Sampling-based Motion Planning via Control Barrier Functions

Guang Yang
Boston University
750 Commonwealth Avenue
Brookline,MA,02446
gyang101@bu.edu

Bee Vang
Boston University
750 Commonwealth Avenue
Brookline,MA,02446
bvang@bu.edu

Zachary Serlin
Boston University
750 Commonwealth Avenue
Brookline,MA,02446
zserlin@bu.edu

Calin Belta
Boston University
750 Commonwealth Avenue
Brookline,MA,02446
cbelta@bu.edu

Roberto Tron
Boston University
750 Commonwealth Avenue
Brookline,MA,02446
rtron@bu.edu

ABSTRACT

Robot motion planning is central to real-world autonomous applications, such as self-driving cars, persistence surveillance, and robotic arm manipulation. One challenge in motion planning is generating control signals for nonlinear systems that result in obstacle free paths through dynamic environments. In this paper, we propose Control Barrier Function guided Rapidly-exploring Random Trees (CBF-RRT), a sampling-based motion planning algorithm for continuous-time nonlinear systems in dynamic environments. The algorithm focuses on two objectives: efficiently generating feasible controls that steer the system toward a goal region, and handling environments with dynamical obstacles in continuous time. We formulate the control synthesis problem as a Quadratic Program (QP) that enforces Control Barrier Function (CBF) constraints to achieve obstacle avoidance. Additionally, CBF-RRT does not require nearest neighbor or explicit collision checks during sampling.

CCS Concepts

•Information systems → Mobile information processing systems;

Keywords

Motion Planning; Sampling; Control

1. INTRODUCTION

Motion planning is a cornerstone of modern robotics, but it is still a challenging problem when non-trivial robot dynamics are combined with input constraints and dynamic obstacles. Control Barrier Functions(CBFs) have been shown to be effective for feedback control in similar settings; however,

they cannot be used alone for producing *complete* planners, as they can get trapped in "local minima" created by obstacles. CBFs can also become infeasible if there is no possible control that allows the robot to remain safe (i.e., to avoid collision). We look to overcome CBF limitations by generating and connecting many smaller, feasible trajectories through sampling. This results in a complete CBF-based planner that can find trajectories not considered by CBFs alone.

Sampling-based motion planning is not a novel concept, however, much of the literature in this area tends to focus on either high-level path planning, or lower level control and trajectory planning. In this paper, we attempt to present a unified approach that takes advantage of both collision free and efficient trajectory planning, as well as efficient path planning. Using CBFs developed in the formal methods and controls community, we augment the classic rapidly exploring random trees algorithm to handle dynamic obstacles and non-linear robot dynamics while also generating controls that are guaranteed to produce safe, collision free, trajectories.

1.1 Related Work

This work draws in past work on both sampling-based algorithms in the motion planning literature, as well as provably correct control synthesis in the formal methods for robotics community.

Motion planning in real-world applications often considers high level path planning and low level control synthesis, given safety requirements and dynamical constraints. Sampling based motion planning algorithms, such as Probabilistic Road Map [1], Rapidly-exploring Random Trees (RRT) [2], and RRT*[3], have been widely explored and are efficient strategies for high dimensional kinematic planning; however, generally these algorithms assume that a low level controller exists to generate collision free trajectories at run time. In recent years, there has been considerable effort to try to bridge the gap between path planning and control synthesis by designing controller that steer the system in-between two generated vertices, such as Kinodynamic RRT* [4], LQR-RRT* [5] and its variants in [6] and [7], however these approaches are limited to linear systems and static environments. Our proposed CBF-RRT method takes a different

approach compared to the work in [4], [5], and [6]. Instead of first generating paths with a fixed number of vertices and then treating control synthesis as a two point boundary value problem (as in [5]), we generate both controls and paths "on-the-fly" and work with nonlinear systems. Motion planning in dynamical environments has been studied in [8] and [9], but focuses on replanning when obstacles cause collisions. Considerable work has been done on generating control strategies for safety-critical systems using CBFs. A popular formulation is to combine CBFs and Control Lyapunov Functions (CLFs) in a Quadratic Program (QP) where the CBF ensures safety and the CLF ensures stability. This approach has been successfully applied to applications such as adaptive cruise control [10], bipedal robot walking [11], and swarm control [12]. The QP based formulation works well when the desired equilibrium point is well-defined and there exists a feasible control sequence for the given problem. Under certain conditions, however, the QP may be infeasible due to the environment or dynamical constraints. Work has also been done, in [13], in designing a safe controller that follows a pre-planned path, but does not include a path-planner that explicitly explores the environment, and therefore also, occassionally, leads to infeasibility in finding a solution.

Contributions:.

In this paper, we propose Control Barrier Function guided Rapidly-exploring Random Trees, a motion planning algorithm that uses sampling techniques to explore the state space and a QP based controller with CBF constraints to generate intermediate controls and trajectories between samples. From the sampling-based motion planning point of view, this paper provides formal guarantees for collision free continuous trajectories between samples. From a formal methods in robotics point of view, this paper offers a partial solution to infeasibility in finding a solution in CBF/CLF QP based controller formulations. The proposed framework guarantees safety, handles obstacles with known dynamics, and its internal control synthesis can be utilized as the low level controller at run-time.

This paper is organized as follows: preliminary information is provided in Section 2; the problem statement and proposed algorithm are presented in Section 3 and Section 4, respectively; a differential drive model is considered in Section 5 followed by simulations in Section 6; Conclusions are given in Section 7.

2. PRELIMINARIES

2.1 Notation

Let $\mathbb{R}^n$ be the set of real numbers in n dimensions. The Lie derivative of a smooth function $h(x(t))$ along dynamics $\dot{x}(t) = f(x(t))$ is denoted as $\mathcal{L}_f h(x) := \frac{\partial h(x(t))}{\partial x(t)} f(x(t))$. Given a continuously differentiable function $h : \mathbb{R}^n \mapsto \mathbb{R}$, we denote h^{r_b} as its r_b-th derivative with respect to time t. A continuous function $\alpha : (-b, a) \mapsto (-\infty, \infty)$, for some $a, b > 0$, belongs to the extended class $\mathcal{K}$ if α is strictly increasing and $\alpha(0) = 0$. The set difference between set A and set B is denoted as $A \backslash B$.

2.2 Dynamics

We consider a motion planing problem for a continuous-time control system

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

where $x \in \mathcal{X} \subset \mathbb{R}^n$ is the state and $u \in \mathcal{U} \subset \mathbb{R}^m$ is the control input, where $\mathcal{U}$ is a set of admissible controls for system (1). The functions $f(x)$ and $g(x)$ are assumed to be locally Lipschitz continuous. The initial state is denoted as $x_{init} := x(t_0) \in \mathcal{X}$ and the goal region is defined as $\mathcal{X}_{goal} \subset \mathcal{X}$. Obstacles are assumed to be non-stationary with known dynamics and move according to the equation

$$\dot{x}_{obs} = f_{obs}(x_{obs}), \tag{2}$$

where $x_{obs} \in \mathcal{X}_{obs} \subset \mathcal{X}$ is the state variable for the obstacles (e.g., center of mass). The function f_{obs} is again assumed to be locally Lipschtiz continuous.

REMARK 1. *For special cases where the obstacle dynamics are unknown, we could approximate their dynamics with techniques, such as Gaussian Process (GP)[14] or online Linear Regression[15]. For simplicity, in this paper, we assume that all obstacle dynamics are known.*

2.3 Exponential Control Barrier Functions

In traditional path planning approaches, obstacle avoidance is often enforced by a collision check along segments or regions of the path. To create a more complete and provably safe trajectory, we instead formulate obstacle avoidance as remaining within a safety set defined by Control Barrier Functions (CBFs, [10]) and their extensions for higher relative degree, Exponential CBFs [16]. Given a continuously differentiable function $h : \mathbb{R}^n \mapsto \mathbb{R}$, the safety set C is defined as

$$C = \{x \in \mathbb{R}^n | h(x) \geq 0\}.$$
$$\partial C = \{x \in \mathbb{R}^n | h(x) = 0\}, \tag{3}$$
$$Int(C) = \{x \in \mathbb{R}^n | h(x) > 0\}$$

where ∂C is the boundary and $Int(C)$ is the interior. The set C is called *forward invariant* for system (1) if $x_0 \in C$ implies $x(t) \in C$ for all t.

For systems with relative degree r_b, we define a traverse variable as

$$\xi_b(x) = \left[h(x), \dot{h}(x), \ldots, h^{(r_b)}(x)\right]^T$$
$$= \left[h(x), \mathcal{L}_f h(x), \ldots, \mathcal{L}_f h^{r_b}(x)\right]^T,$$

and formulate a virtual system with input-output linearization [17]:

$$\dot{\xi}_b(x) = A_b \xi_b(x) + B_b \mu, \tag{4}$$
$$h_l(x) = C_b \xi_b(x),$$

where

$$A_b = \begin{bmatrix} 0 & 1 & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & \cdot & 1 \\ 0 & 0 & 0 & \cdot & 0 \end{bmatrix}, B_b = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix},$$
$$C_b = [1 \ldots 0],$$

and $\mu = (\mathcal{L}_g \mathcal{L}_f^{r_b - 1} h(x))^{-1}(\mu - \mathcal{L}_f^{r_b} h(x))$ is the input-output linearized control.

DEFINITION 1. *Consider the dynamical system in (1) and the safety set C defined in (3). A continuously differentiable function $h(x)$ with relative degree $r_b \geq 1$ is an Exponential*

Control Barrier Function (ECBF) [16] if there exists $K_b \in \mathbb{R}^{1 \times r_b}$, such that

$$\inf_{u \in U} [\pounds_f^{r_b} h(x) + \pounds_g \pounds_f^{r_b - 1} h(x) u + K_b \xi_b(x)] \geq 0, \forall x \in Int(C), \tag{5}$$

where the row vector K_b is selected such that the closed-loop matrix $A_b - B_b K_b$ for (4) is stable.

THEOREM 1. *Given the system (1), and the safety set C defined in (3), if there exists an ECBF $h(x)$, then the system is forward invariant in C [16].*

REMARK 2. *There exist a more general notion of Higher Order Control Barrier Function (HOCBF) [18], of which ECBF is a special case. For the purposes of this paper, we only need ECBFs, although the use of HOCBFs could lead to improved performance.*

2.4 Rapidly-exploring Random Trees (RRT)

We use $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ to denote a tree with a set of vertices $\mathcal{V} \subseteq \mathcal{X}$ and a set of edges $\mathcal{E}$. The classic Rapidly-exploring Random Trees algorithm (RRT, [2]), builds a tree $\mathcal{T}$ where each vertex $v \in \mathcal{V}$ is associated to a full state $v = x \in \mathbb{R}^n$, and edges follow the system dynamics (1). The algorithm needs the following essential components:

- **State Space and Goal Region**: A configuration space $\mathcal{X}$, such that $x \in \mathcal{X}$, and a goal region $\mathcal{X}_{goal} \subset \mathcal{X}$.
- **Collision Check**: A function $\texttt{CollisionCheck}(x)$ that detects if a state trajectory x violates any collision constraint.
- **Metric**: A function $d : \mathcal{X} \times \mathcal{X} \rightarrow [0, \infty)$ that returns the distance between two vertices within $\mathcal{X}$.
- **Nearest Neighbor**: A function $\texttt{NearestNeighbor} : (\mathcal{T}, x_{sample}) \mapsto x_{nn}$ which utilizes the distance function d to find the vertex x_{nn} in $\mathcal{T}$ that is closest to x_{sample}.
- **Inputs**: A set of admissible controls U, such that $u \in U$, for steering the state x.
- **Steer**: As shown in Figure 1, given a sampled vertex x_{sample} and nearest vertex x_{nn}, $\texttt{Steer}(x_{nn}, x_{sample})$ guides the system from x_{nn} to x_{sample} with control u while performing collision checks. The function terminates after the trajectory progresses some fixed distance and returns false if no trajectory is possible.

The RRT algorithm is summarized in Algorithm 1 and Algorithm 2.

Algorithm 1 RRT

$\mathcal{T} = (\mathcal{V}, \mathcal{E})$, $\mathcal{V} \leftarrow \{x_{init}\}$; $\mathcal{E} \leftarrow \emptyset$;
while $x \notin \mathcal{X}_{goal}$ **do**
    $x_{sample} \leftarrow \texttt{RandomState}(\mathcal{V})$
    $\mathcal{T} \leftarrow \texttt{Extend}(\mathcal{T}, x_{sample})$
return $\mathcal{T}$

Algorithm 2 Extend($\mathcal{T}, x_{sample}$)

$x_{nn} \leftarrow \texttt{NearestNeighbor}(\mathcal{T}, x_{sample})$;
if Steer($x, x_{nn}, x_{new}, u_{new}$) **then**
    $\mathcal{V}.\texttt{AddVertex}(x_{new})$
    $\mathcal{E}.\texttt{AddEdge}(x_{new}, u_{new}, x_{nn})$
    if $x_{new} = x_{sample}$ **then**
        return $(\mathcal{V}, \mathcal{E})$
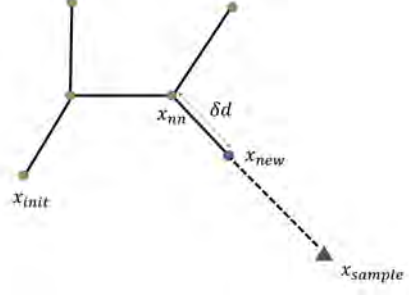   **else**
        return $\mathcal{T}$
return Trapped

Figure 1: $\texttt{Steer}$ function in classic RRT

There are three possible outcomes from the $\texttt{Steer}$ function: $\texttt{Reached}$ indicates that $x_{new} = x_{sample}$ can be reached under the dynamical and obstacle constraints, $\texttt{Advanced}$ indicates that $\|x_{sample} - x_{new}\| \leq \delta d$, where δd is a positive constant, $\texttt{Trapped}$ indicates that the function cannot find a feasible control u to steer the system. The $\texttt{Steer}$ function is demonstrated in Figure 1.

3. PROBLEM STATEMENT

PROBLEM 1. *Consider a nonlinear system in the form of (1), with initial state $x_{init} \in \mathcal{X}_{init} \subset \mathcal{X} \subset \mathbb{R}^n$, where $\mathcal{X}_{init}$ is an initial obstacle free set, and a bounded goal region $\mathcal{X}_{goal}$, generate feasible control inputs $u(t)$ that steer the system to $\mathcal{X}_{goal}$ while avoiding dynamical obstacles.*

To approach the problem, we combine both obstacle dynamics and system dynamics into a composite system. By treating the obstacles' state as part of the composite system, we can effectively construct the CBF constraints for the QP controller.

4. CBF-RRT

The CBF-RRT algorithm is different from other RRT variants in that there is no explicit collision or nearest neighbor checks. Instead, we introduce the notion of safe steering that encodes collision avoidance as staying within the safety set (3). The nearest neighbor check is implicitly handled by sampling the vertices in $\mathcal{V}$.

Before we formally introduce the algorithm, we define the following components:

- **State Space**: A topological space $\mathcal{X} \subset \mathbb{R}^n$, and $\mathcal{X}_{goal}$, $\mathcal{X}_{obs} \subset \mathcal{X}$.
- **Inputs**: A set of admissible controls U for steering the state x.
- **Safe Steering**: A function that generates both safe controls and trajectories in a time interval t_h, given the system dynamics (1) and obstacle dynamics (2). More details are introduced in Section 4.2.

In general, the vertices are sampled points of the state space, $\mathcal{V} \subseteq \mathcal{X}$ and we assume $\mathcal{X}_{goal} \backslash \mathcal{X}_{obs}$. To account for the time-varying obstacles, the vertices need to store an additional parameter that determines when, in time, the vertices are safe. In other words $\mathcal{V} \subseteq \mathcal{X} \times \mathbb{R}$ where the additional parameter is time and an element $v = (x, t) \in \mathcal{V}$. We use $\mathcal{T} = (\mathcal{V} \subseteq \mathcal{X} \times \mathbb{R}, \mathcal{E})$.

4.1 Sampling

4.1.1 Vertices

The function `VerticesSample` : $v_s \in \mathcal{V}$ samples, with a desired probability distribution, $p_v(\mathcal{V})$, on the existing vertices of the tree $\mathcal{T}$. By varying the probability distributions, the behavior of the tree expansion, as well as convergence speed, will be drastically different.

4.1.2 State Space

Depending on the problem, some state variables may not play an essential role in the task requirements and/or system dynamical constraints. We denote these variables as free state variables that can be arbitrarily chosen to increase the probability of finding a path that satisfies the dynamics and safety constraints. We define function `StateSample` : $v_s \mapsto v_e$ that updates the free state variables of a given vertex with a probability distribution. We denote this probability distribution as $p_{state}(x)$ and define v_e as the expanding vertex. For example, if the task is to steer a nonholonomic first-order planar robot, such as a unicycle, from one position to another and orientation does not matter. Then, the vertices only need to contain information relevant to the position of the robot, i.e. $\mathcal{V} \in \mathbb{R}^2 \times \mathbb{R} \subset \mathbb{R}^3 \times \mathbb{R}$.

4.1.3 Control Reference

If no free states exist in the system dynamics, the u_{ref} value must be sampled from a distribution $p_{u_{ref}}$ in order to maintain the probabilistic nature of CBF-RRT. In this case, we define a function `ReferenceSample` : $u_{ref} \in U$ that updates the u_{ref} for the chosen sample from $p_{u_{ref}}$.

4.2 Safe Steering

We define a function `SafeSteer`(v_0, t_h, u_{ref}) that contains two components: controls synthesis and collision-free trajectory generation. Given an initial vertex v_0 which includes an initial state x_0 and time element t_0, a fixed time horizon t_h, a control reference u_{ref}, obstacle dynamics (2) and system dynamics (1), `SafeSteer` solves a sequence of QPs (Section 4.4) with CBF constraints and generate a sequence of control inputs $u(t)$. The control inputs generate a collision-free trajectory to x_{new} at time $t_0 + t_h$ which is added to the tree $\mathcal{T}$ as a new vertex.

REMARK 3. *The control reference u_{ref} determines how the robot explores the space and the QP ensures the robot's safety while doing so by modifying u_{ref} as necessary. For example, if the reference command u_{ref} is defined as "go forward in the body-fixed frame", then the QP's job is to steer the robot away from obstacles when necessary. If there are no obstacles, the robot should move in a straight line.*

4.3 Goal Check

Given a desired goal x_{goal}, we define a goal region $\mathcal{X}_{goal} = \{y \in \mathbb{R}^n : d(y, x_{goal}) \leq \epsilon\}$, where ϵ is a positive constant chosen such that $\mathcal{X}_{goal}$ is obstacle free. If the trajectory $x(t) \in \mathcal{X}_{goal}$, then the algorithm terminates and a path is found. Otherwise, the algorithm continues.
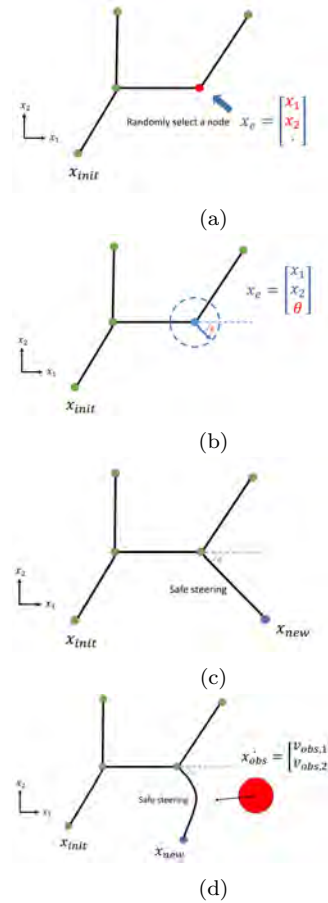

Figure 2: Illustration of CBF-RRT for a system with states $x = [x_1, x_2, \theta]^T$ where θ is a free variable: The algorithm picks a random vertex within tree $\mathcal{T}$ and sets it as the sampled vertex v_s. Next, it performs another sampling on the state variable θ under state distribution p_{state}. The steering function then generates a sequence of controls u to steer the system while avoiding collisions. (a) Selection of a random vertex in $\mathcal{V}$. (b) Selection of random state θ. (c) Safe steering trajectory when no nearby obstacle present. (d) Safe steering trajectory with dynamic obstacle present.

4.4 Quadratic Program Formulation

The QP based controller takes in a reference control u_{ref}, current state of the system $x(t)$ and obstacle $x_{obs}(t)$ as inputs and finds a feasible control $u(t)$ point-wise in time that tries to follow u_{ref} while maintain safety, as shown in Figure 2. Given a safety set $h_i(x)$ and a system with relative degree r_b, we define the i-th CBF constraint as

$$\zeta_i(x) = \pounds_f^{r_b} h_i(x) + \pounds_g \pounds_f^{r_b - 1} h_i(x) u + K_{b,i} \xi_{b,i}(x) \geq 0.$$

Additional linear constraints (with respect to u) may be considered, such as control bounds. These requirements may be formulated as a QP as follows:

$$\begin{aligned} \min_{u \in U} \quad & \|u - u_{ref}\|^2 \\ \text{s.t.} \quad & \zeta_i(x) \geq 0, \quad i = 1, ..., N_{obs} \\ & \underline{u} \leq u \leq \overline{u}, \end{aligned} \quad (6)$$

where $\underline{u}, \overline{u}$ are the lower and upper control bounds, respectively. The CBF constraint ζ_i is linear in terms of decision

variable u and N_{obs} is the total number of obstacles.

Algorithm 3 CBF-RRT

1: $\mathcal{V} \leftarrow \{(x_{init}, t_{init})\}; \mathcal{E} \leftarrow \emptyset$; ▷ Initialize first vertex to initial state.
2: $\mathcal{X}_{goal} \subset \mathcal{X}$ ▷ Define goal region.
3: **while** $x \notin \mathcal{X}_{goal}$ **do**
4:    $v_s \leftarrow$ VerticesSample($\mathcal{V}, p_v$) ▷ Sample a vertex in $\mathcal{V}$
5:    $v_e \leftarrow$ StateSample(v_s, p_{state}) ▷ Sample the state at vertex x_s (i.e. θ)
6:    $u_{ref} \leftarrow$ ReferenceSample($p_{u_{ref}}$) ▷ Sample reference control if needed
7:    $\mathbf{u}_{traj}, \mathbf{x}_{traj}, x_{new}, t_{new} \leftarrow$ SafeSteer(v_e, t_h, u_{ref}) ▷ Solve CBF QP
8:     **if** $x_{new} \neq \emptyset$ **then**   ▷ If QP was feasible
9:         $\mathcal{V} \leftarrow (x_{new}, t_{new}), \mathcal{E} \leftarrow \mathbf{x}_{traj}$ ▷ Update the tree
    return $\mathcal{T} = (\mathcal{V}, \mathcal{E}), \mathbf{u}_{traj}$

Algorithm 4 SafeSteer

   Given v_e, t_h, u_{ref}
2: $\zeta_i \leftarrow$ i-th CBF constraint, $\forall i$;
   $\mathbf{u}_{traj}, \quad \mathbf{x}_{traj}, \quad x_{new}, \quad t_{new} \quad \leftarrow \quad$ Integrator(x_e, t_e, t_h, QPcontroller($x(t), x_{obs}(t), u_{ref}$));
4: **return** $\mathbf{u}_{traj}, \mathbf{x}_{traj}, x_{new}, t_{new}$

---

5. NUMERICAL EXAMPLES

In this section, we consider the motion planning problem of steering a planar robot from an initial state x_{init} to a goal position that is independent of orientation. In particular, the goal region $\mathcal{X}_{goal}$ is defined by x_{goal} and ϵ where $x_{goal} = [x_{1,goal}, x_{2,goal}, \theta_{goal}]^T$ and θ_{goal} is arbitrary. Thus, the motion planning problem is solved in the work space $\mathcal{X}_{work} \in \mathbb{R}^2$.

5.1 Dynamical System

Consider a unicycle model for a two-wheeled differential drive robot

$$\dot{x_1} = v\cos(\theta), \quad (7)$$
$$\dot{x_2} = v\sin(\theta),$$
$$\dot{\theta} = \omega,$$

where the state $x = [x_1, x_2, \theta]^T \in \mathbb{R}^3$ corresponds to the location (x_1, x_2) in work space $\mathcal{X}_{work} \subset \mathbb{R}^2$ and heading θ with respect to the inertial frame. The control input $u = [v, \omega]^T \in \mathbb{R}^2$ consists of the translational and angular velocity that are bounded, respectively. The equations of motion (7) can be written in control affine form as

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \omega. \quad (8)$$

5.2 Safety Sets

We consider rigid body obstacles and model them as the union of circles with centroids $(x_{obs,i,1}(t), x_{obs,i,2}(t))$ and fixed radii $r_{obs,i}$ where each obstacle is inscribed by the union of their respective circles. We denote the i-th safety set as

$$C_i = \{x \in R^2 : h_i(x) \geq 0\}, \quad (9)$$

where

$$h_i(x) = (x_1(t) - x_{obs,i,1}(t))^2 + (x_2(t) - x_{obs,i,2}(t))^2 - r_{obs,i}^2. \quad (10)$$

Each circle has the following dynamics

$$\dot{x}_{obs,i,1} = v_{obs,i,1}, \quad (11)$$
$$\dot{x}_{obs,i,2} = v_{obs,i,2}, \forall i.$$

The safe set of the robot is given as the intersection of all the safe sets for the circles

$$C_{robot} = \bigcap_{i=1}^{N_{obs}} C_i. \quad (12)$$

The i-th ECBF constraint is

$$\zeta_i(x) = \mathscr{L}_f^2 h_i(x) + \mathscr{L}_g \mathscr{L}_f h_i(x) u + k_1 h_i(x) + k_2 \mathscr{L}_f h_i(x) \geq 0, \quad (13)$$
$$\forall x \in Int(C_i),$$

where

$$h_i(x) = (x_1 - x_{obs,i,1})^2 + (x_2 - x_{obs,i,2})^2 - r^2 \quad (14)$$

$$\mathscr{L}_f h_i(x) = 2v(x_1 - x_{obs,i,1})\cos\theta + 2v(x_2 - x_{obs,i,2})\sin\theta$$
$$-2(x_1 - x_{obs,i,1})v_{obs,i,1} - 2(x_2 - x_{obs,i,2})v_{obs,i,2},$$

and k_1, k_2 are positive constants that are selected appropriately to ensure forward invariance, as mentioned in [16]. The resultant ECBF constraint is

$$\zeta_i(x) = 2x_1 v^2 \cos^2\theta + 2x_2 v^2 \sin^2\theta \quad (15)$$
$$+(2(x_2 - x_{obs,2})\cos\theta - 2(x_1 - x_{obs,1})\sin\theta)\omega$$
$$+2v_{obs,i,1}^2 + 2v_{obs,i,2}^2 + k_1 h(x) + k_2 \mathscr{L}_f h(x) \geq 0.$$

REMARK 4. *The inequality* (15) *is linear with respect to ω but not v, therefore it cannot be add directly into the QP as a linear constraint. This is due to the mixed-relative degree of the control inputs. To overcome this limitation, we set the translational velocity $v = c \in \mathbb{R}$, which becomes part of the system dynamics function $f(x)$ in* (1).

5.3 Sampling Distributions

Vertex Sampling: Given a set of vertices $\mathcal{V}$ in $\mathcal{T}$, we define a discrete uniform distribution over all vertices

$$p_v = \frac{1}{N_v}, \quad (16)$$

where N_v is the total number of vertices in $\mathcal{V}$. Therefore, each vertex has equal probability to be selected as an expanding vertex.

State Sampling: Given the state of the sampled vertex $[x_{1,s}, x_{2,s}, \theta_{free}]^T$, we first calculate the desired heading angle θ_{goal} toward x_{goal} by

$$\theta_{goal} = \arctan\left(\frac{x_{2,goal} - x_{2,e}}{x_{1,goal} - x_{1,e}}\right)$$

We then define a Gaussian distribution for the state θ_{free}

$$p_{state}(\theta_{free}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\theta_{free} - \theta_{goal})^2}{2\sigma^2}}, \quad (17)$$

where σ^2 is the variance and we set θ_{goal} as the mean.

6. SIMULATIONS

For simulations, the control inputs are $v \in [-1, 1]$ and $\omega \in [-4.25, 4.25]$. We then add (15) as linear constraints in the QP (6) with ω as the only decision variable and $v = 1$ (see Remark 4). We choose $\omega_{ref} = 0$, i.e., the system minimally change its heading direction θ. The combination of (v, ω_{ref}) is chosen such that the robot moves in straight lines as it explores the space. All experiments are performed on a desktop computer with a i7-8700K CPU with the Gurobi 8.1.1 [19] solver.

6.1 Example 1: Static Environment

In the first example, we consider an environment with $N_{obs} = 3$ static obstacles, i.e., $v_{obs,i,1} = v_{obs,i,2} = 0, \forall i = 1, ..., N_{obs}$, which implies $x_{obs,i,1}$ and $x_{obs,i,2}$ are constants in (10). The corresponding ECBF is

$$\zeta_i(x) = 2x_1 v^2 \cos^2(\theta) + 2x_2 v^2 \sin^2(\theta) + k_1 h_i(x)$$
$$+k_2(2x_1 v \cos(\theta) + 2x_2 v \sin(\theta)) + [-2v(x_1 - x_{obs,i,1})\sin(\theta) +$$
$$2v(x_2 - x_{obs,i,2})\cos(\theta)]w \geq 0.$$

The initial state is $x_{init} = [-0.5, -0.5, 1]^T$ and the goal state is $x_{goal} = [2, 2, \cdot]^T$. We define three obstacles with their centroids locate at $(0.3, 1.2)$, $(1.0, 0.5)$, $(1.7, -0.5)$ with a radius $r_i = 0.2, \forall i$. The hyper parameters in Table 1 include ECBF coefficients k_1, k_2, variance σ^2 for state distribution p_{state}, radius of the goal region ϵ and time horizon t_h.
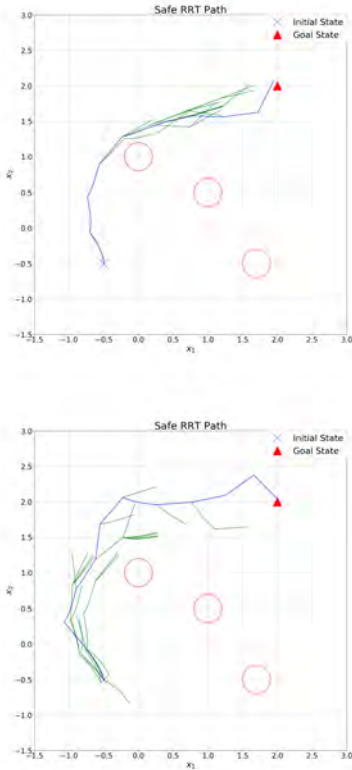
Table 1: Hyper Parameters for Example 1

Case	k_1	k_2	σ^2	ϵ	t_h	v	Run Time
1	2.0	4.0	0.2	0.15	0.5	1.0	20.37s
2	2.0	4.0	0.6	0.15	0.5	1.0	1.81s

In Figure 3, the generated tree curves around the obstacle as the `SafeSteer` function ensures the generated trajectory never enter the obstacle regions. Note case 2 has a higher variance ($\sigma^2 = 0.6$) which result in more exploration in the work space before finding a path to $\mathcal{X}_{goal}$.

6.2 Example 2: Dynamical Environment

In this example, without the loss of generality, we consider multiple obstacles with constant velocity $v_{obs,i,1}, v_{obs,i,2}$ for the i-th obstacle. We formulate a composite system that includes obstacles dynamics (11) as the following

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{\theta} \\ \dot{x}_{obs,i,1} \\ \dot{x}_{obs,i,2} \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ 0 \\ v_{obs,i,1} \\ v_{obs,i,2} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \\ . \\ 0 \end{bmatrix} \omega, \quad (18)$$

$$i = 1, ..., 4. \quad (19)$$

REMARK 5. *The composite system* (18) *can be easily extended to multi-agents path planning problems. In this example, we only consider path planning problem for a single agent.*

The four obstacles are initially located on the edge of the workspace and gradually move toward the center. We set $v_{obs,1,1} = v_{obs,2,1} = 0.08, v_{obs,1,2} = v_{obs,2,2} = 0.3$, $v_{obs,3,1} = v_{obs,4,1} = -0.08, v_{obs,3,2} = v_{obs,4,2} = -0.3$. The additional hyper-parameters can be found in Table 2. The generated tree can be found in Figure 4 and the simulation snapshots are in Figure 5.
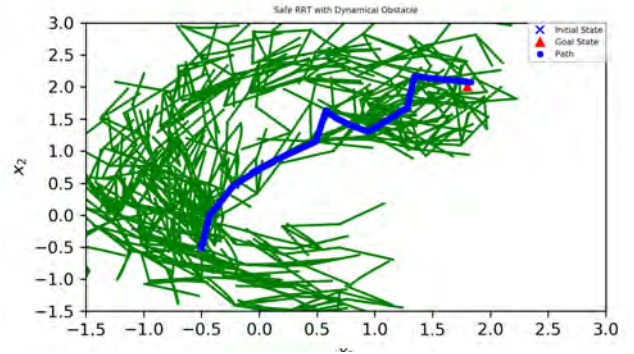

Figure 4: For Example 2, the figure shows the generated tree from CBF-RRT under dynamical environment with 4 moving obstacles. The obstacles are not shown here for easier visuals

Table 2: Hyper Parameters for Example 2

k_1	k_2	σ^2	ϵ	t_h	v	Run Time
1.0	1.9	1.0	0.15	0.5	1.0	32.4s

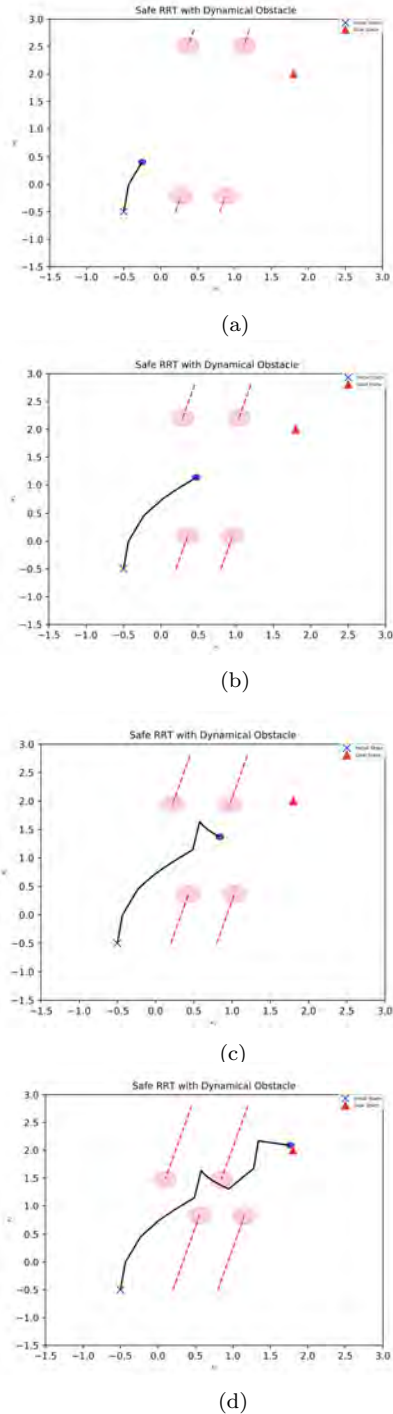Figure 3: Example 1: CBF-RRT Motion Planning

(a)

(b)

(c)

(d)

Figure 5: CBF-RRT snapshots for multi-dynamical obstacles ($\sigma^2 = 1.0$): The plots from (a) to (d) demonstrate how the robot (blue dot) progress as the obstacles (red circles) crosses its path. The `SafeSteer` function steers the robot away from the obstacle such that it can reach the goal without collision. Video:URL link

6.3 Comparison with RRT and RRT*

One of the major advantages of CBF-RRT is to guarantee of collision free trajectory generation in continuous time. In RRT and RRT* path planning, an end-point collision check function typically only checks if the end vertex x_{new} is within an obstacle region. While it is computationally efficient, the generated trajectories have the potential to collide with intermediary obstacles that are smaller than the chosen step size δd (Fig. 1). We compare our proposed CBF-RRT algorithm with both RRT and RRT* in the same static environment from Example 1 and Fig. 6 highlights how trajectories have the potential to collide with the obstacles when δd is chosen to be too large. CBF-RRT however does not suffer from this issue since it considers continuous trajectories and has no such explicit collision check. Table 3 shows the run time and number of vertices comparisons between the CBF-RRT, RRT, and RRT* algorithms. CBF-RRT preforms between RRT and RRT* in terms of both computation time and number of vertices generated. This illustrates that CBF-RRT is able to guarantee collision free trajectories for non-linear dynamics without excessive increases in computation time over less-safe RRT and RRT*. Furthermore, we expect CBF-RRT to outperform RRT and RRT* in highly complex (crowded) environments because it does not need to perform a collision check explicitly.

Table 3: Comparison of CBF-RRT, RRT, and RRT*

Alg.	Step Size	Run Time (s)	Number of Vertices
CBF-RRT	0.25s	2.98	496
RRT	0.25m	0.00125	39
RRT*	0.25m	2.109	488
CBF-RRT	1s	0.281	26
RRT	1m	0.00027	9
RRT*	1m	1.480	494

7. CONCLUSION AND FUTURE WORK

In this paper we present CBF-RRT, a motion planning algorithm that successfully generates collision-free trajectories and control strategies for a nonlinear system under both static and dynamic environments. In future work, we will further increase the richness of the mission specifications by adding temporal logic based constraints. Second, we will further improve the current algorithm with rewiring technique that is similar to RRT* and perform theoretical analysis. Third, we would like to extend the current algorithm to handle multi-agent motion planning.
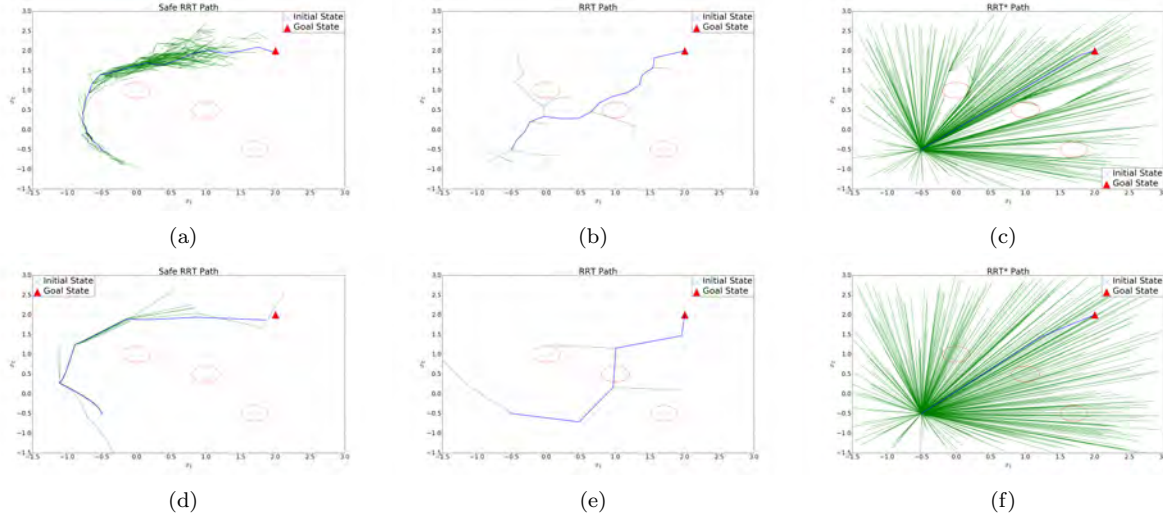
8. ACKNOWLEDGEMENTS

Figure 6: Comparison between CBF-RRT, RRT and RRT* with End-point collision check. (a) CBF-RRT with $t_h = 0.25s$. (b) RRT with $\delta d = 0.25m$. (c) RRT* with $\delta d = 0.25m$. (d) CBF-RRT with $t_h = 1.00s$. (e) RRT with $\delta d = 1.00m$. (f) RRT* with $\delta d = 1.00m$.

9. REFERENCES

[1] L. Kavraki, P. Svestka, and M. H. Overmars, *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*, vol. 1994. Unknown Publisher, 1994.

[2] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[3] 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.

[4] D. J. Webb and J. Van Den Berg, "Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics," in *2013 IEEE International Conference on Robotics and Automation*, pp. 5054–5061, IEEE, 2013.

[5] 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*, pp. 2537–2542, IEEE, 2012.

[6] 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*, pp. 2429–2436, IEEE, 2013.

[7] 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, pp. 5576–5581, IEEE, 2004.

[8] O. Adiyatov and H. A. Varol, "A novel rrt*-based algorithm for motion planning in dynamic environments," in *2017 IEEE International Conference on Mechatronics and Automation (ICMA)*, pp. 1416–1421, IEEE, 2017.

[9] D. Connell and H. M. La, "Dynamic path planning and replanning for mobile robots using rrt," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 1429–1434, IEEE, 2017.

[10] 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*, pp. 6271–6278, IEEE, 2014.

[11] 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)*, pp. 4542–4548, IEEE, 2015.

[12] U. Borrmann, L. Wang, A. D. Ames, and M. Egerstedt, "Control barrier certificates for safe swarm behavior," *IFAC-PapersOnLine*, vol. 48, no. 27, pp. 68–73, 2015.

[13] S. L. Herbert, M. Chen, S. Han, S. Bansal, J. F. Fisac, and C. J. Tomlin, "Fastrack: A modular framework for fast and guaranteed safe motion planning," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, p. 1517, IEEE, 2017.

[14] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer School on Machine Learning*, pp. 63–71, Springer, 2003.

[15] J. Neter, W. Wasserman, and M. H. Kutner, "Applied linear regression models," 1989.

[16] 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, IEEE, 2016.

[17] A. Isidori, *Nonlinear control systems*. Springer Science & Business Media, 2013.

[18] W. Xiao and C. Belta, "Control barrier functions for systems with high relative degree," *arXiv preprint arXiv:1903.04706*, 2019.

[19] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2018.