2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Safe Bipedal Path Planning via Control Barrier Functions for Polynomial Shape Obstacles Estimated Using Logistic Regression

Chengyang Peng[1], Octavian Donca[1], Guillermo Castillo[1], and Ayonga Hereid[1]

***Abstract*— Safe path planning is critical for bipedal robots to operate in safety-critical environments. Common path planning algorithms, such as RRT or RRT*, typically use geometric or kinematic collision check algorithms to ensure collision-free paths toward the target position. However, such approaches may generate non-smooth paths that do not comply with the dynamics constraints of walking robots. It has been shown that the control barrier function (CBF) can be integrated with RRT/RRT* to synthesize dynamically feasible collision-free paths. Yet, existing work has been limited to simple circular or elliptical shape obstacles due to the challenging nature of constructing appropriate barrier functions to represent irregularly shaped obstacles. In this paper, we present a CBF-based RRT* algorithm for bipedal robots to generate a collision-free path through space with multiple polynomial-shaped obstacles. In particular, we used logistic regression to construct polynomial barrier functions from a grid map of the environment to represent irregularly shaped obstacles. Moreover, we developed a multi-step CBF steering controller to ensure the efficiency of free space exploration. The proposed approach was first validated in simulation for a differential drive model, and then experimentally evaluated with a 3D humanoid robot, Digit, in a lab setting with randomly placed obstacles.**

I. Introduction

Mobile robots have shown encouraging promises in many real-world applications outside traditional well-structured factory settings thanks to the recent advancement of real-time path planning [1]. Path planning in 2D space has been extensively studied over the past decades [2], [3]. A feasible path for a robot requires starting from an initial position to the goal position without colliding with any obstacle in the environment. Arguably the most prevailing approach in path planning is the sampling-based Rapidly Exploring Random Trees (RRT) algorithm, which expends the path by randomly sampling points in the configuration space [4]. To improve the optimality of the resulting path, Karaman and Frazzoli [5] proposed RRT*, which can reconnect the newly added node to the nearby nodes based on the minimum cost from the root node to the new node. Much progress has been made recently in combining low-level control synthesis and path planning, such as LQR-RRT* and anytime multi-directional RRT* [6]–[9], to ensure that the generated paths are consistent with the underlying dynamics constraints of the robot.

With the trending occasions of robots operating in the safety-critical environment (e.g., around people or in crowded spaces), the safety of robot motion becomes increasingly critical for the continuous deployment of these intelligent machines. Control Barrier Function (CBF) is a popular tool in guaranteeing safety for nonlinear systems and constraints [10], which has been shown effective in enforcing the safety-critical constraints on nonlinear systems such as autonomous vehicles and bipedal robot locomotion [11]–[15]. Recently, this method has also been used for designing safety-critical path planners. Yang et al. introduced a Quadratic Program (QP) that enforces CBF constraints to achieve obstacle avoidance [16]. Aniketh et al. proposed a framework to incorporate CBF constraints into RRT path planning [17]. On these foundations, Ahmad et al. combined RRT* algorithm with the CBF and equipped it with an adaptive sampling method to improve efficiency [18]. Liu et al. present a real-time safe planning system for bipedal robots based on one CBF [19]. The system detects non-overlapping obstacles in the environment through LiDAR point cloud data and computes the elliptical CBF representation. However, the obstacles studied by the above methods and algorithms only focused on circular and elliptical shapes because it would be easy to obtain their barrier functions. In many real-world scenarios, the circular barrier function is insufficient or wasteful to represent complex-shaped obstacle regions.

Fig. 1. The snapshots of the bipedal robot, Digit, following the collision-free path generated by the proposed algorithm.

In this work, we developed a modified CBF-RRT* algorithm in Python with a new CBF-QP based multi-step steering controller for safe path planning in 2D complex environments. The contributions of the proposed work are as follows. First, we proposed a new method that uses logistic regression to construct barrier functions that use polygon shapes to represent complex obstacles. Second, instead of calculating CBF-QP once when sampling and moving one

*This work was supported in part by the National Science Foundation under grant FRR-21441568.

[1]Mechanical and Aerospace Engineering, Ohio State University, Columbus, OH, USA. (peng.947, donca.2, castillomartinez.2, hereid.1)@osu.edu.

iteration step, we would divide one step into four small steps, which can effectively trade off the planning speed and path safety. Finally, we applied our modified CBF-RRT* algorithm to a bipedal robot to enable it to navigate safely in a room with complex obstacles and unreachable regions. We evaluated the proposed algorithm on a Digit robot in the lab setting and demonstrated the safe navigation of bipedal walking robots.

The rest of the paper is organized as follows. Section II reviews the background of CBF and its integration with RRT/RRT* based planning algorithms. In Section III, we present the core contribution of the paper, a CBF-RRT* planning algorithm with multi-step steering and polynomial-shaped barrier representation of obstacles. The simulation and experimental results with the Digit robot are presented in Section IV. Finally, Section V briefly discusses the limitation of the proposed work and future research directions.

II. Background

In this section, we briefly review the mathematical basis of the control barrier function (CBF) and how it has been integrated with the RRT/RRT* based planning algorithms for safe navigation.

A. Control Barrier Function (CBF)

We consider the robot dynamics can be written as the following affine nonlinear system:

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

where $x \in \mathcal{X}$ is the system state with $\mathcal{X} \subseteq \mathbb{R}^n$ being the state space, and $u \in \mathcal{U}$ is the control input with $\mathcal{U} \subseteq \mathbb{R}^m$ being the control space. If there exists a continuous and differentiable function $h : \mathbb{R}^n \to \mathbb{R}$, the safety set $\mathcal{C}$ of the system may be defined as [20]:

$$\begin{aligned} \mathcal{C} &= \{x \in \mathbb{R}^n | h(x) \geq 0\}, \\ \partial\mathcal{C} &= \{x \in \mathbb{R}^n | h(x) = 0\}. \end{aligned} \tag{2}$$

If $h(x)$ has relative degree $m > 1$, we can define a serious function $\psi_m(x) : \mathbb{R}^n \to \mathbb{R}$ given as [21]:

$$\begin{aligned} &\psi_0(x) := h(x), \\ &\psi_i(x) := \dot{\psi}_{i-1}(x) + \alpha_i(\psi_{i-1}(x)) \qquad i \in \{1, ..., m\}, \end{aligned} \tag{3}$$

where $\alpha_i(\cdot)$ is a class κ function. The forward invariance safety condition can then be guaranteed if the following inequality constraints are satisfied for all $x \in \mathcal{C}$:

$$\begin{aligned} L_f^m h(x) + L_g L_f^{m-1} h(x) u + \frac{\partial^m h(x)}{\partial t^m} + O(h(x)) \\ + \alpha_m(\psi_{m-1}(x)) \geq 0, \end{aligned} \tag{4}$$

where $O(h(x))$ denotes the remaining Lie derivatives along f and partial derivatives with respect to t with degree less than or equal to $m-1$. Therefore, if $h(x)$ satisfied both (2) and (4), it can be called a control barrier function. Since the control input u is affine in (1) and (4), one can formulate a quadratic programming (QP) controller subject to the CBF constraint in (4) to synthesize safe control actions [13], [20], [21].

B. CBF-RRT/RRT*

Built upon the standard RRT algorithm, Yang et al. developed the CBF-RRT path planning algorithm that uses CBF-QP [13] based safety-critical controllers to generate intermediate control actions to steer the robot away from the obstacles when approaching them [16]. The CBF controller replaces the collision check function in the traditional RRT algorithm while still ensuring safety. In [17], Aniketh et al. improved the computational efficiency of CBF-RRT further by replacing CBF-QP with a random sampling of control actions that satisfy the barrier condition described in (4). While it preserves the nature of random exploration by RRT, these approaches are often unable to generate (probabilistically) optimal paths. To improve the optimality of the resulting path, Ahmad et al. combined CBF-QP with RRT* based on the work in [16], and improved the sampling efficiency through adaptive sampling based on the cross-entropy method (CEM) [18]. It has been shown that RRT* yields a relatively optimal solution if given sufficient computation time. This is realized through two critical procedures described below [22]:

- `ChooseParent`: finds the near neighbor nodes around the new node. If there is no obstacle collision between the new node and each near node, the algorithm will compute the cost of the new node through each near node. Finally, it chooses the neighbor node that makes the cost minimum, as the parent node of the new node.
- `Rewrite`: reconnects each near neighbor node with the new node and checks their collisions. It calculates the costs of these near nodes through the new node. Finally, it selects the optimal cost and rewrites the tree.

In [18], the authors replaced the collision check function in the above two procedures with a CBF-QP based steering function, which inevitably increased the computational overhead of the CBF-RRT*. It is also important to note that the three aforementioned studies expand the tree by randomly sampling a node on the tree to extend toward the target position. This practice is inefficient in expanding the tree outward into feasible areas, thereby increasing the total number of iterations, as well as the computation overhead, required for the algorithm. Moreover, determining a proper set of barrier functions to describe obstacles and unreachable areas remains challenging when using CBF for sampling-based path planning. The existing work only considers simple shapes, such as circles or ellipses.

III. Safe Navigation via Multi-Step CBF-QP Steering with RRT*

In this section, we present a safe path planning algorithm for bipedal robots that integrates the control barrier function with RRT* to provide guaranteed obstacle avoidance without explicit collision checking. Moreover, we propose to construct polynomial barrier functions to represent complex obstacles or unreachable regions using logistic regression on the planar grid map of the environment. Finally, we develop a multi-step CBF steering algorithm to address the infeasibility issues that the state may end up in the unsafe set.

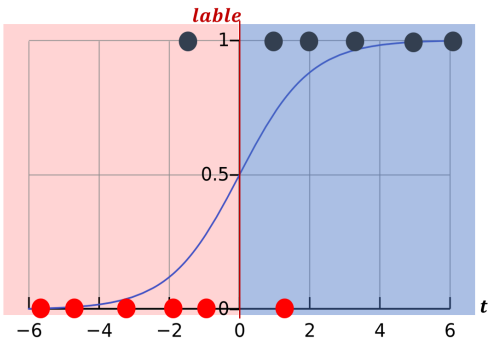


Fig. 2. Two sets of data. Label 0 data is in red, and Label 1 is in gray. The sigmoid function is helpful in classifying two sets of data into two regions. In the blue region($t > 0$), the label of data is more likely to be 1. In the red region($t < 0$), the label of data is more likely to be 0. And $t = 0$ becomes a boundary that separates these two regions.

A. Bipedal Path Planning with Simplified Model

With the purpose of finding an obstacle-free path, we consider the bipedal robot as a simple mass, assuming that there exists a stable low-level locomotion control that can follow waypoint or velocity commands. While a biped robot can walk in all directions, we only consider forward walking and turning in this paper. This is due to the difficulty of accurately controlling lateral walking speeds as the robot swings left and right while walking sideways. To mitigate these complexities for initial studies, we regard the bipedal robot as a differential drive type model, with states and derivative of states given by:

$$\mathbf{x} = [p_x, p_y, \theta]^T, \tag{5}$$

$$\dot{\mathbf{x}} = [v\cos\theta, v\sin\theta, \omega]^T, \tag{6}$$

where (p_x, p_y, θ) corresponds to robot's position and heading direction in the world coordinate, and (v, ω) represents the robot's forward and angular velocity. If we assume v is a constant, the system of the robot can be simplified as

$$\dot{\mathbf{x}} = \begin{bmatrix}\dot{p}_x \\ \dot{p}_y \\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}v\cos\theta \\ v\sin\theta \\ 0\end{bmatrix} + \begin{bmatrix}0 \\ 0 \\ 1\end{bmatrix}\omega = f(\mathbf{x}) + g(\mathbf{x})u, \tag{7}$$

with the control input being $u = \omega$. To avoid a collision with the obstacle, one needs to synthesize angular velocity commands that safely steer the robot away from the obstacle.

B. Construct Polynomial Barrier Functions via Logistic Regression from 2D Obstacle Map

To avoid collisions using CBF, we need to define a set of barrier functions $h(\mathbf{x})$ to represent the boundary of obstacles in the environment. In this paper, we propose to use logistic regression to naturally construct polynomial-shaped barrier functions from a grid map. Compared to circular or ellipse shapes, polynomial shapes can efficiently represent arbitrarily-shaped obstacles. Since the free space (i.e., safe set) is defined as the outside of the closed shapes, polygons can be used to represent obstacles when using CBF.

Logistic regression is a standard probabilistic statistical classification model, which classifies data with different labels [23], [24]. The outcome of logistic regression on one data sample is the probability of belonging to label 1 or label 0. The classification model (sigmoid function) is given by:

$$\mathbb{P}(y = 1|t) = \frac{1}{1 + e^{-t}}, \tag{8}$$

where $\mathbb{P}(y = 1|t)$ represents the probability of the label of the feature t is 1. Given (8), $\mathbb{P}(y|t) \in [0.5, 1)$ if $t > 0$, meaning the label is more likely be 1; and $\mathbb{P}(y|t) \in [0, 0.5)$ if $t < 0$, meaning that the label is more likely be 0. Assuming we have a set of data whose label could be either 1 or 0, as shown in Fig. 2, we can use $t = 0$ as a classifier or decision boundary to classify this set of data into two clusters. In particular, one can express t as an affine function of a set of variables $\mathbf{z} = (z_0, z_1, z_2, ..., z_{j-1})^T \in \mathbb{R}^j$, given as:

$$t = \boldsymbol{\beta}\mathbf{z}, \tag{9}$$

where $\boldsymbol{\beta} = (\beta_0, \beta_1, \beta_2, ..., \beta_{j-1}) \in \mathbb{R}^j$ is a vector of the unknown coefficient of the function. By giving N sets of data $\mathbf{z}$ and their label $\mathbf{y}$, the parameters $\boldsymbol{\beta}$ can be determined by minimizing the binary cross entropy cost:

$$\boldsymbol{\beta} = \text{argmin} \sum_{n=1}^{N} (\ln(1 + e^{t_i}) - y_i t_i) \quad \text{for } t_i = \boldsymbol{\beta}\mathbf{z_i}. \tag{10}$$

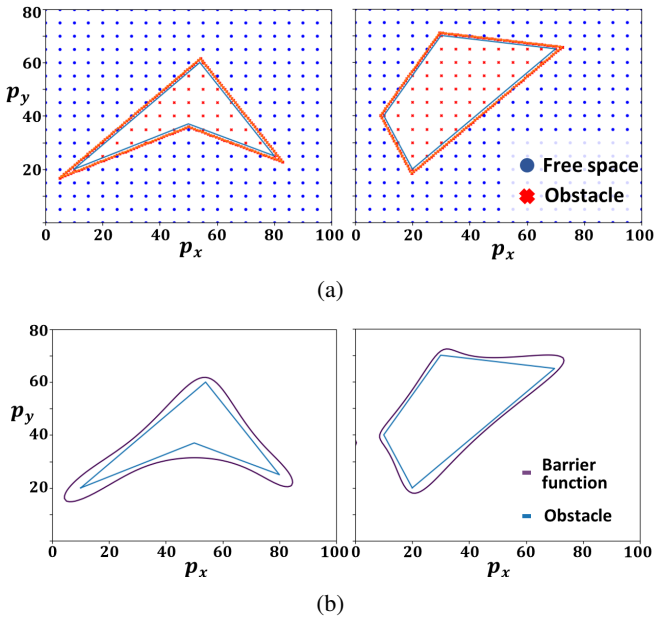
In this paper, we used Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm in Python open source package, Scipy. This is a method for solving unconstrained nonlinear optimization problems, to find $\boldsymbol{\beta}$.

The equation $t = \boldsymbol{\beta}\mathbf{z} = 0$ is the decision boundary of two sets. For our purpose, we can consider areas that have $t > 0$ as free space (i.e., safe set), and $t < 0$ as obstacle space (i.e., unsafe set). This is consistent with the CBF definition in (2). To construct barrier functions from a 2D obstacle map, the first step is to determine the set of variables $\mathbf{z}$ from the robot's state variables $\mathbf{x}$. In this paper, we empirically select a set of polynomial functions of the robot's position with the maximum power of 4, given by

$$\mathbf{z} = [1, p_x^1 p_y^0, p_x^0 p_y^1, ..., p_x^1 p_y^3, p_x^0 p_y^4]^T \tag{11}$$

where (p_x, p_y) is an x-y coordinate of a point on the map. The coefficient vector is also defined accordingly, as $\boldsymbol{\beta} = [\beta_0, \beta_1, \beta_2, ..., \beta_{14}, \beta_{15}]$. A barrier function is then given by $h(\mathbf{x}) := \boldsymbol{\beta}\mathbf{z}$, which encloses an occupied area in the map.

To solve the parameters $\boldsymbol{\beta}$, we get arrays by putting each point position into (11), and save these arrays into matrix $\mathbf{Z}$. Derive the cost function for $h(\mathbf{x})$, and using array $\mathbf{Y}$ which stores the label (occupied or free) of corresponding position and matrix $\mathbf{Z}$ to minimize the cost function in (10). Finally, using BFGS method, we solve the parameters $\boldsymbol{\beta}$ to construct barrier functions. Fig. 3 shows two examples of resulting barrier boundaries of convex and concave-shaped obstacles.



(a)

(b)

Fig. 3. Illustration examples of constructing barrier functions from 2D obstacle map. We first sampled equidistant n points on the map and labeled each point either free or obstacle. (a). The sampled points in a map. Red crosses mean Label 0: when the point is in an obstacle; and Blue dots mean label 1: which is free space. (b). The generated barrier functions closely represent concave and convex obstacle regions.

C. CBF-QP Safe Steering Controller

Given a barrier function $h(\mathbf{x}) = \boldsymbol{\beta}\mathbf{z}$ generated by solving (10), where $\mathbf{z}$ is composed of polynomial combinations of the states (p_x, p_y), we will formulate a CBF-QP steering control for the system defined in (7). Since $h(\mathbf{x})$ has relative degree two, we can construct a CBF according to (4), given as

$$L_f^2 h(\mathbf{x}) + L_g L_f h(\mathbf{x}) u + \alpha_2(\psi_1(\mathbf{x})) \geq 0, \tag{12}$$

where

$$\psi_0(\mathbf{x}) = h(\mathbf{x}), \tag{13}$$
$$\psi_1(\mathbf{x}) = \dot{\psi}_0(\mathbf{x}) + \alpha_1\left(\psi_0(\mathbf{x})\right). \tag{14}$$

Following the definition of $h(\mathbf{x})$, we have

$$\begin{aligned}\dot{\psi}_0(\mathbf{x}) = \dot{h}(\mathbf{x}) &= \boldsymbol{\beta}\left(\frac{\partial \mathbf{z}}{\partial p_x}\dot{p}_x + \frac{\partial \mathbf{z}}{\partial p_y}\dot{p}_y\right) \\ &= \boldsymbol{\beta}\left(\frac{\partial \mathbf{z}}{\partial p_x} v\cos\theta + \frac{\partial \mathbf{z}}{\partial p_y} v\sin\theta\right),\end{aligned} \tag{15}$$

$$\begin{aligned}L_f^2 h(\mathbf{x}) = \boldsymbol{\beta}&\left(\frac{\partial^2 \mathbf{z}}{\partial p_x^2} v^2\cos^2\theta + \frac{\partial}{\partial p_x}\left(\frac{\partial \mathbf{z}}{\partial p_y}\right) v^2\sin\theta\cos\theta\right. \\ &\left. + \frac{\partial}{\partial p_y}\left(\frac{\partial \mathbf{z}}{\partial p_x}\right) v^2\cos\theta\sin\theta + \frac{\partial^2 \mathbf{z}}{\partial p_y^2} v^2\sin^2\theta\right),\end{aligned} \tag{16}$$

$$L_g L_f h(\mathbf{x}) u = \boldsymbol{\beta}\left(-\frac{\partial \mathbf{z}}{\partial p_x} v\sin\theta + \frac{\partial \mathbf{z}}{\partial p_y} v\cos\theta\right) u. \tag{17}$$

Algorithm 1: CBFSteer($\boldsymbol{\beta}, n_{near}, v, \theta_{x_s}, v$)

Initialization: i=0, steps=4, $\theta = \theta_{x_s}$
while $i < steps$ **do**
  $i \leftarrow i + 1$;
  $n_{new} \leftarrow$ `Extend` (n_{near}, θ, v);
  $\omega \leftarrow$ `CBF-QP` $(n_{new}.p_x, n_{new}.p_y, \theta, v)$;
  $\theta_{new} \leftarrow$ `AngleUpdate` (ω, θ);
  $n_{new} \leftarrow$ `Extend` $(n_{near}, \theta_{new}, v)$;
  $node\_list \leftarrow$ `AddNode` (n_{new});
  $\theta = \theta_{new}$
end
Return: $n_{new}, node\_list$

Hence, the resultant CBF constraint for a single obstacle i is determined by:

$$\begin{aligned}\varsigma_i(\mathbf{x}, u) = \boldsymbol{\beta}_i&\left(\frac{\partial^2 \mathbf{z}}{\partial p_x^2} v^2\cos^2\theta + \frac{\partial}{\partial p_x}\left(\frac{\partial \mathbf{z}}{\partial p_y}\right) v^2\sin\theta\cos\theta\right. \\ &+ \frac{\partial}{\partial p_y}\left(\frac{\partial \mathbf{z}}{\partial p_x}\right) v^2\cos\theta\sin\theta + \frac{\partial^2 \mathbf{z}}{\partial p_y^2} v^2\sin^2\theta \\ &+ \left(-\frac{\partial \mathbf{z}}{\partial p_x} v\sin\theta + \frac{\partial \mathbf{z}}{\partial p_y} v\cos\theta\right) u \\ &\left. + k_1\left(\frac{\partial \mathbf{z}}{\partial p_x} v\cos\theta + \frac{\partial \mathbf{z}}{\partial p_y} v\sin\theta\right) + k_0\mathbf{z}\right) \geq 0\end{aligned} \tag{18}$$

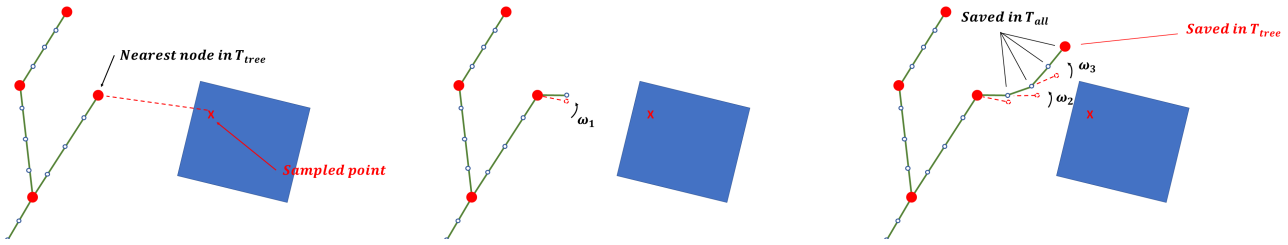
where k_0, k_1 are user-determined CBF coefficients in the form of $k_1 = \alpha_2, k_0 = \alpha_1\alpha_2$. If there are N obstacles, the CBF-QP controller can be derived:

$$\begin{aligned}\min_{u} \quad & ||u - u_{ref}||^2 \\ \text{s.t} \quad & \varsigma_i(\mathbf{x}, u) \geq 0 \quad \text{for } i = 1, ..., N_{obs} \\ & u_{\min} \leq u \leq u_{\max}\end{aligned} \tag{19}$$

where u_{ref} is a reference angular velocity command.

D. Implementation of the CBF-RRT* Algorithm

Based on the CBF-QP, we formulate a steering procedure in RRT* every time it samples a new point on the map. Different from the steering method in the CBF-RRT* algorithm presented in [18], our steering algorithm splits one big step into four small steps, and each of their headings is controlled by the angular velocity calculated through solving the CBF-QP in (19). The structure of the steering method is shown in Algorithm 1. Our proposed multi-step steering algorithm requires two node lists to save path data: tree node list (T_{tree}), that only saves the node after the tree has grown one outer iteration in each loop; all node list (T_{all}), that saves all four nodes where the tree grows one outer iteration step, is used for tracking real motion trajectory. First, we sample points in the map, and select nearest node in T_{tree}, see Fig. 4a. Then we calculate the position where the robot may go in the direction of the sample point for the first small step and solve its CBF-QP, which generates a control ω and may change the robot's heading and walk to a new node, see Fig. 4b. After that, we solve QP with the position along the



(a) (b) (c)

Fig. 4. The green lines represent the existing path or tree. T_{tree} only saves red nodes, and T_{all} saves all nodes in the tree. The blue square is an obstacle. (a). Sample a point in a map and find the nearest node in T_{tree} in the tree. (b). The red dashed point is the position that the robot plans to go. By solving CBF-QP in this position, a control input ω is generated and drives the robot in a new direction. (c). Repeat the process until all four steps have grown. The red node will be saved in T_{tree}, and four new nodes will be saved in T_{all}.

new heading and repeat the same process until the tree has grown all four steps, see Fig. 4c. All four newly generated nodes would be saved in T_{all}, and the last of four nodes would be saved in T_{tree}. By introducing this method, the proposed algorithm can effectively avoid the situation that CBFs become infeasible, and remain the robot safe.

The overall procedure of the proposed CBF-RRT* is shown in Algorithm 2. After saving the nodes into T_{all} and T_{tree}, `ChooseParent` and `Rewrite` methods are used to change the nodes' parents to optimize the path. In particular, we use the simple collision-check function in these two procedures to improve the computational efficiency of the algorithm. The code is developed in Python3.

Algorithm 2: CBF-RRT*

Input: $\mathcal{M}, \boldsymbol{\beta}, n_{init}, n_{goal}, N_{iter}$
{The map features, parameters of a barrier function, initial position, goal position, max iterations}
Initialization: $i = 0$, $T_{all} = \{n_{init}\}$, $T_{tree} = \{n_{init}\}, v = velocity$
while $i < N_{iter}$ **do**
  $i \leftarrow i + 1$;
  $x_s \leftarrow$ `Sampling` $(\mathcal{M})$;
  $n_{near} \leftarrow$ `Nearest` (T_{tree}, x_s);
  $\theta_{x_s} \leftarrow$ `Atan2` (x_s, n_{near});
  $n_{new}, node\_list \leftarrow$ `CBFSteer` $(\boldsymbol{\beta}, n_{near}, v, \theta_{x_s})$;
  $T_{tree} \leftarrow$ `AddNode` (n_{new});
  for *each* **in** *node_list* **do**
    $Ind_{near} \leftarrow$ `NearIndex` $(each, T_{all})$;
    $each \leftarrow$ `ChooseParent` (T_{all}, Ind_{near});
    $T_{all} \leftarrow$ `AddNode` $(each)$;
    $T_{all} \leftarrow$ `Rewrite` $(T_{all}, each, Ind_{near})$;
    if `NearGoal` *(each)* **then**
      $path \leftarrow$ `FinalPath` $(T_{all}, each, n_{goal})$;
  end
end

IV. Simulations and Hardware Experiments

We evaluate the effectiveness of the proposed work through several simulation and experimental tests. To start with, we simulated generating barrier functions and a safe path in an imagined environment with simple polygon shape obstacles. Then, we construct a map of an actual lab room using depth cameras and LiDAR of the Digit robot. The proposed algorithm can effectively identify obstacles in the middle of the room and surrounding desks and construct appropriate barrier functions. The CBF-RRT* then generates a safe path that enables the Digit robot to navigate between two chairs randomly placed in the room[1].

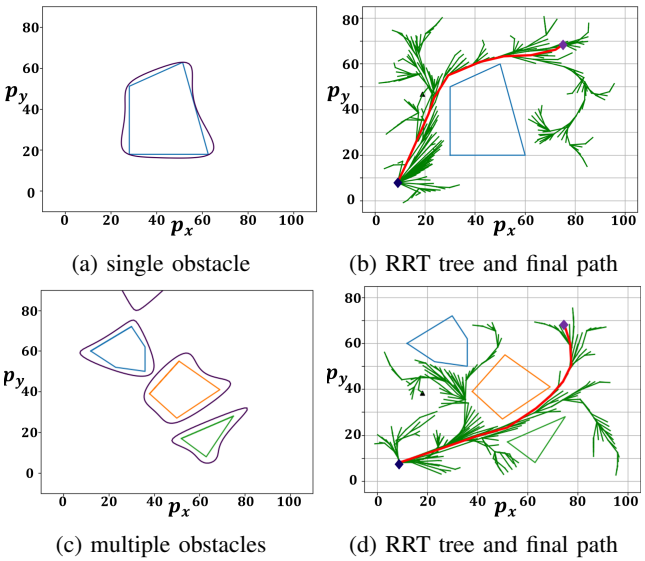
A. Simulation Results

In this test, we consider two imaginary environments with $N_{obs} = 1$ and $N_{obs} = 3$, respectively. The obstacles and their barrier functions are shown in Fig. 5a and Fig. 5c. The time to construct barrier functions in each case is 0.64s and 1.84s. Before running the algorithm, we set the iteration upper bound $N_{iter} = 120$. In order to keep the robot safe, we expand the obstacles with safety distance ($ds = 0.2m$). We set CBF parameters $k_0 = 4, k_1 = 1$. The robot velocity is 0.2m/s, and it starts at (0.9, 0.8)m and the goal is located at (7.45, 6.8)m. The path traced by CBF-RRT* is shown in Fig. 5b, and Fig. 5d. Due to the randomness of the algorithm, we run it 15 times for each case and calculate the average time cost and iteration for the algorithm to generate the first trajectory. For $N_{obs} = 1$, its average time cost is 13.02s with 40.13 average iterations. For $N_{obs} = 3$, its average time cost is 12.02s with 34.86 average iterations. The box-plot of these two cases is shown in Fig. 8

B. Hardware Experiments with Digit

Digit has an integrated perception system that includes three depth modules (Intel RealSense D430), one RGB-Depth module (Intel RealSense D435), one color camera (The Imaging Source DFM 27UP), and one LiDAR sensor (Velodyne LiDAR Puck VLP-16). The robot can walk robustly with a maximum velocity of 0.5 m/s. The locomotion can be controlled by either way-point or velocity commands. In our experiment, we used two pelvis depth modules and the LiDAR sensor to build the room map and used the way-point command to control the robot. In our experiment, we used two pelvis depth modules and the LiDAR sensor to build the room map and used the way-point command to control the robot to walk.

[1]Experiment recordings are shown in the following video: https://youtu.be/r_hkuK5cMw4



(a) single obstacle (b) RRT tree and final path

(c) multiple obstacles (d) RRT tree and final path

Fig. 5. Simulated tests for polygon shape obstacles.

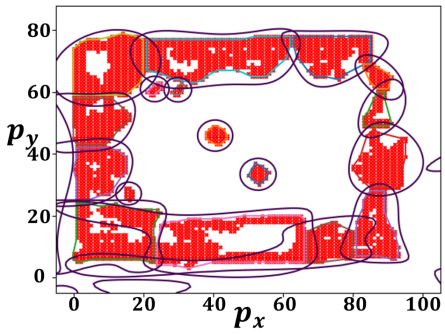


Fig. 6. The generation of barrier functions from real-world 2D occupancy map.

In this test, we first generated an occupancy map of the room from the point cloud data obtained from the depth cameras and LiDAR. The Random Sample Consensus method is used to segment point clouds to distinguish between obstacles and free space. The resulting obstacle points are projected onto a 2D plane, and the cells containing projected points are considered occupied by obstacles. To improve the efficiency of constructing barrier functions, we divided the map into multiple regions and generated the barrier functions for each region respectively. Fig. 6 shows the occupancy map of the room and generated barrier functions of obstacles. The time to construct these obstacles is 3.03s. We set the robot starting at (6.5, 4.0)m and the goal located at (3.6, 3.0)m, and run the algorithm 15 times. The average time cost is 14.26s with 32.03 average iterations. The box-plot of the hardware test is shown in Fig. 8 In the path following phase, we set Digit's velocity to $0.1m/s$, and send the way-point in the path to Digit.Fig. 1 shows the snapshots of the robot following the path and avoiding obstacles.

C. Analysis

One of the major focuses of this work is avoiding the polynomial shape obstacles and generating a safety path

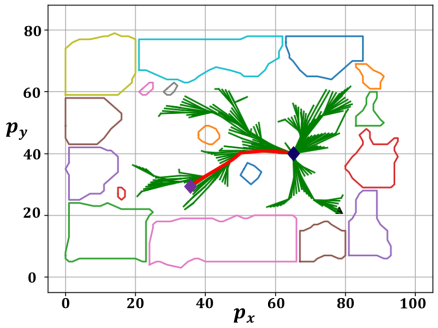


Fig. 7. Collision-free safe path generated by the proposed CBF-RRT*.

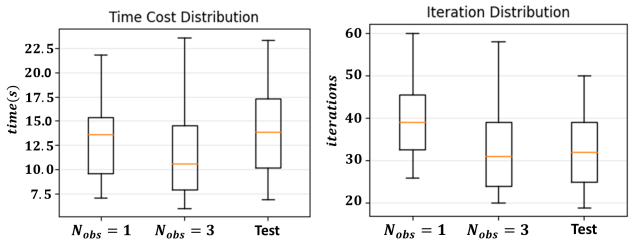


Fig. 8. The time cost and iteration distribution of three tests with 15 times

on CBF-based RRT*. Through the above simulations and experimental tests, we validated that the proposed algorithm can construct appropriate CBFs for arbitrary shape obstacles and generate a collision-free path effectively. From Fig. 8, we find that the obstacle which relatively greatly blocks the potential path to the goal will cause higher time cost and iterations. In addition, the large number of CBFs would slow the code's computational speed, but it can still generate paths in 15s with 40 iterations. Compared to other CBF-RRT/RRT* methods, our method can consider efficiency while making the path optimized and the resulting path shows its feasibility for Digit in the real environment test. However, we also notice the low Digit's walking velocity in the path-following phase, and incoherent following progress, which may be due to the way-point command.

V. Conclusions

In this paper, we introduced a new framework of CBF-RRT* with the ability to generate a collision-free path for complex-shaped obstacles. We also demonstrated the feasibility of the algorithm in real-world environments through hardware experiments. However, our algorithm is still insufficient in some cases, such as obstacles occupying too little space to produce suitable CBFs; unable to express surrounding obstacles with a single CBF, like Fig. 6. In future work, we plan to explore improved algorithms in graphics and machine learning to address these issues and improve performance in practice. We will further improve the computational efficiency of the algorithm so that it can also be applied to safe real-time navigation. Moreover, the differential drive model may limit the agility of bipedal robots. Hence, we will explore more suitable model representations for path planning that unlocks the potential of bipedal robots.

REFERENCES

[1] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, "Path planning and trajectory planning algorithms: A general overview," *Motion and operation planning of robotic systems*, pp. 3–27, 2015.

[2] N. Sleumer and N. Tschichold-Gürmann, "Exact cell decomposition of arrangements used for path planning in robotics," *Technical Report/ETH Zurich, Department of Computer Science*, vol. 329, 1999.

[3] Y. Xue and J.-Q. Sun, "Solving the path planning problem in mobile robotics with the multi-objective evolutionary algorithm," *Applied Sciences*, vol. 8, no. 9, p. 1425, 2018.

[4] J. Bruce and M. M. Veloso, "Real-time randomized path planning for robot navigation," in *Robot soccer world cup*. Springer, 2002, pp. 288–295.

[5] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[6] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez, "Lqr-rrt*: Optimal sampling-based motion planning with automatically derived extension heuristics," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 2537–2542.

[7] G. Goretkin, A. Perez, R. Platt, and G. Konidaris, "Optimal sampling-based planning for linear-quadratic kinodynamic systems," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 2429–2436.

[8] T. Schouwenaars, J. How, and E. Feron, "Receding horizon path planning with implicit safety guarantees," in *Proceedings of the 2004 American control conference*, vol. 6. IEEE, 2004, pp. 5576–5581.

[9] J.-K. Huang, Y. Tan, D. Lee, V. R. Desaraju, and J. W. Grizzle, "Informable Multi-Objective and Multi-Directional RRT* System for Robot Path Planning," *arXiv preprint arXiv: Arxiv-2205.14853*, 2022.

[10] W. Xiao, C. G. Cassandras, C. A. Belta, and D. Rus, "Control barrier functions for systems with multiple control inputs," in *2022 American Control Conference (ACC)*, 2022, pp. 2221–2226.

[11] 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.

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

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

[14] S. Teng, Y. Gong, J. W. Grizzle, and M. Ghaffari, "Toward safety-aware informative motion planning for legged robots," *arXiv preprint arXiv:2103.14252*, 2021.

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

[16] G. Yang, B. Vang, Z. Serlin, C. Belta, and R. Tron, "Sampling-based motion planning via control barrier functions," in *Proceedings of the 2019 3rd International Conference on Automation, Control and Robots*, 2019, pp. 22–29.

[17] A. Manjunath and Q. Nguyen, "Safe and robust motion planning for dynamic robotics via control barrier functions," in *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE, 2021, pp. 2122–2128.

[18] A. Ahmad, C. Belta, and R. Tron, "Adaptive sampling-based motion planning with control barrier functions," in *2022 IEEE 61st Conference on Decision and Control (CDC)*. IEEE, 2022, pp. 4513–4518.

[19] J. Liu, M. Li, J.-K. Huang, and J. W. Grizzle, "Realtime Safety Control for Bipedal Robots to Avoid Multiple Obstacles via CLF-CBF Constraints," *arXiv preprint arXiv:2301.01906*, 2023.

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

[21] W. Xiao and C. Belta, "Control barrier functions for systems with high relative degree," in *2019 IEEE 58th conference on decision and control (CDC)*. IEEE, 2019, pp. 474–479.

[22] I. Noreen, A. Khan, and Z. Habib, "Optimal path planning using rrt* based approaches: a survey and future directions," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 11, 2016.

[23] D. Pregibon, "Logistic regression diagnostics," *The annals of statistics*, vol. 9, no. 4, pp. 705–724, 1981.

[24] J. Feng, H. Xu, S. Mannor, and S. Yan, "Robust logistic regression and classification," *Advances in neural information processing systems*, vol. 27, 2014.