Last Time

Lecture 2

EAIS SP'25

Andrea Bajcsy

- ☐ sequential decision-making
- ☐ Dynamic programming

This Time:

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

ANNOUNCEMENTS:

- HW #1 Released! It's on today's topic — safety filters :)
- Ken's OH's:

  Tuesdays, 1-2 pm, NSH 1511.

Attendance Question of the Day:

Why is dynamic programming "helpful" compared to forward simulation?

So far, we did a recap of mathematical modeling frameworks that let us think about decision-making.

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

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

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

This is what we will focus on in the first part of the class $\rightarrow$ $x_t \notin \mathcal{F}$ $\quad \leftarrow$ state constraints

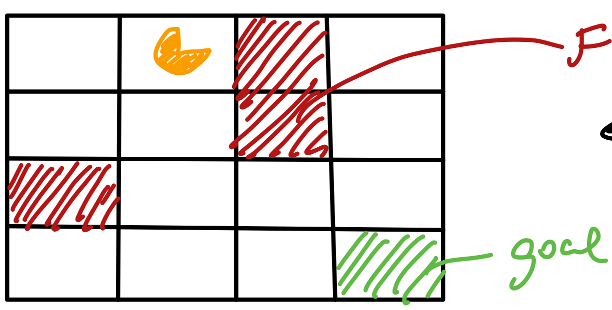
Give me examples of failure sets $\mathcal{F} \subset \mathcal{X}$ that might matter in:

- robot manipulator cleaning cluttered counter top
- drone flying through city

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

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

For example, in Intro AI you'll see gridworld

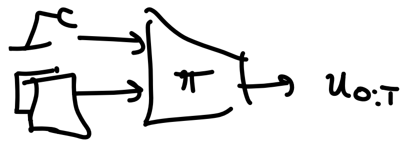


$\leftarrow$ if going to hit $\mathcal{F}$: don't;

BUT, for EAI systems from this class, ensuring $x_t \notin \mathcal{F}$ is <u>harder</u>.

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

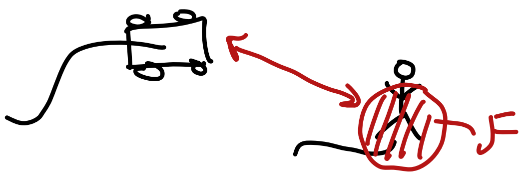
e.g. E2E Diffusion Policy



VLM - web agent.

Reason #2. other (strategic) agents

homicidal chauffer problem (1971)



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

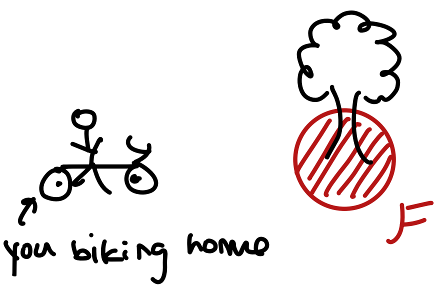
Reason #3: Uncertainty

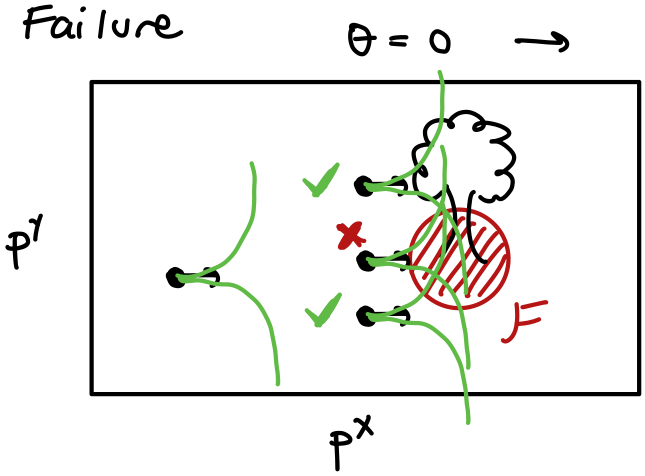
Even though we represent our system via a mathematical model, a model will never perfectly reflect reality.

Statistician George Box:

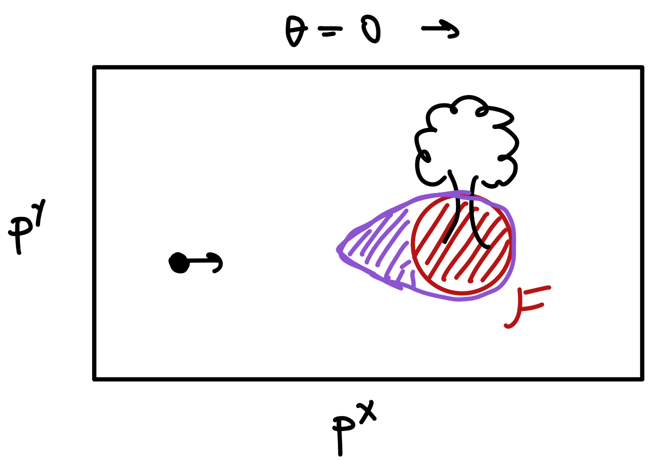
"All models are wrong, but some are useful"

Reason #4: Inevitable Failure





Your brakes don't work! can only turn.

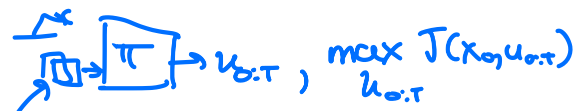


These purple states are the inevitable failure states
↓ also called $\boxed{\text{unsafe set}}$

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 $\boxed{\text{ANY "base" policy /}}$ decision-making system

(1) monitor if the system is @ "risk"

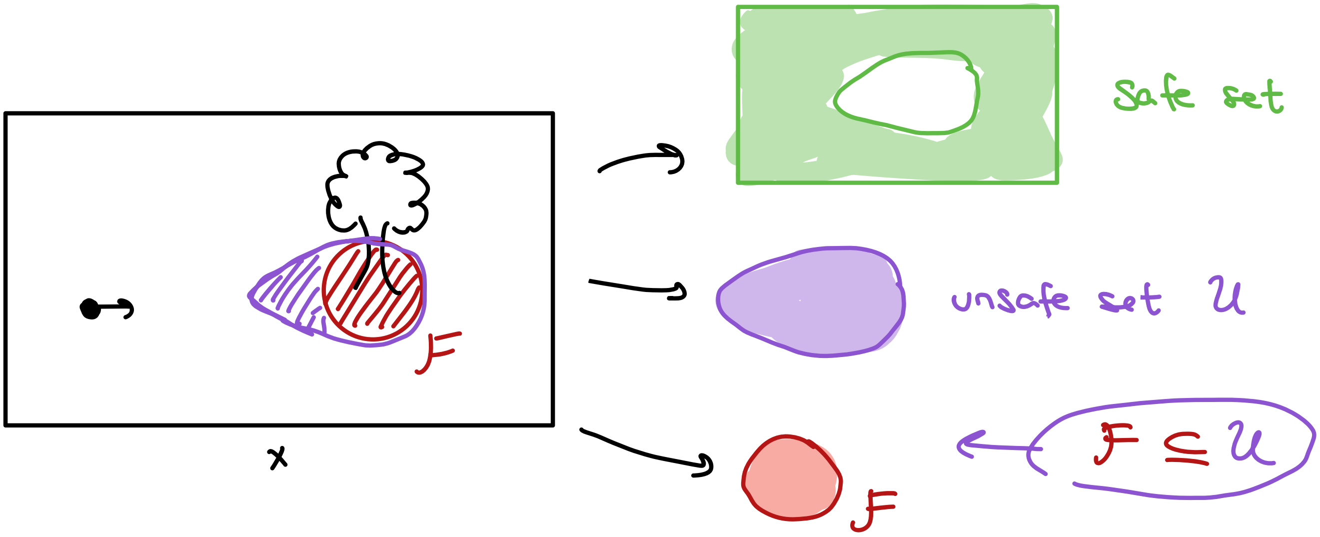
(2) adjust the robot's base policy @ runtime to prevent future failure. ($x_t \notin \mathcal{F} \ \forall t \in \{0, 1, \dots\}$)

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.

ex. SIMPLEST STRATEGY!

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

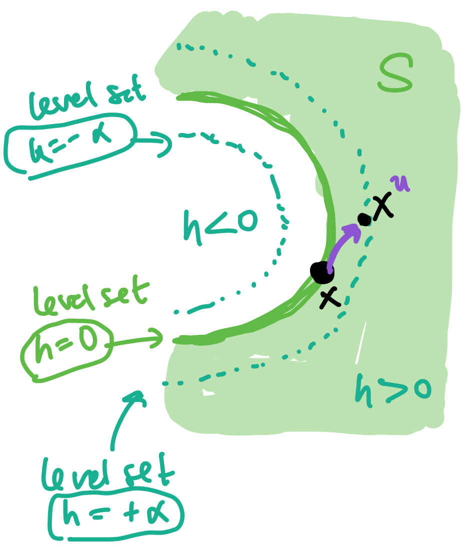
$\boxed{\text{Q}}$ How do we know safety is @ risk, and the safe ctrl.?



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

1. $S \cap F = \phi$ — continuously differentiable f^n $h: \mathcal{X} \to \mathbb{R}$
2. $x \in S \iff h(x) \geq 0$
   $x \in \partial S \iff h(x) = 0$ } encode the set via this func.

"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.

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

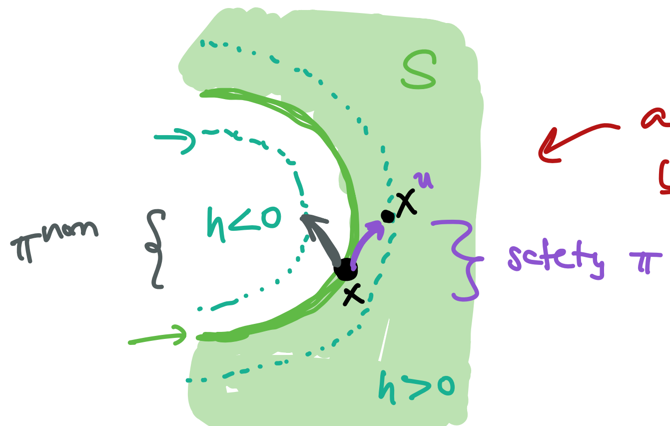
then this policy can prevent the system from leaving S (i.e. $\pi(x) \equiv \pi^{safe}$)

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

LEAST RESTRICTIVE SAFETY FILTER

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

$\boxed{Q}$ Problem: can lead to switch (aggressively) btwn. the nominal policy & safety policy



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"

$\boxed{Q}$ What is this? What is set of safe ctrls?

$\boxed{A}$ 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

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

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

$$\frac{\partial x}{\partial t} \to f(x,u)$$

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

Now, we have the following safety filter:

$$\boxed{\begin{aligned} u^*(x) = \arg\min_{u} \ & \| u - \pi^{nom}(x)\|_2^2 \\ \text{s.t. } & h(x(t)) + \delta\left[\frac{\partial h}{\partial x}\cdot f(x,u)\right] \geq 0 \end{aligned}} \quad (*)$$

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!

↓ solve fast online!

→ objective is <u>quadratic</u> in u

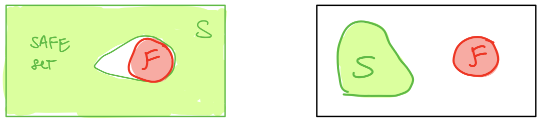
→ constraint is <u>linear</u> in u

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

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

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

and $h(\cdot)$ function!



VALID ✓ ✓

MAXIMAL ✓ ✗

⚠ The CBF framework we have seen today doesn't 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:

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 VALID & MAXIMAL
2. naturally handles disturbances robustly
3. is compatible w/ modern comp. tools { RL, SSL !