Point Cloud-Based Control Barrier Function Regression for Safe and Efficient Vision-Based Control

Massimiliano de Sa, Prasanth Kotaru, and Koushil Sreenath

***Abstract*— Control barrier functions have become an increasingly popular framework for safe real-time control. In this work, we present a computationally low-cost framework for synthesizing barrier functions over point cloud data for safe vision-based control. We take advantage of surface geometry to locally define and synthesize a quadratic CBF over a point cloud. This CBF is used in a CBF-QP for control and verified in simulation on quadrotors and in hardware on quadrotors and the TurtleBot3. This technique enables safe navigation through unstructured and dynamically changing environments and is shown to be significantly more efficient than current methods.**

I. Introduction

A. Motivation

Safety-critical systems are all around us in the modern world, from autonomous cars to aerial systems. In recent years, Control Barrier Functions (CBFs) [1] have proven to be a versatile method for designing provably safe controllers for such systems, and have demonstrated strong performance in tasks such as safety-critical control and obstacle avoidance.

The ability to deploy vision-in-the-loop control in a safety-critical manner is extremely valuable. In aerial robotics, for instance, safe vision-based navigation is vital to applications such as construction-site surveying and search & rescue.

Due to the complexity of vision-based data and the challenge of dealing with unstructured environments, performing safe and efficient vision-in-the-loop control remains an open challenge. To efficiently maintain safety for vision-in-the-loop systems, we aim to formulate control barrier functions over depth data with minimal processing.

B. Prior Work

Traditionally, the problem of processing vision data for safe control is performed by generating a metric map of the environment. However, mapping using classical SLAM techniques such as the occupancy grid [2] involves slow, computationally intensive processing of vision data, thus proving ineffective for agile and resource-constrained robots.

Vision-based control barrier functions have been explored as a means of developing verifiable vision-based control in recent work [3]–[6]. For instance, an online learning-based CBF controller for static environments was designed based on stereo depth data [3]. However, this approach relies on the assumption that the environment is static in time and requires computationally expensive training.

Authors are with the Department of Mechanical Engineering, University of California, Berkeley, CA, 94703, United States. {mz.desa, prasanth.kotaru, koushils}@berkeley.edu

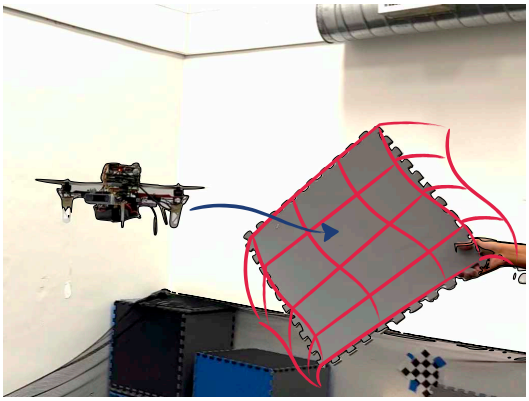

Fig. 1. Resource-constrained systems need an efficient way to verify the safety of their actions in unstructured environments. By thinking about depth data as representing a surface in space, we may efficiently formulate depth-based CBFs to ensure no collisions with the environment occur.

Control barrier functions were also formulated over neural radiance fields (NeRFs) for safe vision-based control [5]. However, this approach was largely limited by computation speed, achieving a limited control frequency of 10 Hz due to the heavy computations associated with the NeRF. Additionally, the NeRF requires a static environment.

Many practical systems in robotics face a unique combination of being computationally constrained and requiring a high control frequency. Furthermore, the ability to adapt to environments that evolve in time is crucial for many systems. Current safe vision-based control techniques are inadequate for platforms like quadrotors, where computational power is limited, but fast control and rapid adaptation to changing environments are crucial.

To provide a safe and efficient technique for the vision-based control of resource-constrained systems, we propose Depth-CBF, a technique which formulates control barrier functions over point clouds. This technique requires no training and minimal computation while allowing for the safe exploration of unstructured and dynamic environments.

C. Contributions

Our contributions in this work are summarized as follows:

1) We present Depth-CBF, a novel technique for formulating vision-based control barrier functions and estimating their Lie derivatives for control affine systems.
2) We detail the design of depth-based barrier function controllers with only a single optimization constraint.
3) We validate the performance of the Depth-CBF in simulation in unstructured environments on a quadrotor.

4) We validate the efficacy of this control technique in unstructured and dynamic environments on hardware using a TurtleBot3 and a quadrotor. This technique is shown to be significantly faster than previous methods.

D. Paper Organization

In Section II, we provide the background on barrier functions and point clouds. In Section III, we formulate barrier functions over point clouds. In Section IV, we discuss point cloud CBF control design for the TurtleBot and quadrotor. In Section V, we present and discuss simulation and hardware results, and in Section VI we provide concluding remarks.

II. Background

A. Control Barrier Functions

Consider a locally Lipschitz control affine system for which we wish to define safety,

$$\dot{x} = f(x) + g(x)u, \quad x \in \mathbb{R}^n, u \in \mathbb{R}^m. \tag{1}$$

A safe set $\mathcal{C}$ for (1) is defined to be the zero superlevel set of a continuously differentiable function $h : D \subseteq \mathbb{R}^n \to \mathbb{R}$,

$$\mathcal{C} = \{x \in D \mid h(x) \geq 0\} \subseteq D \subseteq \mathbb{R}^n. \tag{2}$$

To define a control barrier function for $\mathcal{C}$, we first consider the set of extended class $\mathcal{K}_\infty$ functions, denoted $\mathcal{K}_\infty^e$. A function $\alpha : \mathbb{R} \to \mathbb{R}$ belongs to class $\mathcal{K}_\infty^e$ if it is strictly increasing, continuous, and satisfies $\alpha(0) = 0$.

h is said to be a control barrier function (CBF) of system (1) if the following conditions are satisfied:

1) $\frac{\partial h}{\partial x} \neq 0$ on the boundary $\partial\mathcal{C}$ of the safe set.
2) There exists a function $\alpha \in \mathcal{K}_\infty^e$ such that for all $x \in D$
$$\sup_{u \in \mathcal{U}} [L_f h(x) + L_g h(x) u] \geq -\alpha(h(x)), \tag{3}$$
where $\mathcal{U} \subseteq \mathbb{R}^m$ is the set of admissible inputs and $L_f h = \frac{\partial h}{\partial x} f(x), L_g h = \frac{\partial h}{\partial x} g(x)$ are Lie derivatives.

If h is a CBF, then for all t there exists an input $u(t) \in \mathcal{U}$ that ensures $x(t)$ remains within the safe set $\mathcal{C}$ [1].

B. CBF-QP Controller

A popular technique for deploying a CBF for safe real-time control is the CBF-QP controller [1], which frames the search for a safe input as a quadratic program.

Suppose $k(x)$ is a nominal controller for system (1), for instance a trajectory tracking controller. If $h(x)$ is a CBF for the system, the CBF-QP "filters" the nominal input $k(x)$ into a safe input by solving the quadratic program

$$u^* = \arg\min_{u \in \mathcal{U}} \frac{1}{2} ||u - k(x)||_2^2 \tag{4}$$

$$\text{s.t. } L_f h(x) + L_g h(x) u \geq -\alpha(h(x)). \tag{5}$$

This optimizes for the closest safe input u^* to the nominal input $k(x)$ within the set $\mathcal{U}$ of admissible inputs.

C. Point Clouds

Point clouds convey the 3D spatial geometry of an environment. In this work, we consider point clouds generated by digital stereo depth cameras and lidar sensors. We define such point clouds to be finite sets of points

$$P = \{(x_i, y_i, z_i)\}_{i=1}^N, \tag{6}$$

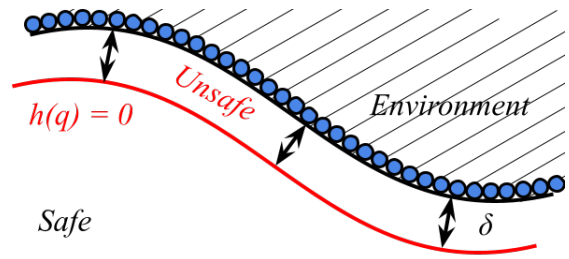

Fig. 2. A point cloud, pictured above in blue, samples the surface of the environment. The barrier function (8) defines a safety buffer of size δ with respect to the point cloud, providing a safe distance from the environment.

where each $(x_i, y_i, z_i) \in P$ represents the 3D coordinates of a point on the surface of the environment and $N \in \mathbb{N}$ represents the number of points sampled from the environment.

III. Depth-Control Barrier Function

In this section, we detail the design of CBFs over point clouds.

A. Depth-CBF Formulation

We wish to define the safe set $\mathcal{C}$ such that the system remains a safe distance away from collision with the environment. We assume that the system's only knowledge of the environment comes from a point cloud $P = \{(x_i, y_i, z_i)\}_{i=1}^N$ in the spatial frame. Assuming the system's spatial position $q \in \mathbb{R}^3$ is contained in its state vector, we define safety with the barrier function

$$h(q) = \min_{p \in P} \{(q - p)^T (q - p) - \delta^2\}, \tag{7}$$

where $\delta > 0$ is a buffer that defines the minimum safe distance to the environment. As $h(q)$ considers the minimum distance from the system to the point cloud, $h(q) > 0$ implies the system is not in collision with any point in the point cloud.

This barrier function may be conveniently rewritten as

$$h(q) = (q - p^*)^T (q - p^*) - \delta^2, \tag{8}$$

where $p^* \in P$ is the closest point in the point cloud to the system at position q. This function is illustrated in Figure 2.

B. Depth-CBF Gradient Computation

The challenge in using this barrier function comes in the form of computing the spatial gradient $\frac{\partial h}{\partial q}$, which is required for the computation of Lie derivatives for CBF-QP control.

Since the closest point p^* in the point cloud varies with q, computing the gradient of h with respect to q is not as simple as treating p^* as a constant and differentiating $h(q)$.

To model how $h(q)$ varies with respect to q, we perform a local smooth fit of $h(q)$ across a mesh of points near the current position q of the system. We define a mesh of points M with spacing $\epsilon \ll 1$ surrounding the current position q as

$$M = \{q + \epsilon(ie_1 + je_2 + ke_3) \mid -L \leq i, j, k \leq L\}, \tag{9}$$

where $e_1, e_2, e_3 \in \mathbb{R}^3$ are the standard Euclidean basis vectors. The CBF (8) is then computed across the mesh as

$$h(M) = \{h(q_m) \mid q_m \in M\}. \tag{10}$$

Critically, each $h(q_m)$ is computed with respect to the closest point p_m^* to q_m in the point cloud. This provides a mesh of CBF values that consider the dependence of p^* on q.

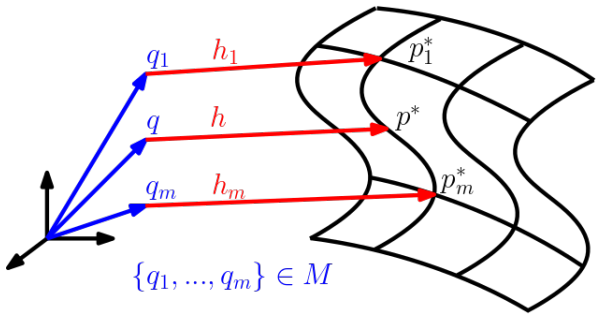

Fig. 3. By sampling the closest point in P to the system in a region surrounding the position q, we gain a local model of the barrier function.

Algorithm 1 Depth-CBF Pipeline

$M, M_0 \leftarrow \{\epsilon(ie_1 + je_2 + ke_3),\ -L \leq i, j, k \leq L\}$
$P_{kd} \leftarrow \text{kdtree}(0_{3\times K})$ ▷ Initialize k-d tree with zeros
while robot is running **do**
  if new point cloud P captured **then**
    $P_{kd} \leftarrow \text{kdtree}(\text{KNN}(P, q))$ ▷ Update k-d tree
  end if
  $M \leftarrow q + M_0$ ▷ Update mesh
  $h(M) \leftarrow \{h(q_m) \mid q_m \in M\}$ ▷ Update h values
  $\hat{A}, \hat{b}, \hat{c} \leftarrow \arg\min \sum ||h(q_m) - (q_m^T A q_m + b^T q_m + c)||_2^2$
end while

Now, we wish to estimate the spatial gradient $\frac{\partial h}{\partial q}$ of the true barrier function from the mesh of sampled values $h(M)$. This is achieved with a local fit of $h(q)$. Theorem 1 suggests a form for this local fit.

Theorem 1. [7], [8] *Consider a smooth surface embedded in 3D space. With respect to the spatial frame S, the second order Taylor approximation of the squared distance function from a point $q \in \mathbb{R}^3$ to the surface is of the form*

$$d(q) = q^T A q + b^T q + c,\ A \in \mathbb{R}^{3\times 3},\ b \in \mathbb{R}^3,\ c \in \mathbb{R}. \quad (11)$$

The proposed CBF (8) is a squared distance function to the surface of the environment, offset by a scalar δ^2. According to Theorem 1, a second order estimate of the CBF (8) in the spatial frame S is therefore of the form

$$\hat{h}(q) = q^T \hat{A} q + \hat{b}^T q + \hat{c}. \quad (12)$$

The parameters $\hat{A}, \hat{b}, \hat{c}$ of this second order estimate are found by solving the following optimization across the mesh of points M surrounding the position q of the system

$$\arg\min_{A,b,c} \sum_{q_m \in M} ||h(q_m) - (q_m^T A q_m + b^T q_m + c)||_2^2. \quad (13)$$

This may be recast as an ordinary least squares problem in A, b, c and solved efficiently in closed form.

Importantly, the estimate $\hat{h}$ (12) is everywhere differentiable, enabling estimation of the gradient of (8) as

$$\frac{\partial h}{\partial q} \approx \frac{\partial \hat{h}}{\partial q} = q^T(\hat{A} + \hat{A}^T) + \hat{b}^T. \quad (14)$$

For a control-affine system $\dot{q} = f(q) + g(q)u$, where q represents the spatial position of the system, the Lie derivatives $L_f h, L_g h$ are thus locally estimated using the CBF fit as

$$L_f h \approx L_f \hat{h} = (q^T(\hat{A} + \hat{A}^T) + \hat{b}^T) f(q) \quad (15)$$

$$L_g h \approx L_g \hat{h} = (q^T(\hat{A} + \hat{A}^T) + \hat{b}^T) g(q). \quad (16)$$

Estimates of the second order Lie derivatives $L_f^2 h,\ L_g L_f h$, required for relative degree 2 systems, are computed

$$L_f^2 \hat{h} = L_f[L_f \hat{h}],\ L_g L_f \hat{h} = L_g[L_f \hat{h}]. \quad (17)$$

C. Depth-CBF Pipeline

The Depth-CBF pipeline is outlined in Algorithm 1. Notice that each mesh M is created by adding the current position q of the system to a default mesh M_0 centered at 0.

The implementation of this Depth-CBF pipeline depends largely on efficient computation of the closest point in the point cloud to the system, as this point must be repeatedly queried to generate the mesh of barrier function values.

Querying efficiency is achieved by inserting the K nearest points to the current position into a k-d tree, a binary tree that enables querying in expected logarithmic time [9].

A new k-d tree is generated for each new point cloud to enable adaptation to changing environments. This must be traded off with a lack of "history" of previous point clouds.

In simulation, the k-d tree improved the speed of Algorithm 1 by a mean of 22.62 Hz compared to a naive search. This includes the extra time necessary to generate the tree.

IV. Depth-CBF Control Design

Once the local fit of the barrier function has been performed, an input is computed using a modified CBF-QP controller. In the place of $L_f h, L_g h$, the estimates $L_f \hat{h}, L_g \hat{h}$ are used. For $k(q)$ a nominal controller for the system $\dot{q} = f(q) + g(q)u$, the Depth-CBF-QP controller is formulated as

$$u^* = \arg\min_{u \in \mathcal{U}} \frac{1}{2}||u - k(q)||_2^2 \quad (18)$$

$$s.t.\ L_f \hat{h} + L_g \hat{h} u \geq -\alpha(h(q)). \quad (19)$$

This formulation allows for safe control using only a single optimization constraint across the entire point cloud.

Since the Depth-CBF-QP simply relies on a point cloud, it can be used with both lidar sensors and stereo depth cameras. Additionally, it is easily generalized to the relative degree 2 case, which is detailed below.

A. TurtleBot Depth-CBF Design

First, we detail the design of a relative degree 1 Depth-CBF for the TurtleBot, modeled with the unicycle dynamics

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} \cos\varphi & 0 \\ \sin\varphi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}, \quad (20)$$

where x, y are the center of the TurtleBot and φ is the heading angle. This system is controlled by two inputs, a linear velocity v and an angular velocity ω [10].

To deploy the Depth-CBF on such a system, we propose a cascaded control structure. In the first stage, a desired (x, y) trajectory $q_d(t) : \mathbb{R} \to \mathbb{R}^2$ is passed into a Depth-CBF-QP controller built around the integrator dynamics $\dot{q} = u$. Here, $q = [x, y]^T \in \mathbb{R}^2$ is the spatial position of the integrator and the control input $u \in \mathbb{R}^2$ is its velocity. For this system, $L_f \hat{h} = 0$ and $L_g \hat{h} = q^T(\hat{A} + \hat{A}^T) + \hat{b}^T$. This produces the following Depth-CBF-QP optimization

$$u^* = \arg\min_{u \in \mathbb{R}^2} \frac{1}{2}||u - k(q)||_2^2 \quad (21)$$

$$s.t.\ L_g \hat{h} u \geq -\alpha h, \quad (22)$$

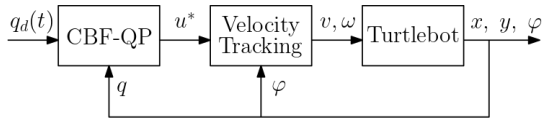

Fig. 4. The Depth-CBF-QP control structure for a TurtleBot system.

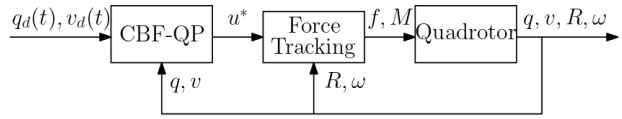

Fig. 5. The Depth-CBF-QP control structure for a quadrotor system.

TABLE I

PARAMETERS OF THE 3D QUADROTOR

Parameter	Description
$m \in \mathbb{R}$	Total mass
$g \in \mathbb{R}$	Acceleration due to gravity
$q \in \mathbb{R}^3$	Position of center-of-mass in spatial frame
$v \in \mathbb{R}^3$	Velocity of center-of-mass in spatial frame
$\omega \in \mathbb{R}^3$	Angular velocity in body frame
$F \in \mathbb{R}$	Total thrust
$M \in \mathbb{R}^3$	Total moment applied in body frame
$e_3 \in \mathbb{R}^3$	Euclidean basis vector $(0, 0, 1)$
$J \in \mathbb{R}^{3\times 3}$	Inertia tensor in the body frame
$R \in SO(3)$	Rotation matrix from body to spatial frame

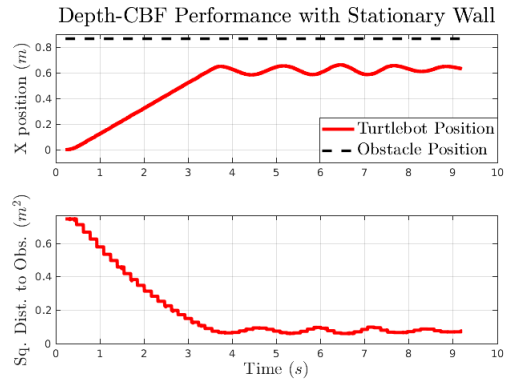

Fig. 6. TurtleBot trajectory and squared distance to obstacle in the presence of a stationary wall in hardware. The Depth-CBF controller enables the TurtleBot to stop short of collision with the wall. The authors interpret the "steps" and oscillation visible in the value of the squared distance as a byproduct of the low refresh rate of the lidar.

where $k(q)$ is a nominal tracking controller for the integrator and $\alpha \in \mathbb{R}_{>0}$. This QP produces a safe velocity vector $u^* \in \mathbb{R}^2$ which may be tracked by the TurtleBot using a velocity tracking controller. This structure is visualized in Figure 4.

Note that the structure of tracking the Depth-CBF-QP input to an integrator is easily adapted to any system for which a velocity tracking controller may be designed.

B. Quadrotor Depth-CBF Design

Here, we discuss the design of a Depth-CBF-QP controller for the quadrotor. With respect to the parameters defined in Table I, the dynamics of the quadrotor are described as [11]

$$\dot{q} = v,\ m\dot{v} = fRe_3 - mge_3, \tag{23}$$

$$\dot{R} = R\hat{\omega},\ J\dot{\omega} + \hat{\omega}J\omega = M, \tag{24}$$

for $\hat{\cdot} : \mathbb{R}^3 \to so(3)$ defined such that $\hat{x}y = x \times y\ \forall x, y \in \mathbb{R}^3$.

A controller is designed by separating the translational and rotational dynamics of the quadrotor [12]. First, a nominal trajectory tracking controller is designed around point mass dynamics, which are represented in control affine form as

$$\begin{bmatrix} \dot{q} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v \\ -ge_3 \end{bmatrix} + \begin{bmatrix} 0_{3\times 3} \\ \frac{1}{m} I_{3\times 3} \end{bmatrix} u, \tag{25}$$

for $u \in \mathbb{R}^3$ a force vector input and $x = (q, v)$ the state vector. For a desired trajectory $x_d(t) : \mathbb{R} \to \mathbb{R}^6$, where $x_d(t) = (q_d(t), v_d(t))$, a tracking input $k(x)$ is chosen

$$k(x) = -K(x - x_d) + m\ddot{q}_d + mge_3, \tag{26}$$

where $K \in \mathbb{R}^{3\times 6}$ is a gain matrix and $\ddot{q}_d \in \mathbb{R}^3$ is the desired acceleration. Now, we design a CBF-QP controller.

As this system is relative degree 2 with respect to the CBF (8) a second order formulation of the Depth-CBF-QP constraint must be used. The input is then evaluated as

$$u^* = \arg\min_{u \in \mathbb{R}^3} \frac{1}{2}||u - k(x)||_2^2 \tag{27}$$

$$s.t.\ L_f^2\hat{h} + L_g L_f \hat{h} u + \alpha_1 L_f \hat{h} + \alpha_0 h \geq 0, \tag{28}$$

where α_0, α_1 are selected such that the characteristic polynomial of the constraint has negative real roots [1], [13]. This constraint provides safety in the relative degree 2 case.

The solution to (27) is passed to a force-tracking controller, which produces a thrust $f \in \mathbb{R}$ and a moment $M \in \mathbb{R}^3$ that allows the quadrotor to track the safe force u^* [12]. This control structure is visualized in Figure 5.

V. RESULTS & DISCUSSION

In this section, we illustrate the performance of the relative degree 1 and 2 Depth-CBF-QP controllers. First, we present the performance of the relative degree 1 Depth-CBF-QP on TurtleBot hardware. Following this, we discuss the performance of the relative degree 2 Depth-CBF-QP on the quadrotor with simulation and preliminary hardware results.

The QP optimizations detailed above were solved using OSQP [14] for both simulation and hardware experiments. Code for the results presented in this work can be found at `https://github.com/HybridRobotics/depth_cbf`.

A. Relative Degree 1 Results

Here, we discuss the performance of the relative degree 1 Depth-CBF on TurtleBot3 hardware. This platform is equipped with a lidar sensor that scans the environment at 5 Hz. Computations were performed on an i7-4770 and relayed to the TurtleBot via ROS Noetic. A buffer of $\delta = 0.275\ m$ was selected to account for noise and for the TurtleBot's radius. A CBF-QP constraint constant of $\alpha = 6$ was chosen.

1) Stationary Wall: In the first test, the TurtleBot is commanded to drive towards a stationary, flat wall. As shown in Figure 6, the Depth-CBF prevents collision with the wall.

Figure 6 also illustrates the effect of the buffer δ, as the TurtleBot always remains a distance away from the wall. This helps ensure safety under sensor noise and is valuable for dealing with low frequency sensors such as the TurtleBot lidar.

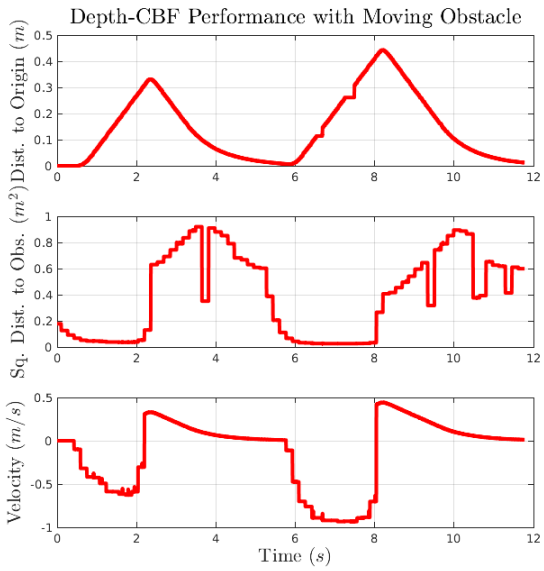

Fig. 7. Depth-CBF performance on the TurtleBot hardware in the presence of a moving obstacle. An obstacle is brought towards the system and removed, and the TurtleBot is able to successfully avoid the moving obstacle.

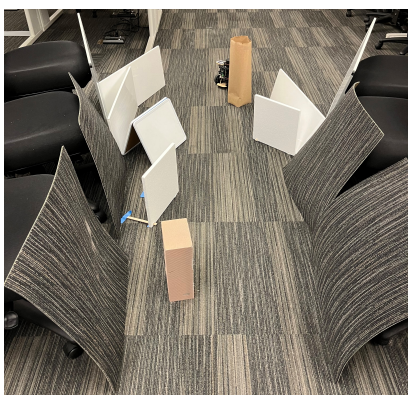

Fig. 8. The TurtleBot must move from one end of the cluttered hallway to the other without colliding with any of the obstacles.

2) Moving Wall: Next, we discuss the performance of the Depth-CBF under a moving obstacle. The system is commanded to stay at $(0, 0, 0)$ while a flat obstacle is periodically brought towards it and removed from view.

As shown in Figure 7, the Depth-CBF smoothly drives the TurtleBot away from the moving obstacle. When the obstacle is removed, the TurtleBot returns to the origin. This process is then repeated a second time. We thus observe the ability of the Depth-CBF to adapt to a dynamic environment.

3) Cluttered Hallway: In this test, a narrow hallway with numerous obstacles, depicted in Figure 8, was constructed. The TurtleBot was placed at the origin and given a nominal straight-line trajectory from $(0, 0, 0)$ to $(3, 0, 0)$ at the end of the hall. This path passes through multiple obstacles.

We visualize the trajectory of the TurtleBot through the cluttered hallway in Figure 9. Despite having no planner other than a single open-loop trajectory, the Depth-CBF enabled the TurtleBot to steer around every obstacle and navigate its way to the end of the hall. The TurtleBot also stopped short of an obstacle obstructing the goal state.

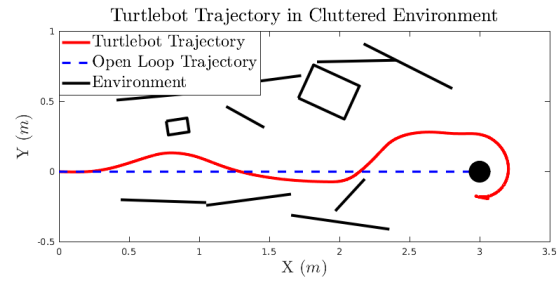

Fig. 9. The trajectory of the TurtleBot through the hallway in hardware. The black lines and boxes represent the obstacles depicted in Figure 8. Using the Depth-CBF, the TurtleBot successfully avoids the obstacles and navigates to the end of the hall with no information other than a straight open-loop trajectory.

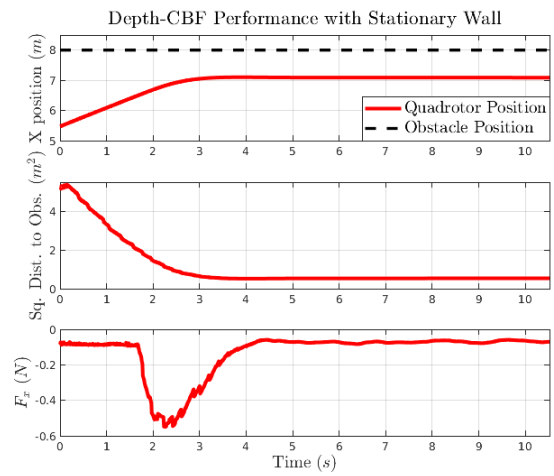

Fig. 10. Simulation results for the quadrotor: in plot 1, we see the x position of the quadrotor with respect to the wall, in plot 2 the squared distance from the quadrotor to the wall, and in plot 3 the x component of the force vector from the Depth-CBF-QP optimization. We observe that the Depth-CBF enables the quadrotor to smoothly avoid collision with the wall.

B. Relative Degree 2 Results

Now, we discuss the performance of the relative degree 2 Depth-CBF on the quadrotor. Simulations were performed in Gazebo with the Intel Realsense-ROS plugin to simulate a 30 Hz depth camera. Hardware experiments were performed on a quadrotor equipped with an Intel NUC and an Intel D435i depth camera. A buffer of $\delta = 0.75\ m$ was chosen to account for the quadrotor's radius and for sensor noise. CBF-QP constraint constants of $\alpha_0 = \alpha_1 = 4$ were chosen.

1) Stationary Wall: In the first simulation test, the quadrotor is commanded to fly towards a flat, stationary wall, and the Depth-CBF must ensure that no collisions occur. The results of this test are visualized in Figure 10. We observe that the Depth-CBF ensures safety throughout the flight.

2) Moving Wall: A preliminary moving obstacle test was performed in hardware on the quadrotor. As observed in Figure 11, the quadrotor successfully evades an obstacle moving towards it. When the obstacle is removed, the quadrotor returns to its original position.

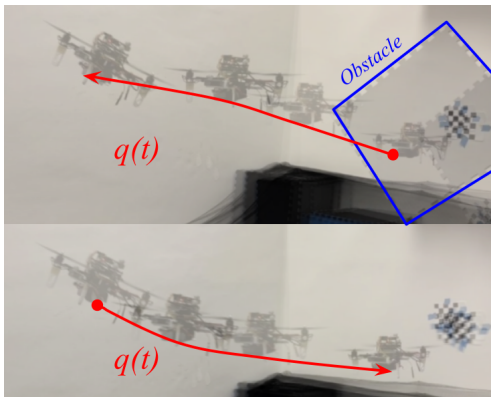

Fig. 11. Quadrotor hardware test: using the Depth-CBF, the quadrotor is able to successfully avoid a moving obstacle which is brought towards it at a constant velocity. When the obstacle is removed, the quadrotor returns to its initial position.

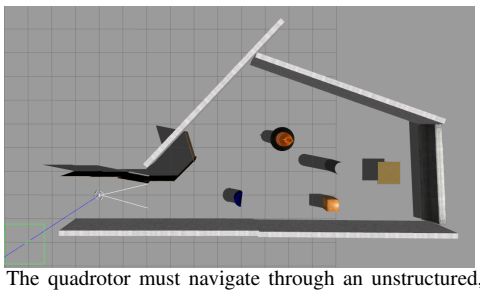

Fig. 12. The quadrotor must navigate through an unstructured, cluttered environment in simulation with no planning while maintaining safety.

3) Unstructured Environment: Now, we consider the performance of the quadrotor in a simulated unstructured environment, depicted in Figure 12. The quadrotor is commanded to follow a straight-line trajectory from $(0, 0, 0) \in \mathbb{R}^3$ to $(11, 0, 1) \in \mathbb{R}^3$, which would take it into direct contact with several obstacles. Using the Depth-CBF, the quadrotor must navigate around the obstacles and reach the goal state. This test is then repeated with a new goal state of $(11, 2, 1) \in \mathbb{R}^3$.

As observed in Figure 13, the quadrotor successfully navigates around the obstacles and reaches the goal state for both trajectories. This safe navigation is provided purely by the Depth-CBF-QP controller with no outside planning.

C. Discussion

Overall, the Depth-CBF controller successfully ensured the system did not come into collision with the environment in a variety of simulation and hardware tests.

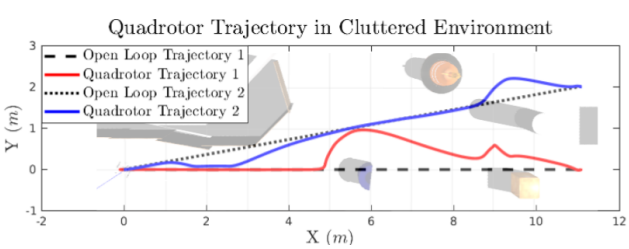

Fig. 13. The trajectories of the quadrotor through the simulated cluttered environment. The quadrotor is successfully able to navigate around the environment without collision for both of the open loop trajectories.

A major attractive point of the Depth-CBF control structure is its computational efficiency. Across the TurtleBot tests, the Depth-CBF ran at an average frequency of 485 Hz. This frequency is measured across the entire pipeline, from receiving a new point cloud to computing the CBF approximation and CBF-QP control inputs. This is a significant increase in speed compared to prior approaches.

A drawback of this approach is the lack of ”history”, that is Algorithm 1 does not consider prior states or point-clouds. Further, using a single closest point could prove challenging in extremely dense environments. One approach to mitigate that is to consider a set of closest points (that are relatively far enough from each other), however, this increases the computational cost. The tradeoff to the lack of history is the ability to react to moving obstacles and the computation speed. Quick reactions to obstacles could be helpful during teleoperations when the operators might not be able to react fast enough to dynamic environments.

VI. Conclusions and Future Work

In this paper, we have presented the Depth-CBF-QP, a lightweight method for synthesizing CBF controllers over point clouds. The Depth-CBF-QP only requires a single optimization constraint and enables fast, accurate estimation of the CBF Lie derivatives. Further, the Depth-CBF-QP can be formulated for both the relative degree 1 and 2 cases, and can be applied to both stereo depth and lidar sensors.

We validated performance in simulation and hardware for both the relative degree 1 and 2 cases. We performed static braking, moving obstacle, and cluttered hallway navigation tests on the TurtleBot and quadrotor. In all three tests, the systems never came into contact with the environment. The cluttered hallway test demonstrated the ability of the Depth-CBF to enable safe navigation through unstructured environments with no outside planning.

The Depth-CBF proved to be very fast, running at an average rate of 485 Hz in TurtleBot experiments. This represents a significant performance increase over previous methods.

Future extensions of this work include performing extensive validations of the Depth-CBF on other hardware platforms and in performing an analysis of the effect of noise and time delay in receiving point clouds on performance.

Acknowledgments

The authors would like to thank UC Berkeley courses EECS C106A/B for providing access to their TurtleBots, and HiPeRLab, UC Berkeley for the use of their flightspace. This work was partially supported through funding from the Tsinghua-Berkeley Shenzhen Institute (TBSI) program.

References

[1] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, “Control barrier functions: Theory and applications,” in *European control conference*, 2019, pp. 3420–3431.

[2] A. Elfes, “Using occupancy grids for mobile robot perception and navigation,” *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[3] R. K. Cosner, I. D. J. Rodriguez, T. G. Molnar, W. Ubellacker, Y. Yue, A. D. Ames, and K. L. Bouman, "Self-supervised online learning for safety-critical control using stereo vision," in *International Conference on Robotics and Automation (ICRA)*, 2022, pp. 11 487–11 493.

[4] 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.

[5] M. Tong, C. Dawson, and C. Fan, "Enforcing safety for vision-based controllers via Control Barrier Functions and Neural Radiance Fields," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 10 511–10 517.

[6] 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.

[7] G. Folland, *Advanced Calculus*. Prentice-Hall, 2002.

[8] H. Pottmann and M. Hofer, *Geometry of the squared distance function to curves and surfaces*. Springer, 2003.

[9] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An Algorithm for Finding Best Matches in Logarithmic Expected Time," *ACM Trans. Math. Softw.*, vol. 3, no. 3, p. 209–226, 1977.

[10] S. Sastry, *Nonlinear systems: analysis, stability, and control*. Springer Science & Business Media, 2013, vol. 10.

[11] G. Wu and K. Sreenath, "Safety-critical control of a 3d quadrotor with range-limited sensing," in *Dynamic Systems and Control Conference*, vol. 50695. American Society of Mechanical Engineers, 2016, p. V001T05A006.

[12] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor UAV on SE(3)," in *IEEE conference on decision and control (CDC)*, 2010, pp. 5420–5425.

[13] Q. Nguyen and K. Sreenath, "Exponential control barrier functions for enforcing high relative-degree safety-critical constraints," in *American Control Conference (ACC)*, 2016, pp. 322–328.

[14] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

[15] H. Abdi, G. Raja, and R. Ghabcheloo, "Safe Control using Vision-based Control Barrier Function (V-CBF)," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 782–788.

[16] J. Zeng, B. Zhang, and K. Sreenath, "Safety-critical model predictive control with discrete-time control barrier function," in *American Control Conference (ACC)*, 2021, pp. 3882–3889.

[17] Y. Ma, S. Soatto, J. Košecká, and S. Sastry, *An invitation to 3-d vision: from images to geometric models*. Springer, 2004, vol. 26.

[18] 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.

[19] X. Xu, T. Waters, D. Pickem, P. Glotfelter, M. Egerstedt, P. Tabuada, J. W. Grizzle, and A. D. Ames, "Realizing simultaneous lane keeping and adaptive speed regulation on accessible mobile robot testbeds," in *IEEE conference on control technology and applications (CCTA)*, 2017, pp. 1769–1775.

[20] Q. Liao, Z. Li, A. Thirugnanam, J. Zeng, and K. Sreenath, "Walking in Narrow Spaces: Safety-critical Locomotion Control for Quadrupedal Robots with Duality-based Optimization," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, Detroit, MI, October 2023.