF whose constraints are learned with only the latent space as the input.
- 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.

II. RELATED WORKS

This work is based on the rich literature on Control Barrier Functions and Differentiable QP, which is explained in the following subsections. However, these approaches try to enhance the pipeline using adaptive algorithms. They use complete system information at some parts of the pipeline and classical approaches for the rest. Hence these approaches need a unified input space for all modules making information consistent across them.

A. Neural Network based Barrier Function definition

1) Safety Guarantee certificates: [7] proposes a neural barrier function only to represent the safe set. It helps to define if the agent is in the safe or unsafe set. However, this is not used for modifying the control signal. Alongside the CBF, a reinforcement learning policy is also employed to take action for data collection and inference. Here the dynamics are already defined and used for the CBFs QP definition, but the learned dynamics are not used for the CBF definition for modifying the control signal.

2) Control Modification: [8], [3][10]. Define a barrier function using a neural network. However, the dynamics model is to be defined for each system. Furthermore, the policies are trained separately for the same task or utilize an MPC for control.

B. Differentiable QP solver

[6] defines how a QP based optimization can be added as a differentiable layer as the last layer of a neural network. It proposes techniques to learn the Hessian and linear cost matrix with a Cost function by backpropagating through the network. [10] Is similar in how the backpropagation for the differentiable layer is defined. The minor difference being the QP is formulated using the HOCBF definition in [10]. Hence again, there is a requirement for a cost function that generally uses the state information, making it dependent on the complete system information.

III. APPROACH

A. Problem Statement

This section briefly describes the system's elements and refers to different papers explaining these modules. The basis of our formulation is the latent space learned using a simple Lipschitz Autoencoder. We mathematically define

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

The latent space from the encoder is used by individual modules, which is explained in upcoming sections. The Major difference between the encoder used and a typical encoder is that it is composed of a Lipschitz neural network making the relation $E : X \rightarrow Z$, from $X \in \mathbb{R}^D$ to $Z \in \mathbb{R}^d$ and holds

$$\begin{aligned} |E(x1) - E(x2)| &<= C * |x1 - x2| \\ |z1 - z2| &<= C * |x1 - x2| \end{aligned} \tag{2}$$

where C is Lipschitz constant More detail of the architecture and the construction of AutoEncoder is given in III-C.

Now defining the system dynamics, we assume the system to be a control affine on the learned latent space and can be represented as

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

This assumption is made based on the convergence of the encoder with the $L_{dynamics}$ explained in III-D. The $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ affine dynamics drift term and $g : \mathbb{R}^n \rightarrow \mathbb{R}^{nq}$ affine dynamics control term. Both are constructed using θ_f and θ_g and learned in a pipeline explained in III-D.

A control barrier function[1] is defined in its rudimentary form using a Lipschitz neural network $B : \mathbb{R}^d - > \mathbb{R}^1$, $B(z)$ with the following constraint for safe control.

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

$$dB(z, \theta_B)/dt > -\alpha * B(z, \theta_B)$$

$$dB(z, \theta_B)/dz * dz/dt > -\alpha * B(z, \theta_B)$$

$$dB(z, \theta_B)/dz * z_dot > -\alpha * B(z, \theta_B)$$

$$dB(z, \theta_B)/dz(f(z, \theta_f) + g(z, \theta_g) * u) > -\alpha * B(z, \theta_B)$$

$$(z, \theta_B)/dz * f(z, \theta_f) + dB(z, \theta_B)/dz * g(z, \theta_g) * u > -\alpha * B(z, \theta_B)$$

$$\begin{aligned} \min_{u} \quad & u^T(t)H(z, \theta_H)u(t) + u^T(t)F(z, theta_F) \\ & L_{f(\theta_f}B + L_{g(\theta_g)}Bu + \alpha B > 0 \\ & u_{min} < u < u_{max} \end{aligned} \tag{5}$$

The solution from the above equation is used as the control algorithm. The architecture and training of B(z), H(z), and F(z) are explained in III-E.

A policy is a shallow neural network on the latent space $\pi(z)$. There are 2 policies, each designated for individual tasks.

- 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.

$$\begin{aligned} \pi &: \mathbb{R}^d \rightarrow \mathbb{R}^c \\ u &= \pi(z, \theta_\pi) \end{aligned} \tag{6}$$

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

B. Overview

With the description of individual terms used throughout the paper, the main inference pipeline used in the paper can be seen in figure 1.

The basic flow of the pipeline can be explained by the following, the observation is embedded into the latent space using a Lipschitz encoder 1. The latent space then is fed into the control pipeline consisting of a policy 6, which outputs raw control signal u. This latent space is also by the Lipschitz continuous control barrier function 4, and the dynamics function 3, which is required to define the Optimization problem as a Quadratic Program 5 that optimizes the control signal using the Hessian and linear cost matrices. The output signal acts on the system hence guaranteeing safety with an optimal barrier function.

In figure 1, a simple barrier function is visualized to with 2 features i.e., the coordinates. It represents how well the learned function is Lipschitz continuous. This is difficult to visualize for a higher-dimensional system. While the pipeline only represents the forward pass, the Following sections of individual modules ex

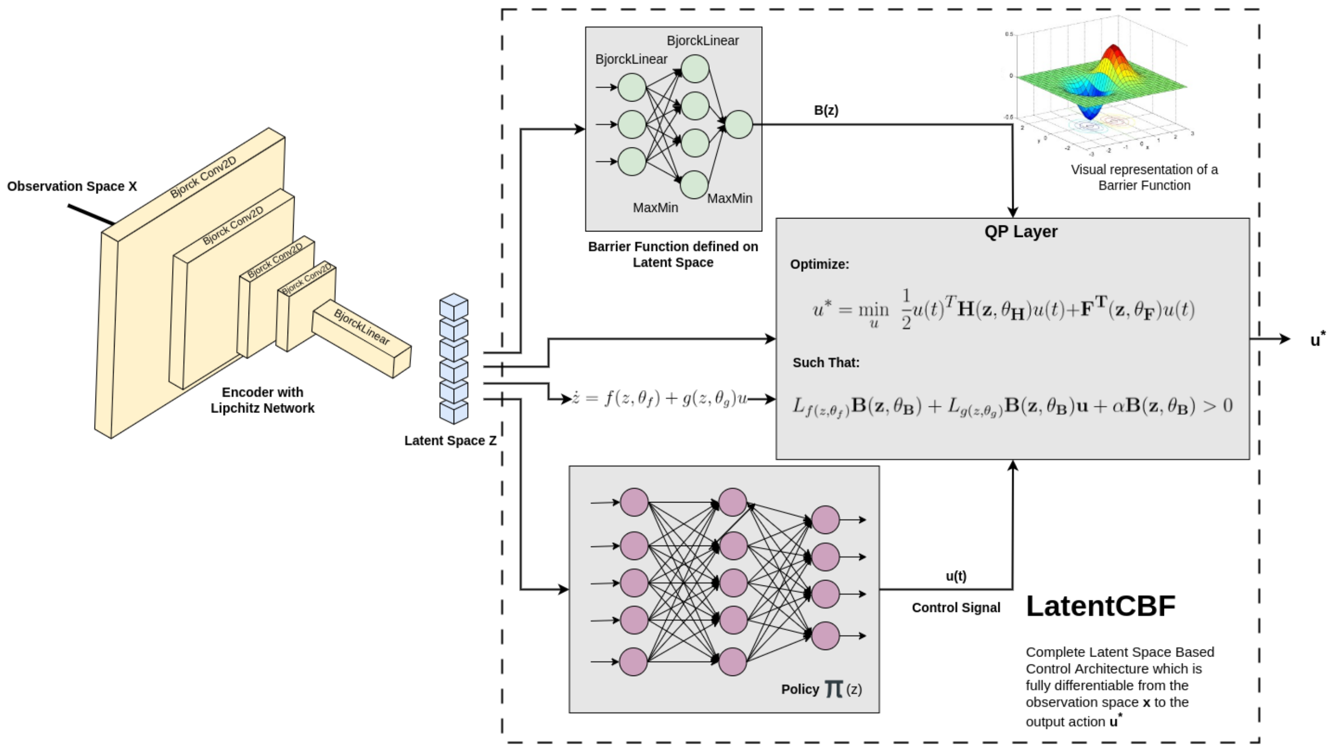
Observation Space X
Bjorck Conv2D
Bjorck Conv2D
Bjorck Conv2D
Bjorck Conv2D
BjorckLinear
Encoder with Lipchitz Network
Latent Space Z
BjorckLinear
BjorckLinear
MaxMin
MaxMin
Barrier Function defined on Latent Space
B(z)
Visual representation of a Barrier Function
QP Layer
Optimize:
$u^* = \min_u \frac{1}{2}u(t)^T\mathbf{H}(\mathbf{z},\theta_\mathbf{H})u(t) + \mathbf{F}^\mathbf{T}(\mathbf{z},\theta_\mathbf{F})u(t)$
Such That:
$L_{f(z,\theta_f)}\mathbf{B}(\mathbf{z},\theta_\mathbf{B}) + L_{g(z,\theta_g)}\mathbf{B}(\mathbf{z},\theta_\mathbf{B})\mathbf{u} + \alpha\mathbf{B}(\mathbf{z},\theta_\mathbf{B}) > 0$
$\dot{z} = f(z,\theta_f) + g(z,\theta_g)u$
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. Complete pipeline illustrating a sample barrier function and modified control signal

C. Lipschitz AutoEncoder

The principle component on which the whole pipeline is dependent is the autoencoder. The proposed autoencoder is a standard autoencoder, with the Encoder being a Lipschitz Network [5]. As the latent space is free to embed the observation into any sparse representation, there has to be a constraint for it to be relevant enough to define a barrier function on it. So we use the architecture similar to [11], which has a classification head on the latent space to backpropagate on the encoder network. This helps the autoencoder in domain adaptation in events of abrupt changes occurring due to training of the control pipeline. Hence the proposed architecture can be seen in 2.

The network is trained using the reconstruction loss given by

$$L_{recon} = |\hat{x} - x| \tag{7}$$

And the loss is backpropagated from the networks that use this latent space. As the Autoencoder has a barrier function in its training phase, even the barrier function loss is explained in section III-E.

The Autoencoder can be designed for any kind of input, whether simple features or images, by replacing a typical encoder for that modality with specific layers for the Lipschitz network. Example: For the Convolutional 2D layer, we replace it with BjorckConv2D[5] layer, and for the Linear Dense layer, we replace them with BjorckLinear[5] Layer. We have limited options for activations, the most popular being the MaxMin layer[5]. Making the following changes enables us to use a Lipschitz autoencoder.

D. Dynamics Model

The Dynamics model of the system is defined in 3. The F affine dynamic drift and G affine dynamic control terms are neural networks and need to be learned. There are many non-linear system identification techniques[12]. However, this can be easily framed as a regression task.

$$\dot{z} = f(z,\theta_f) + g(z,\theta_g)u \tag{8}$$

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

$$\frac{d(z)}{dx} * \frac{dx}{dt} = f(z,\theta_f) + g(z,\theta_g) * u \tag{10}$$

$\frac{d(z)}{dx}$ is the gradient computed in the backward pass of the encoder E as 1.

Hence for the input of x_0 into the Encoder, we can write $\frac{dz}{dx}|_{x_0} = \frac{d(E(x))}{dx}|_{x_0}$ is easily calculated with backpropagation for sample x_0. $\frac{dx}{dt}|_{x_0}$ can easily be fetched or computed from the system by a simple observer. Hence with the value of $\frac{dz}{dt}$ estimated and the value of u from the policy, we can backpropagate the dynamics on the following loss

$$L_{dynamics} = |\frac{dz}{dt}_{e}stim - f(z) - g(z) * u| \tag{11}$$

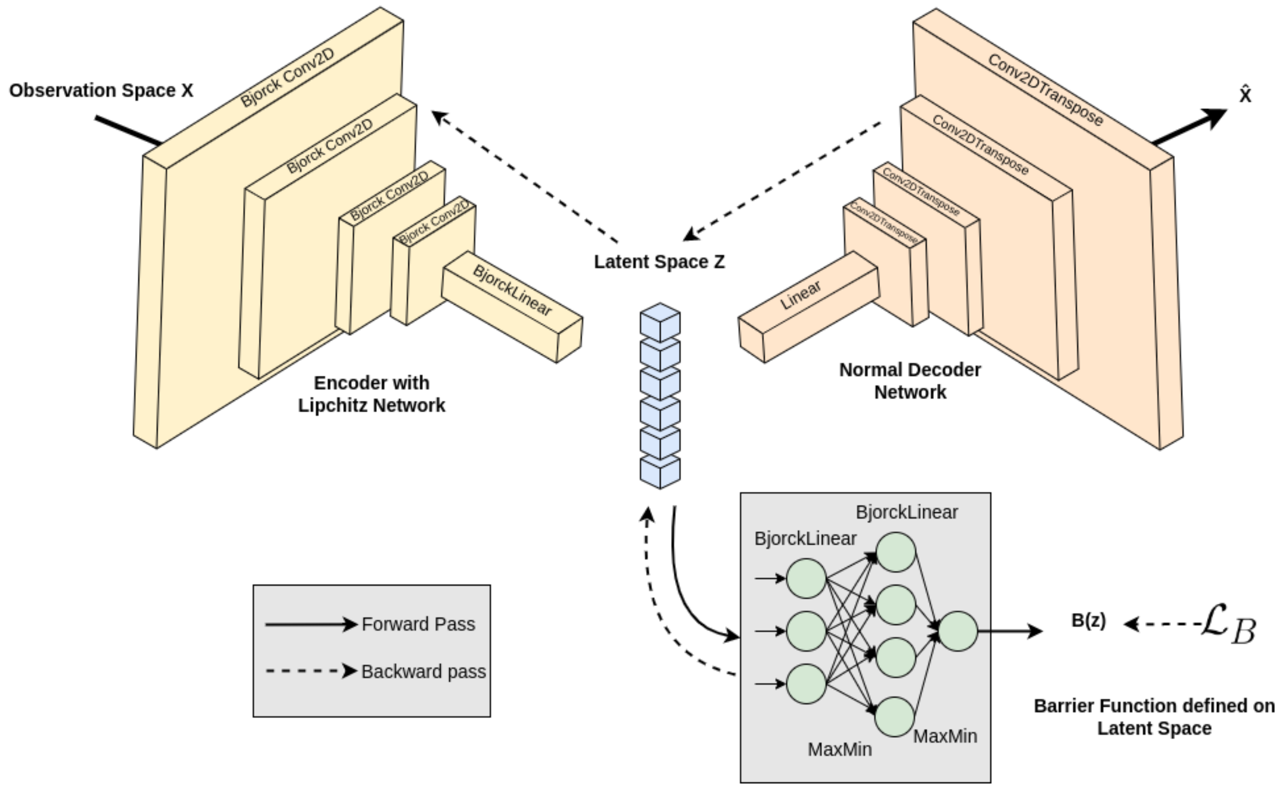

Fig. 2. Domain adaptation for auto encoder with a barrier function

E. Control Barrier Function definition

With the mathematical definition of a Control Barrier function defined at eqn, we construct such a function using the Lipschitz network. The Barrier Network is built using four Bjorck Linear Layers and MaxMin activation.

The definition of the Barrier function is given in 4, and the constraint that the system should follow to be in the safe set is 5.

From our QP formulation and the definition of differentiable layer [6], we can write the gradients for backpropagation for the parameters in the QP as follows

By Solving

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

We obtain d_u and d_λ from which we can compute the gradients

$$\nabla_H l = \frac{1}{2}(d_u u^* T + u^* d_u^T \tag{13}$$

$$\nabla_F l = d_u \tag{14}$$

With a differentiable layer defined by the above formulation. We now require an optimal barrier function. Consider environments with sparse reward signals $r(z,u)$, having a positive reward for reaching the goal, a negative reward for colliding with obstacles or completely deviating from the objective, and otherwise zero. This is generally the reward structure in the control problem, and this is also consistent with our environments.

With this, we can have an annotating algorithm that identifies whether a particular state is safe or unsafe.

For the $x \in X$, after transformation to latent space $E : X \to Z$. If the agent follows a trajectory X_{traj}, its latent space follows a trajectory of Z_{traj} given by the relation 1.

Hence for $z \sim Z_{traj}$

$$z \in \begin{cases} Z_{safe}, & \text{if } \underset{z(t)}{\overset{z_{term}}{\mathbb{E}}}{}_{z \sim Z_{traj}} [\sum r(z(t), u(t))] + p+ \geq 0 \\ Z_{unsafe}, & \text{if } \underset{z(t)}{\overset{z_{term}}{\mathbb{E}}}{}_{z \sim Z_{traj}} [\sum r(z(t), u(t))] + p- < 0 \end{cases} \tag{15}$$

With this information and data collected from phase 1 of training we have $x \in X_{traj}$ We can define the loss function for $B(z, \theta_B)$

$$L_B = \sum_{z \in Z_{safe}} |1 - B(z, \theta_B| + \sum_{z \in Z_{unsafe}} |B(z, \theta_B) + 1| \tag{16}$$

This loss has to be backpropagated through the Encoder as the $E : x \to z$ is also changing hence the only constant space we have is X_{traj} and not Z_{traj}.

F. *Policy Training*

We use simple policy, trained using reinforcement learning to maximize the expected reward. We employ a simple Policy gradient algorithm for training the $\pi_{optimal}$, the policy which is optimal at solving the task. We have two policies in total.

- π_{adapt}: This is mainly for data collection for training the latent space and the barrier function. This is generally a nominal controller for the system that helps generate data. In a complex scenario, this is the $\pi_{optimal}$ while it is training as nominal controllers do not collect a diverse set of observations.
- $\pi_{optimal}$: Here, the policy is entirely trained with off-policy data as the action space is modified by the LCBF. Hence we use DDPG [13] as it works well with both on-policy and off-policy training.

Hence 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. Hence this phase fine-tunes all the modules of the pipeline

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 tests, we use images or a stack of images as the input to the encoder system, along with sensor data when available. In a few environments, we use only sensor data as the environment lacks any vision sensor or input. 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 [8], BarrierNet [10] and also use the same environment and experiments mentioned in these papers for consistency. For implementing rCLBF-QP, we utilized the official implementation, which is made publicly available by the authors. For the state estimation or the vision module, we employed a similar network used in [7], a follow-up paper from the same author. For BarrierNet implementation, we replicated the same system with the details in the paper. For the Vision model, we employed the exact structure of modules as specified in [3], which is also a follow-up paper from the authors of [10]. Both these approaches use predefined dynamics of the system and a predefined barrier function or cost function that is backpropagated. 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. Below is the table of comparison setups for the list of environments.

TABLE I
EXPERIMENTATION RESULT COMPARING WITH TRACKING ERROR

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

TABLE II
EXPERIMENTATION RESULT COMPARING WITH SAFETY RATE

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

These Environments are individually explained in the subsection below.

A. *Car Trajectory tracking*

This experiment is taken from [8], and more details about the task and environment can be found in the paper. For this task, we had to train the 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. Barrier Net is the best performer because of the HOCBF formulation, but our controller converges fastest to learn the barrier function and the latent space. The two different dynamics model makes the results of the two controllers vary while our controller learns its dynamics.

B. *3D Quadcopter with no obstacle*

An individual can refer to [9] for this experiment's exact formulation. This system only has sensor data similar to car

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

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

[3]For 2D quadrotor, we compute % of trials reaching the goal with tolerance = 0.3 without collision

trajectory tracking. Hence a Lyapunov function for rCBLF-QP, Cost Matrices for Barriernet, and Latent Space for Our approach are trained. Due to the high uncertainty in the model dynamics, our approach can perform best with the least number of episodes required for convergence.

C. 2D Quadcopter with obstacles

This environment is defined in [8], and one can refer to learn more about it. The implementation of the system is taken from a publicly available codebase published under [14]. This system uses a 2D image containing markers representing the quadcopter and final goal. The obstacles are represented with a single color throughout the environment. 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 a cost of superpositioning rectangles inscribing 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 amount of iterations for convergence to achieve the same task of regression, which is associated with some error. Hence, these controllers show worse results than our proposed controller because of computing absolute state values.

D. 3D Quadrotor with obstacle

This task is defined in [10], and one can find the formulation. This environment is built from the implementation of [15]. The PyBullet gym environment enables us to get the ground truth and depth image from the drone camera feed. 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, requiring additional iterations. For the BarrierNet, the cost function is defined in the paper, and hence learning the Hessian and the linear cost matrix happens in a few iterations to give a close-optimal solution. Our controller outperforms both BarrierNet and rCBLF-QP controllers regarding the safety rate and the efficiency of the experience required for training.

These experiments have been run with default parameters for network and optimization layers specified in their respective papers.

V. CONCLUSIONS

We proposed LCBF, a control barrier function defined on latent space to mitigate the requirement of ground truth or state estimators. With the current formulations of LCBF, we learned the dynamics and barrier function. These can be generalized and can be used in more control problems. Hence we hope to inspire more work on standardizing such hybrid controllers, As these tend to capture a more accurate to-world scenario and lead to the successful deployment of complex algorithms on hardware. Prospectives in this work would be working on interpreting these latent spaces by understanding how rich these latent spaces are and how they can be obtained from performing different tasks.

REFERENCES

[1] Aaron D Ames, Jessy W Grizzle, and Paulo Tabuada. Control barrier function based quadratic programs with application to adaptive cruise control. In 53rd IEEE Conference on Decision and Control, pages 6271–6278. IEEE, 2014.

[2] Shao, Hao, Wang, Letian, Chen, RuoBing, Li, Hongsheng, and Yu Liu. ”Safety-Enhanced Autonomous Driving Using Interpretable Sensor Fusion Transformer.” ArXiv, (2022). Accessed March 2, 2023. https://doi.org/10.48550/arXiv.2207.14024.

[3] Xiao, Wei, Wang, Tsun, Chahine, Makram, Amini, Alexander, Hasani, Ramin, and Daniela Rus. ”Differentiable Control Barrier Functions for Vision-based End-to-End Autonomous Driving.” ArXiv, (2022). Accessed March 2, 2023. https://doi.org/10.48550/arXiv.2203.02401.

[4] Aaron D Ames, Xiangru Xu, Jessy W Grizzle, and Paulo Tabuada. Control barrier function based quadratic programs for safety critical systems. IEEE Transactions on Automatic Control, 62(8): 3861–3876, 2017.

[5] Anil, Cem, Lucas, James, and Roger Grosse. ”Sorting out Lipschitz function approximation.” ArXiv, (2018). Accessed March 2, 2023. https://doi.org/10.48550/arXiv.1811.05381.

[6] B. Amos and J. Z. Kolter. Optnet: Differentiable optimization as a layer in neural networks. In Proceedings of the 34th International Conference on Machine Learning - Volume 70, pages 136– 145, 2017.

[7] Dawson, C., Lowenkamp, B., Goff, D. & Fan, C. ”Learning Safe, Generalizable Perception-Based Hybrid Control With Certificates”. *IEEE Robotics And Automation Letters*. **7**, 1904-1911 (2022)

[8] Dawson, C., Qin, Z., Gao, S. & Fan, C. Safe Nonlinear Control Using Robust Neural Lyapunov-Barrier Functions. *5th Annual Conference On Robot Learning* . (2021), https://openreview.net/forum?id=8K5kisAnb_p

[9] Dawson, C., Gao, S. & Fan, C. ”Safe Control With Learned Certificates: A Survey of Neural Lyapunov, Barrier, and Contraction Methods for Robotics and Control”. *IEEE Transactions On Robotics*. pp. 1-19 (2023)

[10] Xiao, Wei, Ramin Hasani, Xiao Li, and Daniela Rus. ”Barriernet: A safety-guaranteed layer for neural networks.” arXiv preprint arXiv:2111.11277 (2021).

[11] Ghifary, Muhammad, W. Bastiaan Kleijn, Mengjie Zhang, David Balduzzi, and Wen 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, pp. 597-613. Springer International Publishing, 2016.

[12] Gedon, Daniel, Niklas Wahlström, Thomas B. Schön, and Lennart Ljung. ”Deep state space models for nonlinear system identification.” IFAC-PapersOnLine 54, no. 7 (2021): 481-486.

[13] Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. ”Continuous control with deep reinforcement learning.” arXiv preprint arXiv:1509.02971 (2015).

[14] Ho, C., Shih, K., Grover, J., Liu, C. & Scherer, S. “Provably Safe” in the Wild: Testing Control Barrier Functions on a Vision-Based Quadrotor in an Outdoor Environment. *2nd RSS Workshop On Robust Autonomy: Tools For Safety In Real-World Uncertain Environments (RSS 2020)*. (2020), https://openreview.net/forum?id=CrBJIgBr2BK

[15] Panerati, J., Zheng, H., Zhou, S., Xu, J., Prorok, A. & Schoellig, A. ”Learning to Fly—a Gym Environment with PyBullet Physics for Reinforcement Learning of Multi-agent Quadcopter Control.” *2021 IEEE/RSJ International Conference On Intelligent Robots And Systems (IROS)*. (2021)