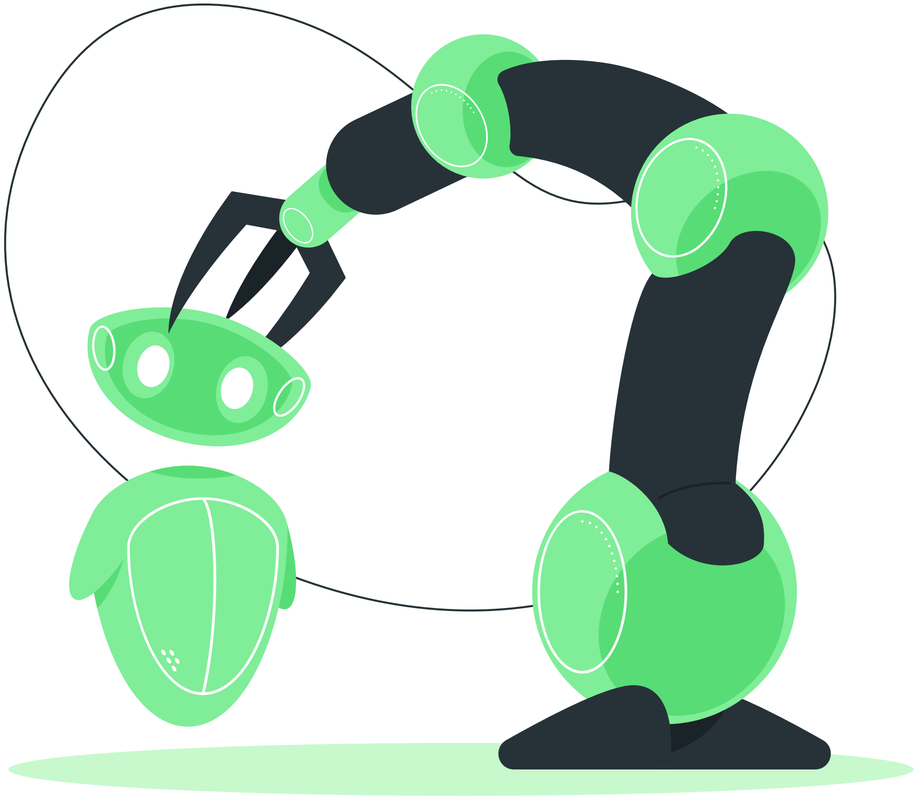
EECS C106B Safe Control

Project 4 Introduction



Tasks

1

Feedback Linearization

Tracking controller with dynamic extension

2

Deadlock CBF-QP

Safety-critical controller for multiple agents

3

Vision CBF-QP

Vision-based safety critical controller

4

Hardware Implementation

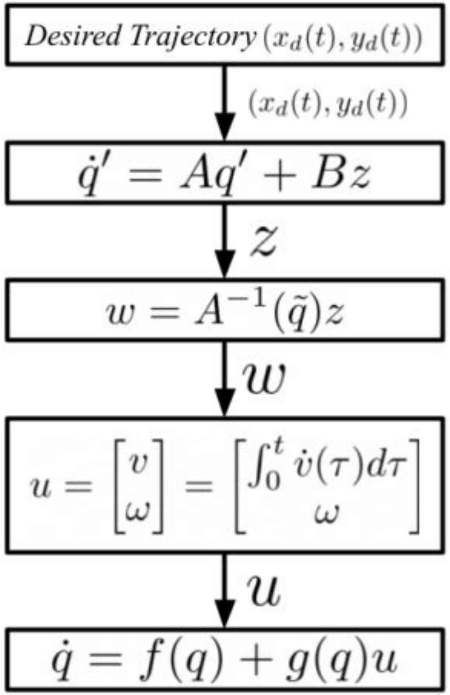
Deploy a braking CBF-QP on turtlebots

Feedback Linearization

- How can we transform a nonlinear system into a linear system using *feedback control?*
- Once we’ve made the system linear with feedback, how can we design tracking controllers?

Dynamic Extension

- For the turtlebot, we will have to use *dynamic extension* for feedback linearization
- Augment the state and input vector of the system to linearize
- Implementing in code: use a numerical integral

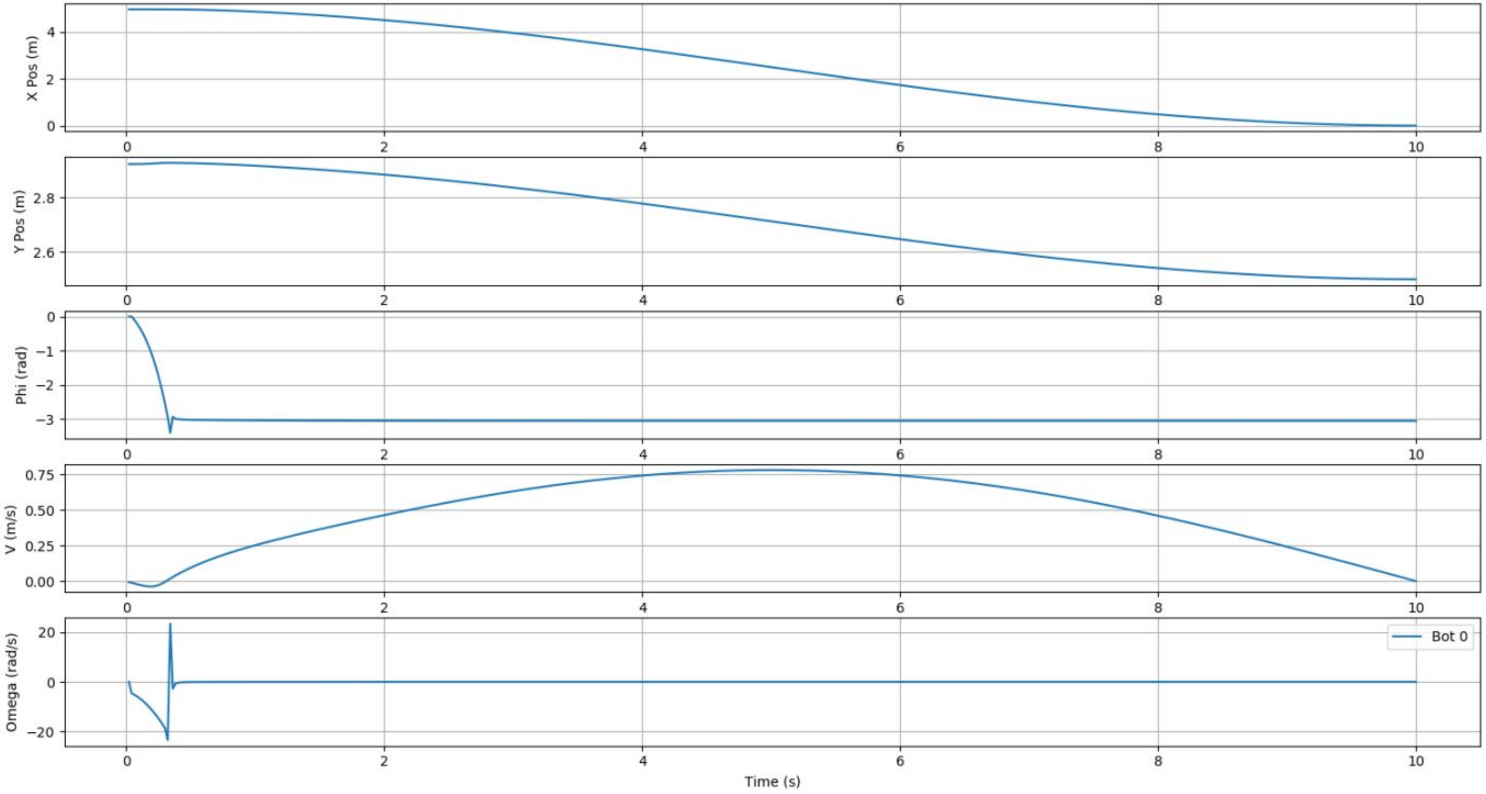


Linear Tracking Control

- How can we design an effective linear tracking controller for the system?
    - We'd like to track a desired trajectory $q_d(t)$ (only care about tracking x(t), y(t) - phi(t) may be anything
    - We have access to the first and second time derivatives of $q_d(t)$
- *Hint: how can we choose an input z to get stable second order error dynamics, where e is the tracking error?*

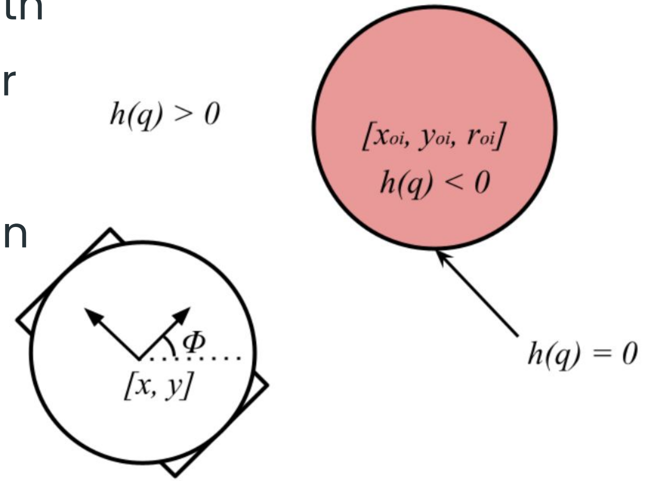
$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} z \rightarrow \ddot{e} + c_1 \dot{e} + c_2 e = 0$$

Response Plots



Control Barrier Functions

- Imagine we have a system of many turtlebots
  - May encode the safety of each turtlebot with respect to the others using a control barrier function h(q)
- We'll consider a control barrier function between the turtlebot we wish to control (ego turtlebot) and an obstacle turtlebot
- What should h(q) be?



Deadlock CBF-QP

- Let's use our control barrier function to enable a system of multiple turtlebots to safely track trajectories
- Challenge: if we apply a CBF-QP directly to a feedback linearizing input, turtlebots will get stuck in face-off scenarios called *deadlocks*
    - How can we resolve these deadlocks?

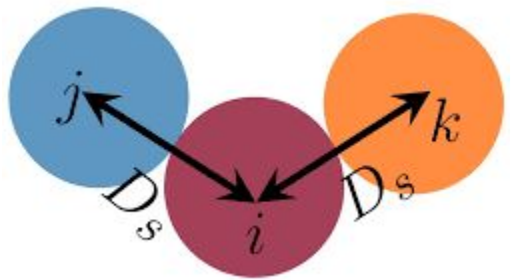


Image Source: Deadlock Analysis and Resolution for Multi-Robot Systems

Deadlock CBF-QP

- We'll apply a CBF-QP in the *innermost linear layer* of feedback linearization - then pass the safe z to the outer layers
- To encourage the turtlebot to steer around obstacles, we can weight the steering term in the input differently with a matrix Q
- Apply a barrier function constraint for each turtlebot, with derivatives taken along the trajectories of the **linear system**

$$z_{safe} = \underset{u \in \mathbb{R}^2}{\arg\min}\ (z - k(z))^T Q (z - k(z))$$

$$s.t.\ \ddot{h}_i(q, z) + k_1 \dot{h}_i(q) + k_2 h_i \geq 0,\ i = 2, 3, ..., n$$

Deadlock CBF-QP Constraint

- Why do we need a second derivative in the constraint?
  - We take the derivatives of the barrier function h(q) along the trajectories of the linear system

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} z$$

- The input z will not appear until we take the second derivative of our barrier function along the trajectories of the linear system
- Challenge: how can we select the weights in the barrier constraint?
  - Try solving the ODE for h(t) - for which k1, k2 is h > 0?

$$\ddot{h}_i(q, z) + k_1 \dot{h}_i(q) + k_2 h_i \geq 0, \ i = 2, 3, ..., n$$

Deadlock CBF-QP Summary

- Let's summarize the structure of the CBF-QP:

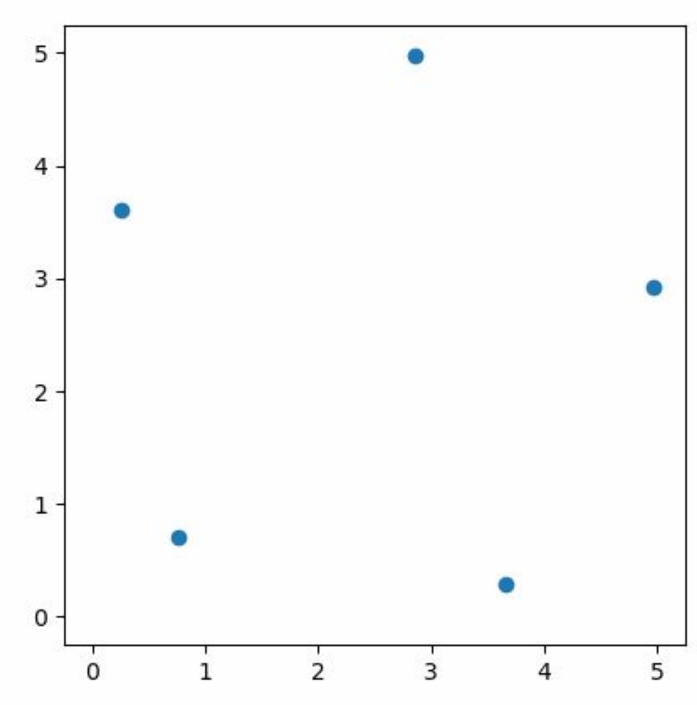
Use your tracking controller to find the tracking z input

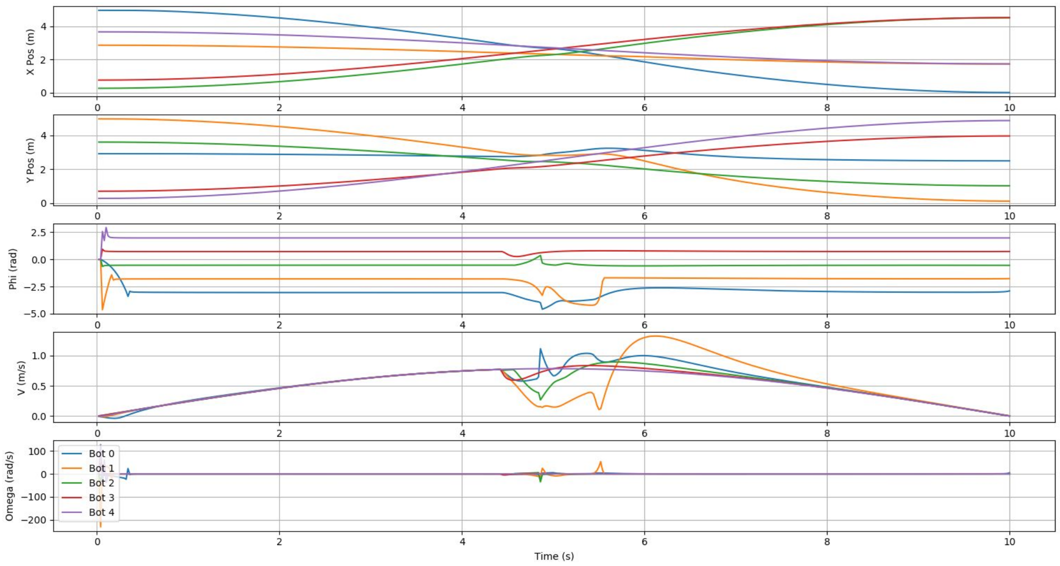
Apply a CBF-QP to the tracking z input to get a safe z input

Convert z input to a w input using z = A(q) w

Convert w input to a u input via integration and sent u to the system

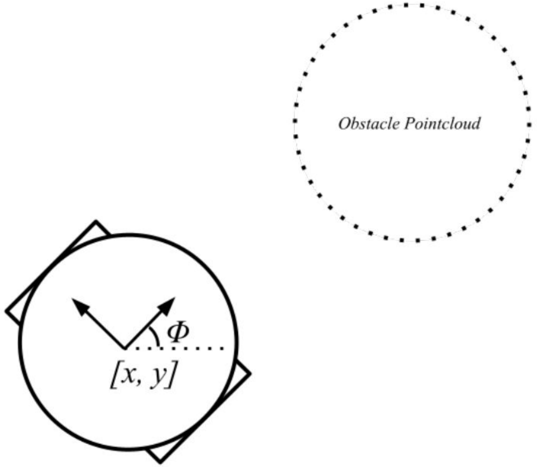
Deadlock CBF-QP Demo





Vision-Based CBF-QP

- The turtlebots interact with their environment using LIDAR sensors, which return a pointcloud in the turtlebot frame
- How can we come up with a single barrier function h(q) that encodes the safety of the system based on the pointcloud?
  - Closest point, clustering, and many more methods!
  - Fine if you get some deadlocks



Braking CBF-QP

- We'll implement a simplified CBF-QP on hardware
- We won't incorporate the full relative degree 2 constraint for simplicity - we'll just require that our bot brakes for obstacles

$$u^*_{safe} = \arg\min_{u \in U} ||u - k(q)||^2$$
$$s.t.\ \dot{h}(q) \geq -\gamma h(q)$$

$$\dot{q} = \begin{bmatrix} \cos\phi & 0 \\ \sin\phi & 0 \\ 0 & 1 \end{bmatrix} u$$

- Now, we'll apply this CBF-QP directly over the feedback linearizing input, k(q) to get the input u_{safe}
- Now, the derivative of h should be taken along the *nonlinear* turtlebot dynamics, rather than the linear system

Braking CBF-QP

- The barrier function h(q) should be exactly the same as your vision-based barrier function from simulation
- Implement this controller on hardware
  - Stand in the way of the turtlebot as it moves - the braking CBF should slow the turtlebot and prevent it from crashing
  - If you move towards the turtlebot, it should evade you

Braking CBF-QP Demo

Using the TurtleBots

- Remember to switch them OFF to charge
  - If a TurtleBot is not sufficiently charged for the next group you may lose points
- Carry them by the base
- Watch where they're going!
  - Be ready to press Ctrl + C
- Refer to the Robot Usage Guide for setup