Learning Barrier Functions With Memory for Robust Safe Navigation

Kehan Long, Cheng Qian, Jorge Cortés, and Nikolay Atanasov

***Abstract*—Control barrier functions are widely used to enforce safety properties in robot motion planning and control. However, the problem of constructing barrier functions online and synthesizing safe controllers that can deal with the associated uncertainty has received little attention. This letter investigates safe navigation in unknown environments, using on-board range sensing to construct control barrier functions online. To represent different objects in the environment, we use the distance measurements to train neural network approximations of the signed distance functions incrementally with replay memory. This allows us to formulate a novel robust control barrier safety constraint which takes into account the error in the estimated distance fields and its gradient. Our formulation leads to a second-order cone program, enabling safe and stable control synthesis in a prior unknown environments.**

***Index Terms*—Machine learning for shape modeling, robust and adaptive control, sensor-based control.**

I. Introduction

MODERN applications of ground, aerial, and underwater mobile robots necessitate safe operation in unexplored and unstructured environments. Planning and control techniques that ensure safety, relying on limited field-of-view observations, are critical for autonomous navigation. The seminal work of Khatib [1] introduced artificial potential fields to enable collision avoidance during not only the motion planning stage but also the real-time control of a mobile robot. Potential fields inspired subsequent work on joint path-planning and control. Rimon and Koditschek [2] developed navigation functions, special artificial potential functions designed to simultaneously guarantee collision avoidance and stabilization to a desired goal configuration. In the 2000's, barrier certificates were proposed as a general construction to verify safety of closed-loop nonlinear and hybrid systems [3], [4]. Barrier certificates were extended by [5] to consider control inputs explicitly and enable safe control design. Control barrier functions (CBFs) have become a key technique for encoding safety constraints, and have been successfully employed, along with control Lyapunov functions (CLFs), to encode stability requirements in quadratic program (QP) control synthesis for control-affine nonlinear systems [6].

Manuscript received October 15, 2020; accepted March 5, 2021. Date of publication March 31, 2021; date of current version April 19, 2021. This work was supported by NSF RI IIS-2007141. *Kehan Long and Cheng Qian contributed equally to this work. (Corresponding author: Kehan Long.)*

The authors are with the Contextual Robotics Institute, University of California San Diego, La Jolla, CA 92093 USA (e-mail: k3long@ucsd.edu; chqian@ucsd.edu; cortes@ucsd.edu; natanasov@ucsd.edu).

Digital Object Identifier 10.1109/LRA.2021.3070250

A common assumption in many CLF-CBF-QP works is that the robot already has complete knowledge of the unsafe regions, encoded in an *a priori known* CBF. However, navigation in unknown environments requires online estimation of unsafe regions using onboard sensing and, hence, a CBF should also be constructed online. This paper considers a mobile robot equipped with a LiDAR scanner and tasked to follow a motion plan, relying on streaming range measurements to ensure safety. We focus on constructing CBFs, approximating the shape of the objects in the environment, and introduce corresponding safety constraints in the synthesis of the robot's control policy to enforce safety.

The signed distance function (SDF) of a set Ω in a metric space determines the distance of a given point $\boldsymbol{x}$ to the boundary of Ω, with sign indicating whether $\boldsymbol{x}$ is in Ω or not. SDFs are an implicit surface representation, employed in computer vision and graphics for surface reconstruction and rendering [7], [8]. In contrast with other object geometry representations, an SDF provides distance and gradient information to object surfaces directly, which is necessary information for collision avoidance in autonomous navigation [9]. In this work, we approximate the SDFs of observed objects and define corresponding CBF safety constraints for control synthesis. Using the LiDAR measurements, we incrementally train a multilayer perceptron to model each obstacle's SDF shape online. To balance the accuracy and efficiency of SDF reconstruction, we employ replay memory in training [10].

The obstacle SDF approximations provide distance and gradient information for the barrier functions. We take into account the estimation errors for the controller synthesis, resulting in safety constraints where the input appears both linearly and within an l_2-norm term. The presence of such constraints means that the control synthesis problem can no longer be stated as a QP but we show that the problem can be formulated as a second-order cone program (SOCP), which is still convex and can be solved efficiently. The proposed CLF-CBF-SOCP framework integrates the SDF estimation of the CBFs to synthesize safe controls at each time instant, allowing the robot to safely navigate the unknown environment. In summary, this paper makes the following contributions.

- An online incremental training approach, utilizing replay memory, is proposed for deep neural network approximation of the signed distance functions of objects observed with streaming distance measurements.
- Our analysis shows that incorporating safety constraints that account for the worst-case estimation error of the SDF approximations into a control synthesis optimization problem leads to a convex SOCP formulation.

II. Related Work

This section reviews related work on scene representations and CLF-CBF techniques for safe control.

Occupancy mapping techniques aim to partition an environment into occupied and free subsets using sensor observations. Octomap [11] is a probabilistic occupancy mapping algorithm that uses a tree-based structure to store and update the probability of occupied and free space based on streaming range observations. Occupancy information alone may not be enough for safe planning and control, since many algorithms require distance (or even gradient) information to the obstacle surfaces. Voxblox [12] is an incremental truncated SDF mapping algorithm that approximates the distance to the nearest surface at a set of weighted support points. The approach also provides an efficient extension from truncated to complete (Euclidean) SDF using an online wavefront propagation algorithm. Fiesta [9] further improves the efficiency and accuracy of building online SDF maps by using two independent queues for inserting and deleting obstacles separately and doubly linked lists for map maintenance. These SDF representations, however, require discretizing the environment into voxels, resulting in potentially large memory use. Recently, impressive results have been achieved in object shape modeling using deep neural networks. DeepSDF [7] and Occupancy Nets [13] implicitly represent 3D shapes by supervised learning via fully connected neural networks. Gropp *et al.* [14] further extended the DeepSDF model by introducing the *Eikonal* constraint that any SDF function must satisfy in training the neural network. We extend this offline training method in two ways: enabling online training by introducing replay memory, which helps the network to remember the reconstructed shape from past observations, and adding truncated signed distance points to the training set to ensure the correct sign of the approximated SDF. Compared with state-of-the-art SDF methods [9], [12] for navigation and obstacle avoidance, which rely on discretization, our approach enables continuous and differentiable SDF representations, as required by the CBF framework.

Quadratic programming with CBF constraints offers an elegant and efficient framework for safe control synthesis in robot navigation tasks [15], [16]. This approach employs CLF constraints to stabilize the system and achieve the control objectives whenever possible, whereas the CBF constraints are used to ensure safety of the resulting controller at all times. CBFs are used for safe multi-robot navigation in [15]. CLF and CBF constraints are combined to solve a simultaneous lane-keeping (LK) and adaptive speed regulation (ASR) problem in [16]. CLFs are used to ensure convergence to the control objectives for LK and ASR, whereas CBFs are used to meet safety requirements. However, the barrier functions in these approaches are assumed to be known. When the environment is unknown, a robot may only rely on the estimation of barrier functions using its sensors. A closely related work by Srinivasan *et al.* [17] presents a Support Vector Machine (SVM) approach for the online synthesis of barrier functions from range data. By approximating the boundary between occupied and free space as the SVM decision boundary, the authors extract CBF constraints and solve a CBF-QP to generate safe control inputs. Our approach constructs CBFs directly from an approximation of the object SDFs and explicitly considers the potential errors in the CBF constraints when formulating the control synthesis optimization problem.

III. Problem Formulation

Consider a robot whose dynamics are governed by a nonlinear control-affine system:

$$\begin{aligned} \dot{\boldsymbol{x}} &= f(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u} \\ \boldsymbol{y} &= v(\boldsymbol{x}) \end{aligned} \tag{1}$$

where $\boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^n$ is the robot state, $\boldsymbol{u} \in \mathcal{U} \subseteq \mathbb{R}^m$ is the control input, and $\boldsymbol{y} \in \mathcal{Y} \subseteq \mathbb{R}^w$ is the system output. Taking the robot in Section VI as an example, $\boldsymbol{x}$ is the robot position and orientation, $\boldsymbol{u}$ is the linear and angular velocity input, while $\boldsymbol{y}$ is the robot position. Assume that $f : \mathbb{R}^n \mapsto \mathbb{R}^n$, $g : \mathbb{R}^n \mapsto \mathbb{R}^{n\times m}$, and $v : \mathbb{R}^n \mapsto \mathbb{R}^w$ are continuously differentiable functions. Define the admissible control input space as $\mathcal{U} := \{\boldsymbol{u} \in \mathbb{R}^m \mid A\boldsymbol{u} \leq \boldsymbol{b}\}$, where $A \in \mathbb{R}^{k\times m}$ and $\boldsymbol{b} \in \mathbb{R}^k$. The output space $\mathcal{Y}$ is a Euclidean robot workspace (e.g., robot positions), used for collision checking. The state space $\mathcal{X}$ is partitioned into a closed safe set $\mathcal{S}$ and an open obstacle set $\mathcal{O}$ such that $\mathcal{S} \cap \mathcal{O} = \emptyset$ and $\mathcal{X} = \mathcal{S} \cup \mathcal{O}$. Correspondingly, the workspace $\mathcal{Y}$ is partitioned into a closed set $v(\mathcal{S})$ and an open set $v(\mathcal{O})$, obtained by applying the output function v to each element in $\mathcal{S}$ and $\mathcal{O}$. The obstacle set $\mathcal{O}$ is further partitioned into K obstacles, denoted $\mathcal{O}_1, \mathcal{O}_2, \ldots \mathcal{O}_K$, such that $\mathcal{O} = \cup_{i=1}^K \mathcal{O}_i$ and $\mathcal{O}_j \cap \mathcal{O}_k = \emptyset$ for any $0 \leq j, k \leq K$. Each of these sets is defined through a continuously differentiable function $\varphi_i : \mathbb{R}^w \mapsto \mathbb{R}$, with gradient $\nabla\varphi_i : \mathbb{R}^w \mapsto \mathbb{R}^w$. Formally, $\mathcal{O}_i = \{\boldsymbol{x} \in \mathcal{X} \mid \varphi_i(\boldsymbol{v}(\boldsymbol{x})) < 0\}$. Note that $\varphi_i(\boldsymbol{v}(\boldsymbol{x})) < 0$ if the output $\boldsymbol{y}$ is in the open set $v(\mathcal{O}_i)$ and $\varphi_i(\boldsymbol{v}(\boldsymbol{x})) = 0$ if $\boldsymbol{y}$ is on its boundary $\partial v(\mathcal{O}_i)$.

The robot is equipped with a range sensor, such as a LiDAR scanner, and aims to follow a desired path, relying on the noisy distance measurements to avoid collisions. A *path* is a piece-wise continuous function $r : [0, 1] \mapsto \mathrm{Int}(v(\mathcal{S}))$. Let $\mathcal{F}(\boldsymbol{x}) \subset \mathcal{Y}$ be the field of view of the range sensor when the robot is in state $\boldsymbol{x}$. At discrete times t_k, for $k \in \mathbb{N}$, the range sensor provides a set of points $\mathcal{P}_{k,i} = \{\mathbf{p}_{k,i,j}\}_j \subset \mathcal{F}(\boldsymbol{x}(t_k)) \cap \partial v(\mathcal{O}_i)$ on the boundary of each obstacle $v(\mathcal{O}_i)$ which are within its field-of-view.

Problem: Consider the system in (1) operating in an unknown environment, i.e., the obstacles sets $\{v(\mathcal{O}_i)\}_{i=1}^K$ are unknown a priori. Given a desired path r and noisy range sensor measurements $\mathcal{P}_{k,i}$ for $k \in \mathbb{N}$ and $i = 1, \ldots, K$, design a feedback controller $\boldsymbol{u}(\boldsymbol{x})$ for (1) that ensures that the robot can move safely along the path r, i.e., $\boldsymbol{y}(t) \in v(\mathcal{S})$ for all t and $\boldsymbol{y}(t) \to r(1)$ as $t \to \infty$.

IV. Obstacle Estimation via Online Signed Distance Function Approximation

Throughout the paper, we rely on the concept of signed distance function to describe each unsafe region $\{v(\mathcal{O}_i)\}_{i=1}^K$. For each i, the SDF function $\varphi_i : \mathcal{Y} \mapsto \mathbb{R}$ is:

$$\varphi_i(\boldsymbol{y}) := \begin{cases} -d(\boldsymbol{y}, \partial v(\mathcal{O}_i)), & \boldsymbol{y} \in v(\mathcal{O}_i), \\ d(\boldsymbol{y}, \partial v(\mathcal{O}_i)), & \boldsymbol{y} \notin v(\mathcal{O}_i), \end{cases} \tag{2}$$

where d denotes the Euclidean distance between a point and a set. In this section, we describe an approach to construct approximations to the obstacle SDFs φ_i using the point cloud observations $\{\mathcal{P}_{k,i}\}_k$ at time step t_k.

A. Data Pre-Processing

Given the point cloud $\mathcal{P}_{k,i} \subset \mathbb{R}^w$ on the surface of the ith obstacle at time t_k, we can regard $\mathcal{P}_{k,i}$ as the points on the zero level-set of a distance function. To normalize the scale of the training data, we define the point coordinates with respect to the obstacle centroid. Since the centroid is unknown, we approximate it as the sample mean of the training points $\bar{\mathbf{p}}_{k,i} := \frac{1}{m}\sum_{j=1}^{m} \mathbf{p}_{k,i,j}$. The points $\mathbf{p}_{k,i,j} - \bar{\mathbf{p}}_{k,i}$ are centered around $\bar{\mathbf{p}}_{k,i}$ and have a measured distance of 0 to the obstacle surface $v(\partial\mathcal{O}_i)$. Let $c = \|v(\boldsymbol{x}_k) - \mathbf{q}_{k,i,j}\|$ be the Euclidean distance between the robot state $\boldsymbol{x}_k := \boldsymbol{x}(t_k)$ and $\mathbf{p}_{k,i,j}$. Let $\delta > 0$ be a small positive constant. Define a point $\mathbf{q}_{k,i,j} := \frac{\delta}{c} v(\boldsymbol{x}_k) + (1 - \frac{\delta}{c})\mathbf{p}_{k,i,j}$ along the LiDAR ray from the output $v(\boldsymbol{x}_k)$ to $\mathbf{p}_{k,i,j}$ that is approximately a distance δ from the obstacle surface $\partial v(\mathcal{O}_i)$. We call the set $\{\mathbf{q}_{k,i,j} - \bar{\mathbf{p}}_{k,i}\}_j$ a truncated SDF point set. The training set for each obstacle i is constructed as a union of the points on the boundary and the truncated SDF points. The training set at time t_k is $\mathcal{D}_{k,i} := \{(\mathbf{p}_{k,i,j} - \bar{\mathbf{p}}_{k,i}, 0)\} \cup \{(\mathbf{q}_{k,i,j} - \bar{\mathbf{p}}_{k,i}, \delta)\}$.

B. Loss Function

Inspired by the recent impressive results on multilayer perceptron approximation of SDF [7], [14], we introduce a fully connected neural network $\tilde{\varphi}_i(\boldsymbol{y}; \boldsymbol{\theta}_k)$ with parameters $\boldsymbol{\theta}_k$ to approximate the SDF of each observed obstacle i at time step t_k. Our approach relies on the fact that the norm of the SDF gradient satisfies the *Eikonal* equation $\|\nabla\varphi_i(\boldsymbol{y})\| = 1$ in its domain. We use a loss function that encourages $\tilde{\varphi}_i(\boldsymbol{y}; \boldsymbol{\theta}_k)$ to approximate the measured distances in a training set $\mathcal{D}$ (distance loss ℓ_i^D) and to satisfy the Eikonal constraint for set of points $\mathcal{D}'$ (Eikonal loss ℓ_i^E). For example, $\tilde{\varphi}_i$ equals 0 for points on the obstacle surface and equals δ for the truncated SDF points along the LiDAR rays. The loss function is defined as $\ell_i(\boldsymbol{\theta}_k; \mathcal{D}, \mathcal{D}') := \ell_i^D(\boldsymbol{\theta}_k; \mathcal{D}) + \lambda\ell_i^E(\boldsymbol{\theta}_k; \mathcal{D}')$, with a parameter $\lambda > 0$ and:

$$\begin{aligned} \ell_i^D(\boldsymbol{\theta}_k; \mathcal{D}) &:= \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{p},d)\in\mathcal{D}} |\tilde{\varphi}_i(\boldsymbol{p}; \boldsymbol{\theta}_k) - d|, \\ \ell_i^E(\boldsymbol{\theta}_k; \mathcal{D}') &:= \frac{1}{|\mathcal{D}'|} \sum_{\mathbf{p}\in\mathcal{D}'} (\|\nabla\tilde{\varphi}_i(\boldsymbol{p}; \boldsymbol{\theta}_k)\| - 1)\,. \end{aligned} \tag{3}$$

The training set $\mathcal{D}'$ for the Eikonal term can be generated arbitrarily in $\mathcal{Y}$ since $\tilde{\varphi}_i$ needs to satisfy the Eikonal constraint *almost everywhere in* $\mathcal{Y}$. In practice, we generate $\mathcal{D}'$ by sampling points from a mixture of a uniform distribution and Gaussians centered at the points in the distance training set $\mathcal{D}$ with standard deviation equal to the distance to the k-th nearest neighbor ($k = |\mathcal{D}|/2$). For the distance training set $\mathcal{D}$, ideally, we should use as much data as possible to obtain an accurate approximation of the SDF φ_i. For example, at time t_k, all observed data may be used as the distance training set $\mathcal{D} = \cup_{l=0}^{k}\mathcal{D}_{l,i}$. However, the online training time becomes longer and longer as the robot receives more and more LiDAR measurements, which makes it impractical for real-time mapping and navigation tasks. We introduce an "Incremental Training with Replay Memory" approach in Section IV-C, which obtains accurate SDF estimates with training times suitable for online learning.

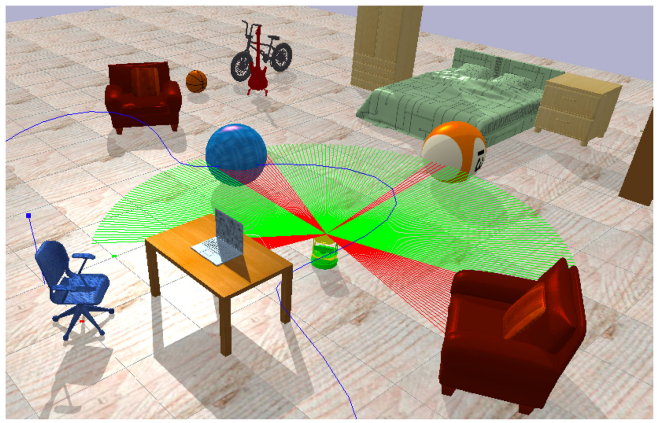


Fig. 1. A ground robot equipped with a LiDAR scanner is navigating safely along a desired path (blue curve) in an unknown room.

C. Incremental Learning

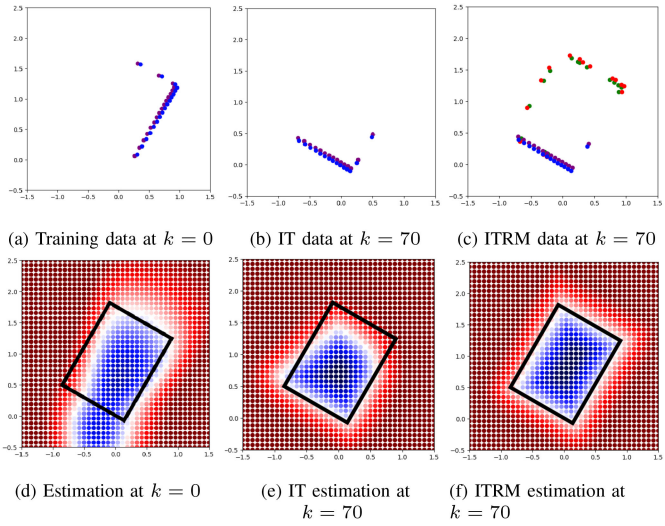
When an obstacle i is first observed at time t_0, we use the data set $\mathcal{D}_{0,i}$ to train the SDF network $\tilde{\varphi}_i(\boldsymbol{y}; \boldsymbol{\theta}_0)$. When new observations $\mathcal{D}_{k,i}$ of obstacle i are obtained at $k = 1, 2, \ldots$, we need to update the network parameters $\boldsymbol{\theta}_k$. We first consider an approach that updates $\tilde{\varphi}_i(\boldsymbol{y}; \boldsymbol{\theta}_{k-1})$ based on the new data set $\mathcal{D}_{k,i}$ and uses the previous parameters $\boldsymbol{\theta}_{k-1}$ as initialization. We call this approach "Incremental Training" (IT). Alternatively, we can update $\tilde{\varphi}_i(\boldsymbol{y}; \boldsymbol{\theta}_{k-1})$ using all prior data $\mathcal{D}_i = \cup_{l=0}^{k}\mathcal{D}_{l,i}$, observed up to time t_k, and $\boldsymbol{\theta}_{k-1}$ as initialization. We call this approach "Batch Training" (BT).

The IT approach is efficient because, at each time t_k, it uses data sets $\mathcal{D}_{k,i}$ of approximately constant cardinality, leading to approximately constant update times during online training. However, discarding the old data $\cup_{l=0}^{k-1}\mathcal{D}_{l,i}$ and using stochastic gradient descent to re-train the network parameters $\boldsymbol{\theta}_{k-1}$ on the new data $\mathcal{D}_{k,i}$ causes degradation in the neural network's ability to represent the old data. In other words, the SDF approximation $\tilde{\varphi}_i(\boldsymbol{y}; \boldsymbol{\theta}_k)$ at time t_k is good at approximating the latest observed obstacle surface but the approximation quality degrades at previously observed surfaces, as shown in Fig. 2. The BT approach does not have this limitation since it uses all data $\cup_{l=0}^{k}\mathcal{D}_{l,i}$ for training at time t_k but, as a result, its training time increases over time. Hence, our motivation for introducing an "Incremental Training with Replay Memory" (ITRM) approach is to balance the trade-off between SDF estimation error and online training time, making it suitable for real-time robotic tasks.

Experience replay is an effective and commonly used technique for online reinforcement learning, which enables convergence of stochastic gradient descent to a critical point in policy and value function approximation [10], [18]. This idea has not been explored for online supervised learning of geometric surfaces. The first contribution of this paper is to use replay memory for online incremental learning of SDF. At each time step t_{k-1}, We construct an experience replay memory $\mathcal{Q}_{k-1,i}$ at each time step by utilizing the SDF approximations $\tilde{\varphi}_i(\boldsymbol{y}; \boldsymbol{\theta}_{k-1})$.

Definition 1: The *replay memory* $\mathcal{Q}$ associated with a signed distance function φ and truncation parameter $\tau \geq 0$ is the set of points that are at most a distance τ from the zero-level set of φ: $\mathcal{Q} := \{(\mathbf{q}, \varphi(\mathbf{q})) \in (\mathbb{R}^w, \mathbb{R}) \mid |\varphi(\mathbf{q})| \leq \tau\}$.

To construct the replay memory set $\mathcal{Q}_{k-1,i}$, we need to generate samples from the level sets of the SDF approximation



(a) Training data at $k = 0$ (b) IT data at $k = 70$ (c) ITRM data at $k = 70$

(d) Estimation at $k = 0$ (e) IT estimation at $k = 70$ (f) ITRM estimation at $k = 70$

Fig. 2. Shape estimation with and without replay memory. The top row shows the training data used at time step $k = 0$ and $k = 70$ by the IT and ITRM approaches. The purple points are the observed LiDAR end points, while the blue points are the truncated SDF points along the LiDAR rays. In (c), the green and red points are boundary and truncated SDF points obtained from the replay memory. The bottom row shows the SDF estimation of the obstacle surface at time step $k = 0$ and $k = 70$ for the IT and ITRM approaches. The black rectangle shows the ground-truth obstacle boundary, while colored regions are level-sets of the SDF estimate. The white region denotes the estimated obstacle boundary. The blue (resp. red) region denotes negative (resp. positive) signed distance. IT and ITRM use the same data and lead to the same estimate at $k = 0$ because the replay memory set is empty. In (e), the SDF estimate of the top obstacle region at $k = 70$, without memory replay, degrades compared to (d). In (f), training with replay memory helps the neural network remember the overall obstacle shape.

$\tilde{\varphi}_i(\boldsymbol{y}; \boldsymbol{\theta}_{k-1})$. In robotics applications, where the environments are commonly 2-D or 3-D, samples from the level sets of $\tilde{\varphi}_i(\boldsymbol{y}; \boldsymbol{\theta}_{k-1})$ can be obtained using the Marching Cubes algorithm [19]. In our experiments in Section VI, we used Marching Cubes to extract samples $\mathbf{q}_0$ and $\mathbf{q}_\delta$ from the zero and δ level-sets of $\tilde{\varphi}_i(\boldsymbol{y}; \boldsymbol{\theta}_{k-1})$, respectively, and construct the replay memory as $\mathcal{Q}_{k-1,i} := \{(\mathbf{q}_0, 0)\} \cup \{(\mathbf{q}_\delta, \delta)\}$.

Given the replay memory $\mathcal{Q}_{k-1,i}$, our ITRM approach constructs a training set at time t_k by combining the latest observation $\mathcal{D}_{k,i}$ with a randomly sampled subset $\bar{\mathcal{Q}}_{k-1,i}$ from $\mathcal{Q}_{k-1,i}$. To make the algorithm efficient without losing significant information about prior data, we let $|\bar{\mathcal{Q}}_{k-1,i}| = |\mathcal{D}_{k,i}|$, i.e., we pick as many points from the replay memory as there are in the latest observation $\mathcal{D}_{k,i}$.

The three training techniques, IT, BT, and ITRM, discussed in this section, are formally defined as follows. In all approaches, for obstacle i, the parameters $\boldsymbol{\theta}_{k-1}$ obtained at the previous time step are used as an initial guess for the new parameters $\boldsymbol{\theta}_k$ at time t_k. The SDF approximations $\tilde{\varphi}_i(\boldsymbol{y}; \boldsymbol{\theta}_k)$ are trained via the loss function in (3) but each method uses a different training set $\mathcal{D}$:

$$\text{IT: } \mathcal{D} = \mathcal{D}_{k,i} \quad \text{BT: } \mathcal{D} = \cup_{l=0}^{k} \mathcal{D}_{l,i} \quad \text{ITRM: } \mathcal{D} = \mathcal{D}_{k,i} \cup \bar{\mathcal{Q}}_{k,i}.$$

We use the continuously differentiable Softplus function $\ln(1 + e^x)$ as the non-linear activation to ensure that $\tilde{\varphi}_i(\boldsymbol{y}; \boldsymbol{\theta}_k)$ is continuously differentiable. The gradient $\nabla \tilde{\varphi}_i(\boldsymbol{y}; \boldsymbol{\theta}_k)$ is an input in the loss function in (3) and can be calculated via backpropagation [14].

V. Safe Navigation With Estimated Obstacles

We rely on the estimated SDFs constructed in Section IV to formalize the synthesis of a controller that guarantees safety with respect to the exact obstacles, despite errors in the SDF approximation. Our analysis assumes error bounds are available, and we leave their actual computation for future work. In this regard, several recent works study the approximation power and error bounds of neural networks [20], [21]. In our evaluation in Section VI, we obtain SDF error bounds by comparing the estimated and ground-truth object SDFs.

A. Safe Control With Estimated Barrier Functions

A useful tool to ensure that the robot state remains in the safe set $\mathcal{S}$ throughout its evolution is the notion of control barrier function (CBF).

Definition 2 ([22]): A continuously differentiable function $h : \mathbb{R}^n \mapsto \mathbb{R}$, with $h(\boldsymbol{x}) > 0$ if $\boldsymbol{x} \in \text{Int}(\mathcal{S})$ and $h(\boldsymbol{x}) = 0$ if $\boldsymbol{x} \in \partial\mathcal{S}$, is a *zeroing control barrier function (ZCBF)* on $\mathcal{X} \subset \mathbb{R}^n$ if there exists an extended class $\mathcal{K}$ function α_h such that, for each $\boldsymbol{x} \in \mathcal{X}$, there exists $\boldsymbol{u} \in \mathcal{U}$ with

$$\mathcal{L}_f h(\boldsymbol{x}) + \mathcal{L}_g h(\boldsymbol{x}) \boldsymbol{u} + \alpha_h(h(\boldsymbol{x})) \geq 0, \tag{4}$$

where $\mathcal{L}_f h(\boldsymbol{x})$ is the Lie derivative of $h(\boldsymbol{x})$ along $f(\boldsymbol{x})$.

Any Lipschitz-continuous controller $\boldsymbol{u} : \mathcal{X} \mapsto \mathcal{U}$ such that $\boldsymbol{u}(\boldsymbol{x})$ satisfies (4) renders the set $\mathcal{S}$ forward invariant for the control-affine system (1) [6], [22]. If the exact SDFs φ_i describing the obstacles $\mathcal{O}_i$ were known, we could define ZCBFs $h_i(\boldsymbol{x}) := \varphi_i(v(\boldsymbol{x}))$ that ensure the forward invariance of $\mathcal{S}$. However, when the environment is observable through online range measurements as described in Section IV, we only have the estimated SDFs $\tilde{\varphi}_i(v(\boldsymbol{x}); \boldsymbol{\theta}_k)$ at our disposal to define $\tilde{h}_i(\boldsymbol{x}) := \tilde{\varphi}_i(v(\boldsymbol{x}); \boldsymbol{\theta}_k)$. Our next result describes how to use this information to ensure the safety of $\mathcal{S}$. The statement is for a generic $\tilde{h}$ (e.g., a single obstacle). We later particularize our discussion to the case of multiple obstacles.

Proposition 1: Let $e_1(\boldsymbol{x}) := h(\boldsymbol{x}) - \tilde{h}(\boldsymbol{x}) \in \mathbb{R}$ be the error at $\boldsymbol{x}$ in the approximation of h, and likewise, let $e_2(\boldsymbol{x}) := \nabla h(\boldsymbol{x}) - \nabla \tilde{h}(\boldsymbol{x}) \in \mathbb{R}^n$ be the error at $\boldsymbol{x}$ in the approximation of its gradient. Assume there are available known functions $e_h(\boldsymbol{x}) : \mathbb{R} \mapsto \mathbb{R}_{\geq 0}$ and $e_{\nabla h}(\boldsymbol{x}) : \mathbb{R}^n \mapsto \mathbb{R}_{\geq 0}$ such that

$$|e_1(\mathbf{x})| \leq e_h(\boldsymbol{x}), \quad \|e_2(\mathbf{x})\| \leq e_{\nabla h}(\boldsymbol{x}) \tag{5}$$

with $e_h(\boldsymbol{x}) \to 0$ and $e_{\nabla h}(\boldsymbol{x}) \to 0$ as $\boldsymbol{x} \to \partial\mathcal{S}$. Let

$$\mathcal{K}_{\tilde{h}}(\boldsymbol{x}) := \{\boldsymbol{u} \in \mathcal{U} \mid \mathcal{L}_f \tilde{h}(\boldsymbol{x}) + \mathcal{L}_g \tilde{h}(\boldsymbol{x}) \boldsymbol{u} - \|f(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u}\| e_{\nabla h}(\boldsymbol{x}) + \alpha_h(\tilde{h}(\boldsymbol{x}) - e_h(\boldsymbol{x})) \geq 0\}. \tag{6}$$

Then, any locally Lipschitz continuous controller $\boldsymbol{u} : \mathcal{X} \mapsto \mathcal{U}$ such that $\boldsymbol{u}(\boldsymbol{x}) \in \mathcal{K}_{\tilde{h}}(\boldsymbol{x})$ guarantees that the safe set $\mathcal{S}$ is forward invariant.

Proof: We start by substituting $h(\boldsymbol{x}) = \tilde{h}(\boldsymbol{x}) + e_1(\boldsymbol{x})$ in (4):

$$\mathcal{L}_f \tilde{h}(\boldsymbol{x}) + \mathcal{L}_g \tilde{h}(\boldsymbol{x}) \boldsymbol{u} + e_2(\boldsymbol{x})^\top f(\boldsymbol{x}) + e_2(\boldsymbol{x})^\top g(\boldsymbol{x}) \boldsymbol{u} \geq -\alpha_h(\tilde{h}(\boldsymbol{x}) + e_1(\boldsymbol{x})).$$

For any fixed $\boldsymbol{x}$ and any errors $e_1(\boldsymbol{x})$ and $e_2(\boldsymbol{x})$ satisfying (5), we need the minimum value of the left-hand side greater than the maximum value of the right-hand side to ensure that (4) still

holds, namely:

$$\min_{\|e_2(\boldsymbol{x})\|\leq e_{\nabla h}(\boldsymbol{x})} \Big\{\mathcal{L}_f\tilde{h}(\boldsymbol{x}) + \mathcal{L}_g\tilde{h}(\boldsymbol{x})\boldsymbol{u} + e_2(\boldsymbol{x})^\top f(\boldsymbol{x}) + e_2(\boldsymbol{x})^\top g(\boldsymbol{x})\boldsymbol{u}\Big\} \geq \max_{|e_1(\boldsymbol{x})|\leq e_h(\boldsymbol{x})} \Big\{-\alpha_h(\tilde{h}(\boldsymbol{x}) + e_1(\boldsymbol{x}))\Big\}. \tag{7}$$

Note that since $e_h(\boldsymbol{x}) \geq 0$ and α_h is an extended class $\mathcal{K}$ function, the maximum value in (7) is obtained by:

$$\max_{|e_1(\boldsymbol{x})|\leq e_h(\boldsymbol{x})} -\alpha_h(\tilde{h}(\boldsymbol{x}) + e_1((\boldsymbol{x}))) = -\alpha_h(\tilde{h}(\boldsymbol{x}) - e_h(\boldsymbol{x})).$$

The minimum value is attained when $e_2(\boldsymbol{x})$ is in the opposite direction to the gradient of $f(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u}$, namely

$$\min_{\|e_2(\boldsymbol{x})\|\leq e_{\nabla h}(\boldsymbol{x})} \big\{e_2(\boldsymbol{x})^\top f(\boldsymbol{x}) + e_2(\boldsymbol{x})^\top g(\boldsymbol{x})\boldsymbol{u}\big\} = -\|f(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u}\| e_{\nabla h}(\boldsymbol{x}).$$

Therefore, the inequality condition (7) can be rewritten as $\mathcal{L}_f\tilde{h}(\boldsymbol{x}) + \mathcal{L}_g\tilde{h}(\boldsymbol{x})\boldsymbol{u} - \|f(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u}\| e_{\nabla h}(\boldsymbol{x}) \geq -\alpha_h(\tilde{h}(\boldsymbol{x}) - e_h(\boldsymbol{x}))$, which is equivalent to the condition in (6). ■

Proposition 1 allows us to synthesize safe controllers even though the obstacles are not exactly known, provided that error bounds on the approximation of the barrier function and its gradient are available and get better as the robot state gets closer to the boundary of the safe set $\mathcal{S}$.

B. Control Synthesis Via Second-Order Cone Programming

We encode the control objective using the notion of a Lyapunov function. Formally, we assume the existence of a control Lyapunov function, as defined next.

Definition 3 ([6]): A *control Lyapunov function (CLF)* for the system dynamics in (1) is a continuously differentiable function $V : \mathbb{R}^n \mapsto \mathbb{R}_{\geq 0}$ for which there exist a class $\mathcal{K}$ function α_V such that, for all $\boldsymbol{x} \in \mathcal{X}$:

$$\inf_{\boldsymbol{u}\in\mathcal{U}} [\mathcal{L}_f V(\boldsymbol{x}) + \mathcal{L}_g V(\boldsymbol{x})\boldsymbol{u} + \alpha_V(V(\boldsymbol{x}))] \leq 0.$$

The function V may be used to encode a variety of control objectives, including for instance path following. We present a specific Lyapunov function for this purpose in Sec. VI.

When the barrier function is precisely known, one can combine CLF and CBF constraints to synthesize a safe controller via the following QP:

$$\begin{aligned} \min_{\boldsymbol{u}\in\mathcal{U},\delta\in\mathbb{R}} \ & \|L(\boldsymbol{x})^\top(\boldsymbol{u} - \tilde{\boldsymbol{u}}(\boldsymbol{x}))\|^2 + \lambda\delta^2 \\ \text{s.t. } & \mathcal{L}_f V(\boldsymbol{x}) + \mathcal{L}_g V(\boldsymbol{x})\boldsymbol{u} + \alpha_V(V(\boldsymbol{x})) \leq \delta \\ & \mathcal{L}_f h(\boldsymbol{x}) + \mathcal{L}_g h(\boldsymbol{x})\boldsymbol{u} + \alpha_h(h(\boldsymbol{x})) \geq 0, \end{aligned} \tag{8}$$

where $\tilde{\boldsymbol{u}}(\boldsymbol{x})$ is a baseline controller, $L(\mathbf{x})$ is a matrix penalizing control effort, and $\delta \geq 0$ (with the corresponding penalty λ) is a slack variable, introduced to relax the CLF constraints in order to ensure the feasibility of the QP. The baseline controller $\tilde{\boldsymbol{u}}(\boldsymbol{x})$ is used to specify additional control requirements such as desirable velocity or orientation (see Section VI) but may be set to $\tilde{\boldsymbol{u}}(\boldsymbol{x}) \equiv 0$ if minimum control effort is the main objective. The QP formulation in (8) modifies $\tilde{\boldsymbol{u}}(\boldsymbol{x})$ online to ensure safety and stability via the CBF and CLF constraints.

Without exact knowledge of the barrier function h, we need to replace the CLF constraint in (8) by (6):

$$\begin{aligned} \min_{\boldsymbol{u}\in\mathcal{U},\delta\in\mathbb{R}} \ & \|L(\boldsymbol{x})^\top(\boldsymbol{u} - \tilde{\boldsymbol{u}}(\boldsymbol{x}))\|^2 + \lambda\delta^2 \\ \text{s.t. } & \mathcal{L}_f V(\boldsymbol{x}) + \mathcal{L}_g V(\boldsymbol{x})\boldsymbol{u} + \alpha_V(V(\boldsymbol{x})) \leq \delta \\ & \mathcal{L}_f\tilde{h}(\boldsymbol{x}) + \mathcal{L}_g\tilde{h}(\boldsymbol{x})\boldsymbol{u} + \alpha_h(\tilde{h}(\boldsymbol{x}) - e_h(\boldsymbol{x})) \\ & \qquad \geq \|f(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u}\| e_{\nabla h}(\boldsymbol{x}). \end{aligned} \tag{9}$$

This makes the optimization problem in (9) no longer a quadratic program. However, the following result shows that (9) is a (convex) second-order cone program (SOCP).

Proposition 2: The optimization problem in (9) is equivalent to the following second-order cone program:

$$\begin{aligned} \min_{\boldsymbol{u}\in\mathcal{U},\delta\in\mathbb{R},l\in\mathbb{R}} \ & l \\ \text{s.t. } & \mathcal{L}_f V(\boldsymbol{x}) + \mathcal{L}_g V(\boldsymbol{x})\boldsymbol{u} + \alpha_V(V(\boldsymbol{x})) \leq \delta \\ & \|f(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u}\| e_{\nabla h}(\boldsymbol{x}) \leq \mathcal{L}_f\tilde{h}(\boldsymbol{x}) + \mathcal{L}_g\tilde{h}(\boldsymbol{x})\boldsymbol{u} \\ & + \alpha_h(\tilde{h}(\boldsymbol{x}) - e_h(\boldsymbol{x})) \\ & \left\| \begin{bmatrix} 2L(\boldsymbol{x})^\top(\boldsymbol{u} - \tilde{\boldsymbol{u}}(\boldsymbol{x})) \\ 2\sqrt{\lambda}\delta \\ l - 1 \end{bmatrix} \right\| \leq l + 1. \end{aligned} \tag{10}$$

Proof: We first introduce a new variable l so that the problem in (9) is equivalent to

$$\begin{aligned} \min_{\boldsymbol{u}\in\mathcal{U},\delta\in\mathbb{R},l\in\mathbb{R}} \ & l \\ \text{s.t. } & \mathcal{L}_f V(\boldsymbol{x}) + \mathcal{L}_g V(\boldsymbol{x})\boldsymbol{u} + \alpha_V(V(\boldsymbol{x})) \leq \delta \\ & \|f(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u}\| e_{\nabla h}(\boldsymbol{x}) \leq \mathcal{L}_f\tilde{h}(\boldsymbol{x}) + \mathcal{L}_g\tilde{h}(\boldsymbol{x})\boldsymbol{u} \\ & + \alpha_h(\tilde{h}(\boldsymbol{x}) - e_h(\boldsymbol{x})) \\ & \|L(\boldsymbol{x})^\top(\boldsymbol{u} - \tilde{\boldsymbol{u}}(\boldsymbol{x}))\|^2 + \lambda\delta^2 \leq l. \end{aligned} \tag{11}$$

The last constraint in (11) corresponds to a rotated second-order cone, $\mathcal{Q}^n_{rot} := \{(\boldsymbol{x}_r, y_r, z_r) \in \mathbb{R}^{n+2} \mid \|\boldsymbol{x}_r\|^2 \leq y_r z_r, y_r \geq 0, z_r \geq 0\}$, which can be converted into a standard SOC constraint [23]:

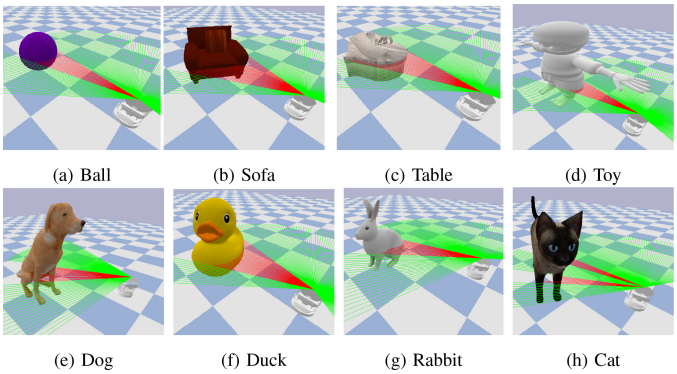
$$\left\| \begin{bmatrix} 2\boldsymbol{x}_r \\ y_r - z_r \end{bmatrix} \right\| \leq y_r + z_r.$$

Let $y_r = l$, $z_r = 1$ and consider the constraint $\|L(\boldsymbol{x})^\top(\boldsymbol{u} - \tilde{\boldsymbol{u}}(\boldsymbol{x}))\|^2 + \lambda\delta^2 \leq l$. Multiplying both sides by 4 and adding $l^2 + 1$, makes the constraint equivalent to

$$4\|L(\boldsymbol{x})^\top(\boldsymbol{u} - \tilde{\boldsymbol{u}}(\boldsymbol{x}))\|^2 + 4\lambda\delta^2 + (l-1)^2 \leq (l+1)^2.$$

Taking a square root on both sides, we end up with $\sqrt{\|2L(\boldsymbol{x})^\top(\boldsymbol{u} - \tilde{\boldsymbol{u}}(\boldsymbol{x}))\|^2 + (2\sqrt{\lambda}\delta)^2 + (l-1)^2} \leq l + 1$, which is equivalent to the third constraint in (10). ■

For multiple obstacles in the environment, one can add multiple CBF constraints to (10). We leave for future work the characterization of the Lipschitz continuity properties of the controller resulting from (10).



(a) Ball (b) Sofa (c) Table (d) Toy

(e) Dog (f) Duck (g) Rabbit (h) Cat

Fig. 3. Object instances used to evaluate the estimation performance of online signed distance function approximation.

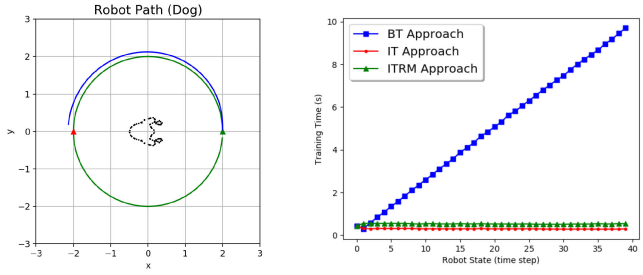


Fig. 4. Robot motion (left) and training time (right) for the SDF estimation experiment in Section VI-B. The robot starts at (2,0) and follows a circular reference path (green) to a goal position at $(-2, 0)$ (red triangle). The actual trajectory followed by the robot is shown in blue. The black dots are points on the surface of a ground-truth Dog object and vary for different object instances. The average training times for the BT, IT, and ITRM methods for SDF approximation of the 8 objects in Fig. 3 are compared.

VI. Evaluation

We use the PyBullet simulator [24] to evaluate our approach for online shape estimation and safe navigation. We use a TurtleBot, equipped with a LiDAR scanner with a 270 ° field of view, 150 rays per scan, 3 m range, and zero-mean Gaussian measurement noise with standard deviation $\sigma = 0.01$. The simulation environments contain obstacles with various shapes, a priori unknown to the robot.

A. Modeling

We model the robot motion using unicycle kinematics and take a small distance $a \neq 0$ off the wheel axis as in [25] to obtain a relative-degree-one model:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \psi \cos(\theta) - a\omega \sin(\theta) \\ \psi \sin(\theta) + a\omega \cos(\theta) \\ \omega \end{bmatrix}, \tag{12}$$

where ψ, ω represent the robot linear and angular velocity, respectively. The state, input, and output are $\boldsymbol{x} := [x, y, \theta]^\top \in \mathbb{R}^2 \times [-\pi, \pi)$, $\boldsymbol{u} := [\psi, \omega]^\top \in \mathbb{R}^2$, and $\boldsymbol{y} = v(\boldsymbol{x}) := [x, y] \in \mathbb{R}^2$. We use a baseline controller $\tilde{\boldsymbol{u}}(\boldsymbol{x}) \equiv [\psi_{max}, 0]^\top$ to encourage the robot to drive at max velocity ψ_{max} in a straight line whenever possible. The desired robot path is specified by a 2-D curve $r(\gamma)$, $\gamma \in [0, 1]$. To capture the path-following task via a CLF constraint, we define $\boldsymbol{\eta}(\boldsymbol{x}) := v(\boldsymbol{x}) - r(\gamma(\boldsymbol{x}))$. The closest point on the reference path $r(\gamma)$ to the robot state $\boldsymbol{x}$ is obtained by $\gamma(\boldsymbol{x}) := \arctan(y/x)/(2\pi) \in [0, 1]$. According to [26], the following is a valid CLF for path following:

$$V(\boldsymbol{x}) = [\boldsymbol{\eta}^\top(\boldsymbol{x}), \dot{\boldsymbol{\eta}}^\top(\boldsymbol{x})] P [\boldsymbol{\eta}^\top(\boldsymbol{x}), \dot{\boldsymbol{\eta}}^\top(\boldsymbol{x})]^\top, \tag{13}$$

where P is a positive-definite matrix calculated by solving the Lyapunov equation of the input-output linearization.

B. SDF Estimation Results

We compare the training time and prediction accuracy of the proposed ITRM approach for SDF estimation versus the IT and BT approaches described in Section IV-C for various 2-D obstacle shapes. Figs. 3 and 4 show our experiment setup. We used the multilayer perceptron architecture proposed by Gropp *et al.* [14] and Park *et al.* [7] for ITRM training. The details of the neural network architecture are specified in Section VI-C.

TABLE I
SDF Estimation Error (14) of the IT, ITRM, and BT Training Methods for LiDAR Measurements With Zero-Mean Gaussian Range Noise With Standard Deviation $\sigma = 0.01$

Case	$\mathcal{E}$ in IT	$\mathcal{E}$ in ITRM	$\mathcal{E}$ in BT
1 (Ball)	0.0287	0.0148	**0.0140**
2 (Sofa)	0.0653	**0.0212**	0.0215
3 (Table)	0.0564	0.0230	**0.0227**
4 (Toy)	0.1064	0.0242	**0.0198**
5 (Dog)	0.0328	0.0211	**0.0123**
6 (Duck)	0.0751	**0.0102**	0.0111
7 (Rabbit)	0.0603	0.0134	**0.0109**
8 (Cat)	0.0292	0.0149	**0.0123**
Average	0.0568	0.0179	**0.0147**

We measure SDF approximation error as:

$$\mathcal{E} = \frac{1}{m} \sum_{i=1}^{m} |\tilde{\varphi}(\boldsymbol{y}_i; \boldsymbol{\theta}_k)|, \tag{14}$$

where $\{\boldsymbol{y}_i\}_{i=1}^m$ are $m = 500$ points uniformly sampled on the surface of the ground-truth object. The SDF error is shown in Table I for the eight object instances in Fig. 3. The error of our ITRM approach is comparable with the error of the BT approach and is much smaller than the error of the IT approach. The training update time is the time needed for updating the network parameters from $\boldsymbol{\theta}_{k-1}$ to $\boldsymbol{\theta}_k$. The average training update time across the 8 object instances for the three methods is shown in Fig. 4. We see that as the robot moves around the environment, the average training update time of the ITRM approach remains at about 0.7 s while the BT approach requires more and more time for training.

C. Network Architecture and Training Implementation

To accelerate the training time of the ITRM approach further to support online navigation, we explore the trade-off between training time and accuracy for different neural network configurations. The baseline neural network used in Section VI-B has 8 fully connected layers with 512 neurons each and a single skip connection from the input to the middle layer. All internal layers use Softplus nonlinear activations. We set the parameter $\lambda = 0.1$ in (3). At each time step t_k, we trained the network for 16 epochs with the ADAM optimizer [27] with constant learning rate of 0.001. We also experimented with neural network configurations

TABLE II
TRAINING TIME AND VALIDATION ERROR OF DIFFERENT NEURAL NETWORK CONFIGURATIONS FOR SDF APPROXIMATION. THE NUMBER OF NETWORK LAYERS, THE NUMBER OF TRAINING EPOCHS, AND WHETHER A GPU IS USED ARE VARIED. THE RESULTS ARE OBTAINED USING THE SETTINGS IN TABLE I FOR THE DOG OBJECT SHOWN IN FIG. 3. THE SDF TRAINING TIME INCLUDES BOTH PRE-PROCESSING OF THE LIDAR DATA AND NEURAL NETWORK TRAINING. WE REPORT THE AVERAGE TIME PER LIDAR SCAN ALONG THE ROBOT PATH. THE SDF VALIDATION ERROR IS COMPUTED VIA (14)

Training Setting (number of layer, training epochs)	Average Training Time		SDF Validation Error	
	GPU	CPU	GPU	CPU
4, 5	**0.0744**	0.197	0.0212	0.0266
4, 10	0.143	0.388	0.0215	0.0207
4, 16	0.224	0.716	0.0207	0.0218
6, 5	0.0978	0.407	**0.0132**	0.0215
6, 10	0.193	0.731	0.0136	0.0193
6, 16	0.312	1.258	0.0149	0.0158
8, 5	0.178	0.544	0.0181	0.0208
8, 10	0.325	1.127	0.0201	0.0184
8, 16	0.497	1.841	0.0189	0.0156

with fewer layers (4 or 6) and different number of training epochs (5, 10, or 16), while keeping fixed the rest of the architecture. As real-time navigation necessitates obstacle shape estimation to be as quick as possible, we aim to obtain the smallest network architecture, trained with as few epochs as possible, that still maintains good SDF prediction accuracy. We report training time and accuracy results using a GPU (one Nvidia Geforce 2080 Super) or a CPU (Intel i7 9700 K) in Table II. The best trade-off between training time and SDF error for a 2-D shape is obtained using a GPU to train a 6 layer network for 5 epochs. This configuration enables training times of less than 0.1 s, suitable for real-time navigation.

D. Safe Navigation Results

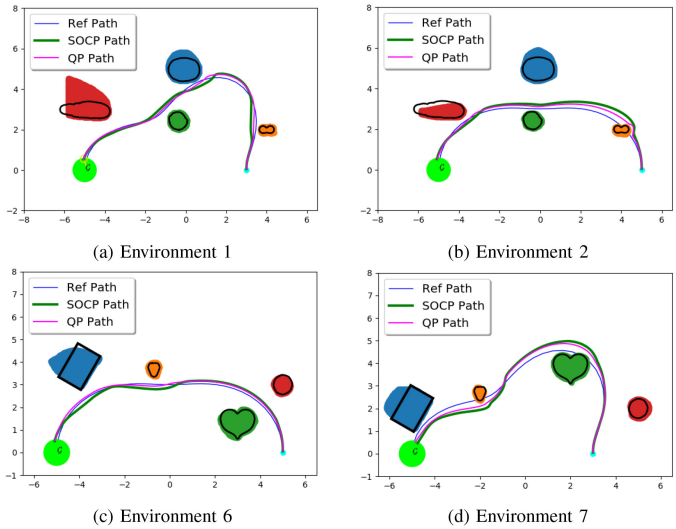
The second set of experiments demonstrates safe trajectory tracking using online SDF obstacle estimates to define constraints in the CLF-CBF-SOCP control synthesis optimization in (10). The robot moves along a reference path while avoiding unknown obstacles in 8 different environments, similar to the one shown in Fig. 1. To account for the fact that the robot body is not a point mass, we subtract the robot radius $\tau = 0.177$ from the SDF estimate when defining the CBF: $\tilde{h}_i(\boldsymbol{x}) = \tilde{\varphi}_i(\boldsymbol{y};\boldsymbol{\theta}) - \tau$. The error bounds of CBF $e_h(\boldsymbol{x})$ and its gradient $e_{\nabla h}(\boldsymbol{x})$ are approximated based on Table I. We compare our CLF-CBF-SOCP approach to a CLF-CBF-QP. To account for estimation errors, it is possible to inflate the CBFs in the CLF-CBF-QP formulation by both the robot radius and the SDF approximation error, $\tilde{h}_i(\boldsymbol{x}) = \tilde{\varphi}_i(\boldsymbol{y};\boldsymbol{\theta}) - \tau - e_h(\boldsymbol{x})$. This preserves linearity of the CBF constraints and leads to a more conservative controller. Comparing with (10), we can see that the CLF-CBF-SOCP formulation accounts for both direct and gradient errors in the CBF approximations, and naturally reduces to a QP if $e_{\nabla h}(\boldsymbol{x})$ is zero. To emphasize the importance of accounting for estimation errors, we compare to a CLF-CBF-QP that assumes the estimated CBFs $\tilde{h}_i(\boldsymbol{x})$ are accurate and ignores the estimation errors.

To avoid low velocities, we set diagonal of $L(\mathbf{x})$ in Section V-B as $l_1 = 10, l_2 = 1, l_3 = 10\sqrt{10}$, where l_1, l_2, l_3 are the penalty parameters for linear velocity, angular velocity and path following, respectively. If there is no solution found at some time step, l_1 is divided by $\sqrt{2}$ until a feasible solution found. We use the Fréchet distance between the reference path r and the paths produced by the CLF-CBF-SOCP and CLF-CBF-QP controllers to evaluate their similarities. For paths A, B, the Fréchet distance is computed by:

$$F(A,B) = \inf_{\alpha,\beta} \max_{t\in[0,1]} \{d(A(\alpha(t)), B(\beta(t)))\}, \qquad (15)$$

where $\alpha, \beta : [0,1] \mapsto [0,1]$ are continuous, non-decreasing, surjections and d is the Euclidean distance in our case.

In Fig. 5, the robot can follow the prescribed reference path if no obstacles are nearby. The green trajectory generated by CLF-CBF-SOCP is always more conservative than the pink trajectory since it takes the errors in the CBF estimation into account. When there is an obstacle near or on the reference path, the robot controlled by the CLF-CBF-SOCP controller stays further away from the obstacles than the robot controlled by the CLF-CBF-QP controller. This agrees with the Fréchet distance results presented in Table III. In Fig. 6, we see that the CLF-CBF-QP controller sometimes fails to avoid obstacles because it does



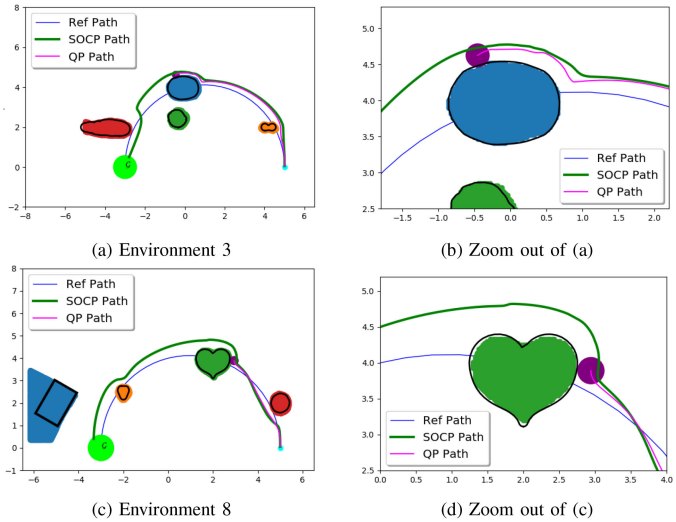
(a) Environment 1 (b) Environment 2

(c) Environment 6 (d) Environment 7

Fig. 5. Simulation results for the CLF-CBF-SOCP and CLF-CBF-QP controllers. The reference path is shown in blue. The ground-truth obstacle surfaces are shown in black. The estimated obstacles, obtained after the whole path is traversed by the CLF-CBF-SOCP controller are shown in different colors (red, green, blue, orange). The trajectory generated by CLF-CBF-SOCP is shown in green, while the trajectory generated by CLF-CBF-QP is shown in pink. The starting point is cyan and the goal region is a light green circle.

TABLE III
FRÉCHET DISTANCE BETWEEN THE REFERENCE PATH AND THE ROBOT TRAJECTORIES GENERATED BY THE CLF-CBF-SOCP AND THE CLF-CBF-QP CONTROLLERS (SMALLER VALUES INDICATE LARGER TRAJECTORY SIMILARITY, N/A INDICATES THAT THE ROBOT FAILED TO REACH THE GOAL REGION)

Environment	CLF-CBF-SOCP	CLF-CBF-QP
1	**0.3428**	0.3442
2	0.5339	**0.3971**
3	0.8356	N/A
4	0.3968	**0.3492**
5	1.0535	N/A
6	0.3611	**0.3444**
7	0.4643	**0.3493**
8	0.8674	N/A



(a) Environment 3 (b) Zoom out of (a)

(c) Environment 8 (d) Zoom out of (c)

Fig. 6. Simulation results in environments where the CLF-CBF-SOCP controller succeeds but CLF-CBF-QP one fails. The robot is shown as a purple circle when crashing into an obstacle using the CLF-CBF-QP controller. Fig. 6(b) and Fig. 6(d) show enlargements of the crash regions in Fig. 6(a) and Fig. 6(c), respectively.

not consider the errors in the CBF estimation. In contrast, the CLF-CBF-SOCP controller is guaranteed by Prop. 2 to remain safe if the CBF estimation is captured correctly in the SOC constraints.

In summary, Table III indicates that the trajectory mismatch with respect to the reference path is larger under our CLF-CBF-SOCP controller than under the CLF-CBF-QP controller if they both succeed. However, our approach guarantees safe navigation, while the CLF-CBF-QP controller sometimes causes collisions due to errors in CBF estimation.

VII. Conclusion

We introduced an incremental training approach with replay memory to enable online estimation of signed distance functions and construction of corresponding safety constraints in the form of control barrier functions. The use of replay memory balances training time with estimation error, making our approach suitable for real-time estimation. We showed that accounting for the direct and gradient errors in the CBF approximations leads to a new CLF-CBF-SOCP formulation for safe control synthesis. Future work will consider capturing localization and robot dynamics errors in the safe control formulation and will apply our techniques to real autonomous navigation experiments.

References

[1] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Robot. Res.*, vol. 5, no. 1, pp. 90–98, 1986.

[2] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Trans. Robot. Automat.*, vol. 8, no. 5, pp. 501–518, Oct. 1992, doi: 10.1109/70.163777.

[3] S. Prajna, "Barrier certificates for nonlinear model validation," in *Proc. Conf. Decis. Control*, 2003, pp. 2884–2889.

[4] S. Prajna and A. Jadbabaie, "Safety verification of hybrid systems using barrier certificates," *Hybrid Systems: Computation and Control*. Berlin Heidelberg: Springer, 2004, pp. 477–492.

[5] P. Wieland and F. Allgöwer, "Constructive safety using control barrier functions," *IFAC Proc. Volumes*, vol. 40, no. 12, pp. 462–467, 2007.

[6] A. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *Proc. Eur. Control Conf.*, 2019, pp. 3420–3431.

[7] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "DeepSDF: Learning continuous signed distance functions for shape representation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 165–174.

[8] C.-H. Lin, C. Wang, and S. Lucey, "SDF-SRN: *Learn. Signed Distance 3D Object Reconstruction from Static Images*," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2020.

[9] L. Han, F. Gao, B. Zhou, and S. Shen, "Fiesta: Fast incremental euclidean distance fields for online motion planning of aerial robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Macau, China, 2019, pp. 4423–4430, doi: 10.1109/IROS40897.2019.8968199.

[10] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. Int. Conf. Learn. Representations*, San Juan, PR, USA, May 2016, pp. 1–21.

[11] A. Hornung, K. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3D mapping framework based on octrees," *Auton. Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[12] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3 d euclidean signed distance fields for on-board mav planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 1366–1373.

[13] L. M. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, "Occupancy networks: Learning 3 d reconstruction in function space," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4455–4465.

[14] A. Gropp, L. Yariv, N. Haim, M. Atzmon, and Y. Lipman, "Implicit geometric regularization for learning shapes," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 3569–3579.

[15] P. Glotfelter, J. Cortés, and M. Egerstedt, "Nonsmooth barrier functions with applications to multi-robot systems," *IEEE Contr. Syst. Lett.*, vol. 1, no. 2, pp. 310–315, Oct. 2017.

[16] X. Xu *et al.*, "Realizing simultaneous lane keeping and adaptive speed regulation on accessible mobile robot testbeds," in *Proc. IEEE Conf. Control Technol. Appl.*, 2017, pp. 1769–1775.

[17] M. Srinivasan, A. Dabholkar, S. Coogan, and P. Vela, "Synthesis of control barrier functions using a supervised machine learning approach," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2020, pp. 7139–7145.

[18] D. Rolnick, A. Ahuja, J. Schwarz, T. Lillicrap, and G. Wayne, "Experience replay for continual learning," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 350–360.

[19] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *Comput. Graph.*, vol. 21, no. 4, pp. 163–169, 1987.

[20] B. Bailey, Z. Ji, M. Telgarsky, and R. Xian, "Approximation power of random neural networks," 2019, *arXiv:abs/1906.07709*.

[21] D. Yarotsky, "Error bounds for approximations with deep relu networks," *Neural Netw.*, vol. 94, pp. 103–114, 2017.

[22] A. Ames, X. Xu, J. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs for safety critical systems," *IEEE Trans. Autom. Control*, vol. 62, no. 8, pp. 3861–3876, Aug. 2017.

[23] F. Alizadeh and D. Goldfarb, "Second-order cone programming," *Math. Program.*, vol. 95, no. 1, pp. 3–51, 2003.

[24] E. Coumans and Y. Bai, "PyBullet, a Python module for physics simulation for games, robotics and machine learning," 2016, pp. 2016–2017. [Online]. Available: http://pybullet.org.

[25] J. Cortés and M. Egerstedt, "Coordinated control of multi-robot systems: A survey," *SICE J. Control, Meas., Syst. Integration*, vol. 10, no. 6, pp. 495–503, 2017.

[26] A. D. Ames, K. Galloway, K. Sreenath, and J. W. Grizzle, "Rapidly exponentially stabilizing control Lyapunov functions and hybrid zero dynamics," *IEEE Trans. Autom. Control*, vol. 59, no. 4, pp. 876–891, Apr. 2014.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.