Last Time

Lecture 2

EAIS SP'25

Andrea Bajcsy

- ☐ sequential decision-making
- ☐ Dynamic programming

This Time:

- ☐ what makes safe decision-making "hard"?
- ☐ safety filters

RECAP : we derived the Bellman Equation for sum-of-cost problem:

Decision - Problem :

$$\min_{u_{0:T}} \sum_{t=0}^{T-1} L(x_t, u_t) + \ell(x_T)$$

$$\text{s.t. } x_{t+1} = f(x_t, u_t)$$

$\Rightarrow$

Bellman Equation

$$V_t(x_t) = \min_{u_t} \left[L(x_t, u_t) + V_{t+1}(x_{t+1}) \right],$$

$$V_T(x_T) = \ell(x_T)$$

$x_{t+1} = f(x_t, u_t)$

- The beauty is this lets us decompose decision-making problems into smaller subproblems and solve recursively, pointwise optim. over ctrl.

Deriving Optimal Control from Value Function :

Ultimately, we want u^*, the optimal control!

By definition of the value function, the optimal control is given by the value function minimizer:

$$u_t^*(x_t) := \arg\min_{u_t} \{ L(x_t, u_t) + V_{t+1}(x_{t+1}) \}$$

Taking a step back, this is the full decision-making prob:

$$\underset{u_{0:T}}{\text{maximize}} \quad J(x_0, u_{0:T}) \quad \leftarrow \text{objective}$$

$$\text{s.t.} \quad x_{t+1} = f(x_t, u_t) \; \forall t \quad \leftarrow \text{dynamics}$$

$$u_t \in \mathcal{U} \quad \leftarrow \text{ctrl bounds}$$

$$x_t \notin \mathcal{F} \subset \mathcal{X} \quad \leftarrow \text{state constraints}$$

This is what we will focus on in the first part of the class $\rightarrow$

also called **failure set**

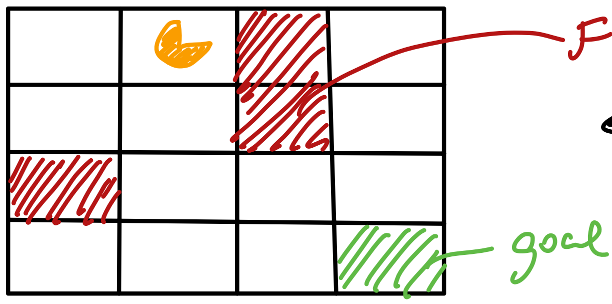
think: examples of failure sets $\mathcal{F} \subset \mathcal{X}$ that might matter in:

- robot manipulator cleaning cluttered countertop
- drone flying through city

Before we go on, let's discuss why is ensuring that a robot makes decisions s.t. $x_t \notin \mathcal{F}$ hard?

I mean, if we know the failure set, isn't this enough?

For example, in Intro AI you'll see gridworld

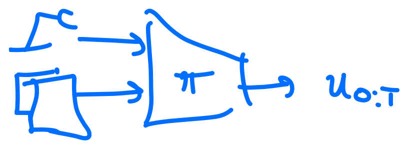


← if going to hit $\mathcal{F}$: don't;

BUT, for EAI systems from this class, ensuring $x_t \notin \mathcal{F}$ is hard.

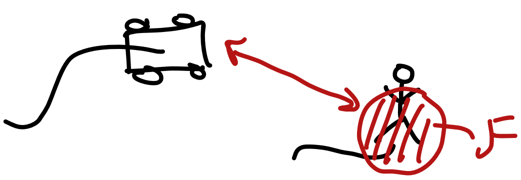
Reason #1. In practice, our decision-making "stack" or "policy" can be arbitrarily complex, or opaque.

e.g. E2E Diffusion Policy
VLM - web agent.
...



Reason #2: other (strategic) agents

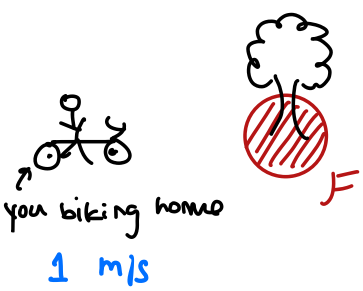
↳ homicidal chauffer problem (1971)

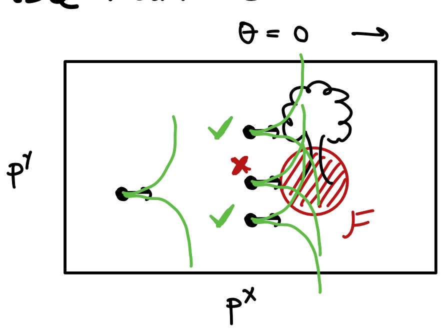


- Attacker or adversary
- environment (ice, wind, ...)

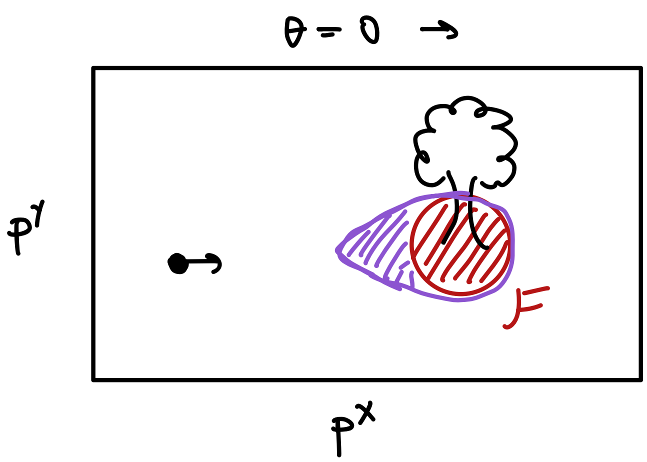
Reason #3: Uncertainty! Even though we represent our system via a model, it will never perfectly reflect reality.

Reason #4: Inevitable Failure





Your brakes don't work! Can only turn but not slow down.



These purple states are the inevitable failure states

↓ also called $\boxed{\text{unsafe set}}$

How do we tackle Problems #1–4 rigorously but also practically?

$\boxed{\text{IDEA : SAFETY FILTERS}}$

Safety Filtering

$\boxed{\text{Goal}}$ A "wrapper" we can put around ANY "base" policy or decision-making system to:

(1) monitor if the system is @ "risk" of violating $x \notin \mathcal{F}$

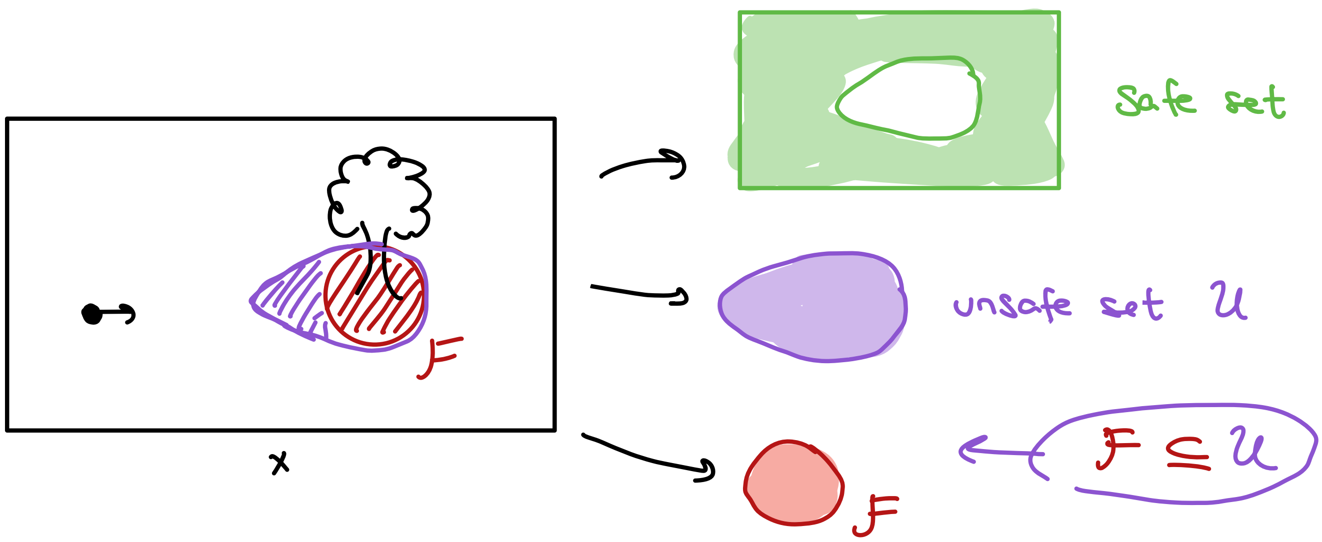
(2) adjust the robot's base policy @ runtime to prevent future failure. ($x_t \notin \mathcal{F} \;\; \forall t \in \{0, 1, \dots\}$)

The idea is that we continue applying our nominal strategy for decision-making (e.g. VLM, Diff., MPC, ...) until safety is at risk; otherwise, apply a safety controller which steers away!

ex. SIMPLEST STRATEGY!

$$u^*(x) = \begin{cases} \pi^{nom}(x) & \text{if system is safe} \\ \pi^{safe}(x) & \text{if safety @ risk} \end{cases}$$

How do we know safety is @ risk, and the safe ctrl.?

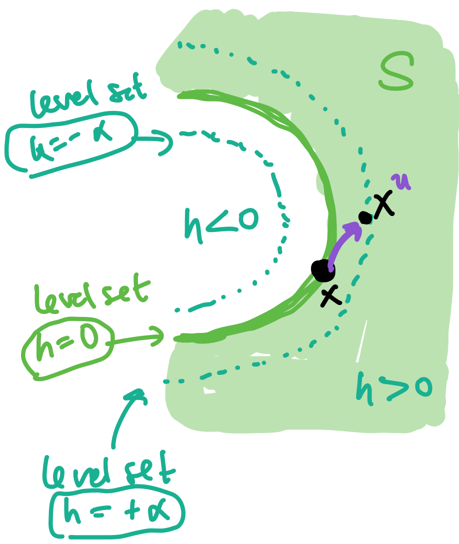


! **ASSUME** we have a known safe set $S \subset \mathcal{X}$ that:

(continuously differentiable) f^n $h: \mathcal{X} \to \mathbb{R}$

1. $S \cap \mathcal{F} = \emptyset$
2. $x \in S \iff h(x) \geq 0$ } encode the set via this function.
   $x \in \partial S \iff h(x) = 0$

"boundary of S"



If we have access to such a set S and $h(x)$ function, then any policy $u := \pi(x)$ such that

you are @ state x apply action u you get to state $x^u(t+\delta)$ in some small t-step δ.

$$h(x^u(t+\delta)) \geq 0$$

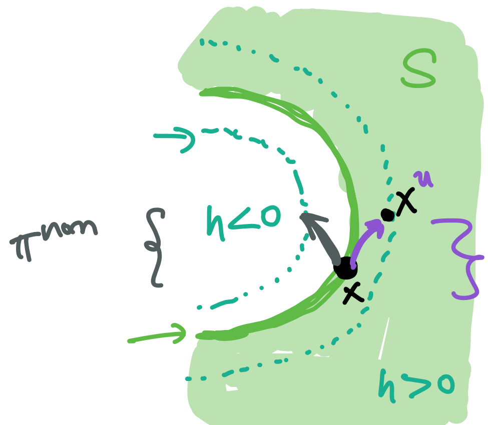
then this policy can prevent the system from leaving S (i.e. $\pi(x) \equiv \pi^{safe}$)

In this context, you may have heard of $h(x)$ function called a control barrier function (CBF) for S.

LEAST RESTRICTIVE SAFETY FILTER

$$u^*(x) = \begin{cases} \pi^{nom}(x) & \text{if } h(x) > 0 \quad (\text{i.e. } x \in S) \\ \pi^{safe}(x) & \text{if } h(x) \approx 0 \quad (\text{i.e. } x \in \partial S) \end{cases}$$

[Q] Problem: can lead to switch (aggressively) btwn. the nominal policy & safety policy:



safety π

a lot of tension btwn maximally safe control and the nominal control

A key insight of CBF is an alternative safety filtering law which looks for similar control to the nominal one that is also safe.

$$u^*(x) = \arg\min_u \|u - \pi^{nom}(x)\|_2^2 \quad \text{s.t. } \boxed{\underline{u \text{ is safe}}}$$

"stay close to π^{nom} while being safe"

[Q] What is this? What is set of safe ctrls?

[A] We know that

$$u \text{ is safe} \iff h(x^u(t+\delta)) \geq 0$$

from before.

$h(x^u(t+\delta)) \geq 0$

Taylor series expansion @ t

$$h(x^u(t+\delta)) \approx h(x(t)) + (t+\delta-t)\frac{dh(x(t))}{dt} \geq 0$$

$$= h(x(t)) + \delta\left[\frac{\partial h}{\partial x} \cdot \boxed{\frac{\partial x}{\partial t}}\right] \geq 0$$

chain rule

$f(x,u)$ this is dyns!

$$= h(x(t)) + \delta\left[\boxed{\frac{\partial h}{\partial x}} \cdot f(x,u)\right] \geq 0$$

tells you how the value changes in space (r.u. getting more positive or neg.?)

Now, we have the following safety filter:

$$u^*(x) = \arg\min_{u} \|u - \pi^{nom}(x)\|_2^2 \quad (*)$$

$$\text{s.t. } h(x(t)) + \delta\left[\frac{\partial h}{\partial x} \cdot f(x,u)\right] \geq 0$$

Some systems make solving fast! (i.e. a quadratic program)

<u>CONTROL AFFINE:</u>

$$f(x,u) := \boxed{f_1(x) + f_2(x)u}$$

This makes our optimization problem $(*)$ a quadratic program!

→ objective is quadratic in u

→ constraint is linear in u

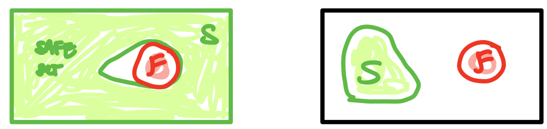
↓ Solve fast online!

$$h(x(t)) + \delta\left[\frac{\partial h}{\partial x} f_1(x) + \frac{\partial h}{\partial x} f_2(x)\,u\right] \geq 0$$

BUT, let's talk about the caviats & elephant in the room

⚠ Obtaining a <u>VALID</u> and not <u>overly conservative</u> safe set S is really challenging for general systems

and $h(\cdot)$ function!



VALID	✓	✓
MAXIMAL	✓	✗

⚠ The CBF framework we have seen today <u>doesn't</u> handle uncertainty/disturbances robustly, and we also haven't talked about control input constraints.

Disturbances typically break the assurance of S we have seen so far!

We will tackle these challenges head-on during the next lecture:

$$\boxed{\text{synthesizing (i.e computing) safe sets \& safety filters}}$$

↳ we will talk about a general, computational framework for getting S and π^{safe} and $h(\cdot)$ that

1. is guaranteed to be <u>VALID</u> & <u>MAXIMAL</u>
2. naturally handles disturbances robustly
3. is compatible w/ modern comp. tools $\left\{ \begin{array}{l} \text{RL} \\ \text{SSL} \end{array} \right.$!