Control Barrier Functions for Safe Multi-Robot Navigation

Satvik Tajane, Harsh Akabari, Nicolas Drager

December 9, 2025

Outline

Introduction

- **Safety in robotics** varies by application domain
  - Manipulators, mobile platforms, drones, humanoids
- Safety-critical systems require formal, mathematically-grounded frameworks
- Control Barrier Functions (CBFs) provide systematic safety enforcement
- Here we present:
  - Theoretical foundations of CBFs
  - Practical implementation via quadratic programming (QP)
  - Application to robotic systems and human-robot collaboration

Control Barrier Functions: Definition

Consider nonlinear control-affine system:

$$\dot{x} = f(x) + g(x)u, \quad x \in \mathcal{D} \subset \mathbb{R}^n, \quad u \in U$$

Safe set $\mathcal{S} \subset \mathcal{D}$: forward invariant region.

Definition of CBF

Function $h : \mathcal{D} \to \mathbb{R}$ is a **CBF** if:

$$\sup_{u \in U} \left[\frac{\partial h}{\partial x} f(x) + \frac{\partial h}{\partial x} g(x) u \right] \geq -\alpha(h(x))$$

where α is extended class $\mathcal{K}$.

CBF-Based Safety Filter

- CBF acts as **safety filter** on nominal controller
- Minimally invasive: modifies control only when necessary
- Implemented via Quadratic Programming (QP):

$$u^* = \arg\min_{u \in U} \|u - u_{\text{des}}\|^2$$

subject to:

$$L_f h(x) + L_g h(x) u \geq -\alpha(h(x))$$

Our Project: Multi-Robot CBF

- CBF acts as **safety filter** on nominal controller

Preliminaries: Control Barrier Functions

Definition

A Control Barrier Function (CBF) $h(\mathbf{x})$ ensures safety by defining:

- **Safe set**: $\mathcal{C} = \{\mathbf{x} \mid h(\mathbf{x}) \geq 0\}$
- **Unsafe set**: $\{\mathbf{x} \mid h(\mathbf{x}) < 0\}$

Safety Guarantee

If the control input $\mathbf{u}$ satisfies:

$$\dot{h}(\mathbf{x}, \mathbf{u}) + \gamma h(\mathbf{x}) \geq 0 \tag{1}$$

where $\gamma > 0$, then $h(\mathbf{x}) \geq 0$ for all future time, ensuring the system remains safe.

Preliminaries: Control Barrier Functions

Definition

A Control Barrier Function (CBF) $h(\mathbf{x})$ ensures safety by defining:

- **Safe set**: $\mathcal{C} = \{\mathbf{x} \mid h(\mathbf{x}) \geq 0\}$
- **Unsafe set**: $\{\mathbf{x} \mid h(\mathbf{x}) < 0\}$

Safety Guarantee

If the control input $\mathbf{u}$ satisfies:

$$\dot{h}(\mathbf{x}, \mathbf{u}) + \gamma h(\mathbf{x}) \geq 0 \tag{2}$$

where $\gamma > 0$, then $h(\mathbf{x}) \geq 0$ for all future time, ensuring the system remains safe.

Robot Model: Differential Drive TurtleBot

State Variables:

- Position: (x, y)
- Heading: θ
- Robot radius: $R = 0.2$ m

Control Inputs:

- Linear velocity: v
- Angular velocity: ω

Physical Constraints:

- Velocity: $0 \leq v \leq 0.5$ m/s, $|\omega| \leq 1.5$ rad/s
- Acceleration: $|\dot{v}| \leq 0.5$ m/s^2, $|\dot{\omega}| \leq 2.0$ rad/s^2

Dynamics:

$$\dot{x} = v\cos(\theta) \tag{3}$$

$$\dot{y} = v\sin(\theta) \tag{4}$$

$$\dot{\theta} = \omega \tag{5}$$

Implementation: LIDAR-Based Obstacle Detection

360° LIDAR Simulation

- 360 rays at 1° angular resolution
- Each ray checks intersection with all obstacles
- Proper occlusion handling (ray stops at first hit)
- Returns **surface points**, not obstacle centers

Obstacle Types:

- **Circles**: Solve quadratic equation for ray-circle intersection
- **Rectangles**: Check ray-line segment intersection for four edges
- **Other robots**: Detected as circular obstacles

Clustering (Optional)

Can group consecutive LIDAR detections (within 0.3m) into clusters to identify separate objects and distinguish between static obstacles and other robots.

CBF Formulation

For closest detected surface point $\mathbf{p}_{\text{obs}}$:

Barrier Function

$$h(\mathbf{x}) = \|\mathbf{p}_{\text{robot}} - \mathbf{p}_{\text{obs}}\|^2 - (R_{\text{robot}} + \text{buffer})^2 \tag{6}$$

Safety buffers:

- Static obstacles: buffer $= 0.03$ m
- Other robots (normal): buffer $= 0.0$ m
- Other robots (low priority): buffer $= -0.05$ m

Time Derivative

$$\dot{h} = 2(\mathbf{p}_{\text{robot}} - \mathbf{p}_{\text{obs}})^T \cdot \dot{\mathbf{p}}_{\text{robot}} = 2(dx \cdot v\cos\theta + dy \cdot v\sin\theta) \tag{7}$$

where $(dx, dy) = \mathbf{p}_{\text{robot}} - \mathbf{p}_{\text{obs}}$

CBF constraint: $\dot{h} + \gamma h \geq 0$ with $\gamma = 2.0$

Quadratic Programming Controller

At each timestep, solve:

$$\begin{aligned} \min_{\mathbf{u}} \quad & \|\mathbf{u} - \mathbf{u}_{\text{des}}\|^2 \\ \text{s.t.} \quad & \dot{h} + \gamma h \geq 0 \\ & v_{\min} \leq v \leq v_{\max} \\ & \omega_{\min} \leq \omega \leq \omega_{\max} \end{aligned} \tag{8}$$

Acceleration bounds:

$$v_{\min} = \max(0, v_{\text{prev}} - a_{\max} \cdot dt) \tag{9}$$

$$v_{\max} = \min(v_{\max}, v_{\text{prev}} + a_{\max} \cdot dt) \tag{10}$$

$$\omega_{\min} = \max(-\omega_{\max}, \omega_{\text{prev}} - \alpha_{\max} \cdot dt) \tag{11}$$

$$\omega_{\max} = \min(\omega_{\max}, \omega_{\text{prev}} + \alpha_{\max} \cdot dt) \tag{12}$$

Nominal Controller

Goal: Drive robot to target position $\mathbf{g} = (g_x, g_y)$

Proportional Control

Linear velocity:

$$v_{\text{des}} = \min(v_{\max}, 0.5 \cdot \|\mathbf{p}_{\text{robot}} - \mathbf{g}\|) \tag{13}$$

Angular velocity:

$$\omega_{\text{des}} = 3.0 \cdot \text{atan2}(\sin(\theta_{\text{goal}} - \theta), \cos(\theta_{\text{goal}} - \theta)) \tag{14}$$

where $\theta_{\text{goal}} = \text{atan2}(g_y - y, g_x - x)$

Behavior:

- Slows down as approaching goal (prevents overshoot)
- Turns proportionally to heading error
- Provides desired control $\mathbf{u}_{\text{des}}$ for QP

QP Objective Function

Standard case (not heading directly toward obstacle):

$$J(\mathbf{u}) = (v - v_{\text{des}})^2 + 0.5(\omega - \omega_{\text{des}})^2 \tag{15}$$

Obstacle avoidance mode (when $|\text{angle\_diff}| < 60$ and $h < 1.0$):

$$J(\mathbf{u}) = \begin{cases} 8.0v^2 + 0.5(\omega - \omega_{\text{target}})^2 & \text{if } h < 0.3 \\ 4.0v^2 + 0.5(\omega - \omega_{\text{target}})^2 & \text{if } 0.3 \leq h < 0.6 \\ 2.0(v - v_{\text{des}})^2 + 0.5(\omega - \omega_{\text{target}})^2 & \text{if } h \geq 0.6 \end{cases} \tag{16}$$

where ω_{target} encourages turning away from obstacle.

Multi-Robot Coordination: Priority System

Challenge: Symmetric avoidance can cause deadlocks
Solution: Priority based on proximity to static obstacles

Priority Determination

For each robot, compute h_{static} = distance to nearest static obstacle
Robot with smaller h_{static} gets priority (closer to static obstacle = more constrained)

Differential Safety Buffers

- **Priority robot**: Treats other robot with buffer $= -0.05$ m
  - Can approach closer, effectively "pushing"
- **Non-priority robot**: Treats other robot with buffer $= 0.0$ m
  - Normal avoidance, yields to priority robot

Result: Asymmetric interaction breaks deadlocks and creates natural yielding behavior

Emergency Recovery: QP Failure Handling

When does QP fail?

- Constraint becomes infeasible (no valid velocities satisfy safety)
- Typically when $h \approx 0$ and robot has forward momentum

Smart Recovery Strategy

1. Test both turn directions: compute $\dot{h}$ for $\omega = \pm 0.8\omega_{\max}$
2. Choose direction with larger $\dot{h}$ (increases safety margin faster)
3. Check if small forward velocity ($v = 0.05$ m/s) is safe
4. If safe: turn while moving forward
5. Else: pure rotation ($v = 0$)

Advantage: Escapes dangerous situations faster than pure rotation

Discrete-Time Implementation

Challenge: Continuous-time CBF guarantee doesn't hold perfectly in discrete time

Integration Error

With Euler integration and timestep $dt = 0.05$ s:

- Robot can overshoot by $\approx 0.5 a_{\max} dt^2$
- This can cause h to become negative despite satisfying constraint

Compensation

Add small margin to constraint:

$$\dot{h} + \gamma(h - \varepsilon) \geq 0 \tag{17}$$

where $\varepsilon = 0.5 a_{\max} dt^2 \approx 0.000625$ m^2

Result: Accounts for discrete-time effects while minimally affecting behavior

Implementation Summary

Algorithm Pipeline:

1. **Sense**: LIDAR scan $\rightarrow$ detect obstacle surfaces
2. **Cluster**: Group detections $\rightarrow$ identify objects
3. **Compute**: Calculate h for closest point
4. **Optimize**: Solve QP for safe control
5. **Execute**: Apply velocity commands
6. **Fallback**: If QP fails, use smart recovery

Key Features:

- Surface-based detection (not center-based)
- Handles multiple obstacle types (circles, rectangles)
- Dynamic obstacle avoidance (other robots)
- Priority-based coordination
- Robust emergency recovery