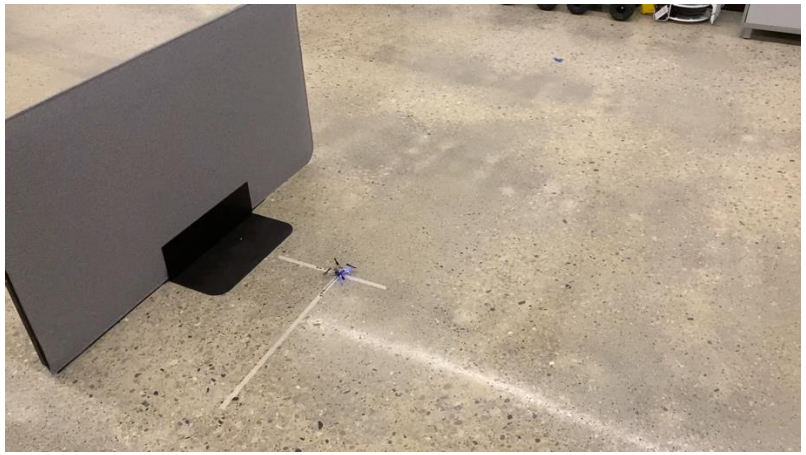
Architectures for Safe Autonomy:
Provable Guarantees Across Control, Planning, and Perception

Devansh R Agrawal

PhD Defense

March 17, 2025



Architectures for Safe Autonomy:
Provable Guarantees Across Control, Planning, and Perception

Devansh Agrawal

University of Michigan, Ann Arbor

March 17, 2025

two main questions

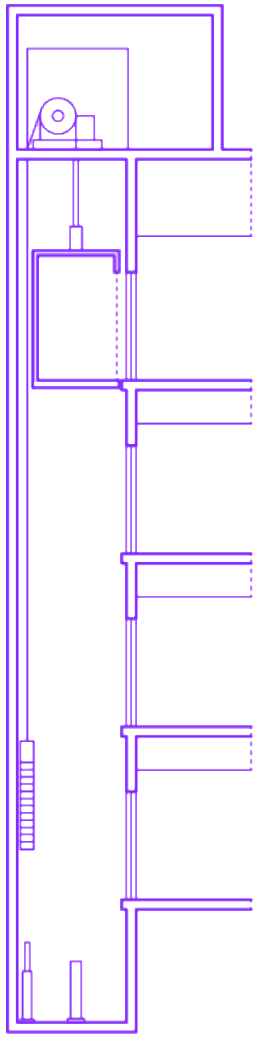
How can we **ensure** safety constraints are not violated during operation?

What are the limits of the information gathering ability of robots?

Part 1: `gatekeeper`

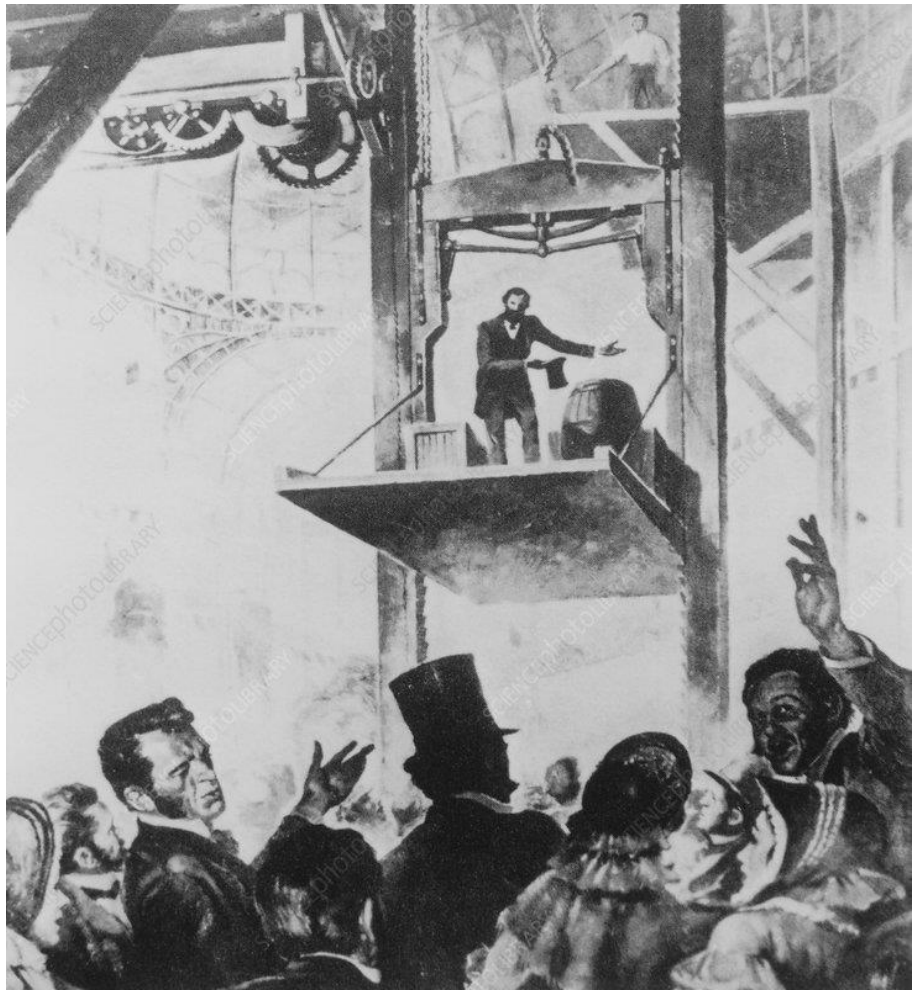
Part 2: Clarity and Perceivability

why do we want safety?



Why do you choose to use the lift?

why do we want safety?



Elisha Otis, 1854

Why do you choose to use the lift?

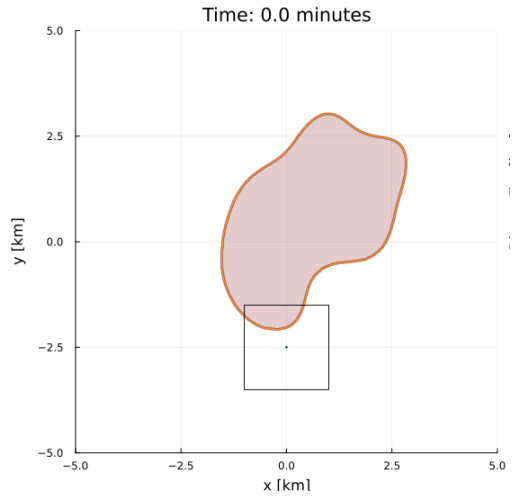
Otis' lift had a built-in safety mechanism:
when the cable is severed, a leaf-spring is released, and it grabs onto the lift shaft.

How does this extend to modern autonomous systems?

what are safety guarantees?

Ex. 1: Subterranean Navigation

Navigate through the caves to get a goal location.
Could be teleoperated or fully autonomous.

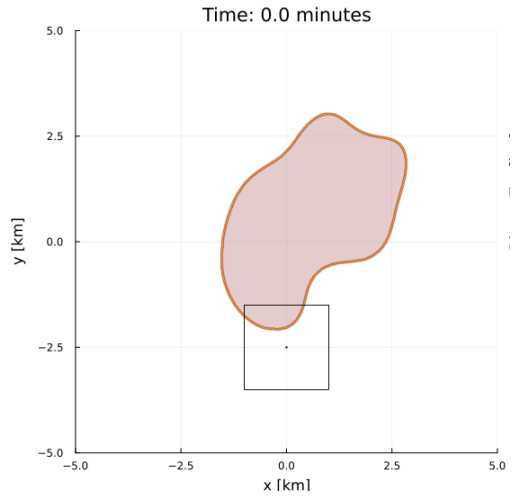


Ex. 2: Firewatch Mission

Fly along the fire perimeter.
The helicopter's GPS trace will used to allocate firefighting resources later.

what are safety guarantees?

Ex. 1: Subterranean Navigation

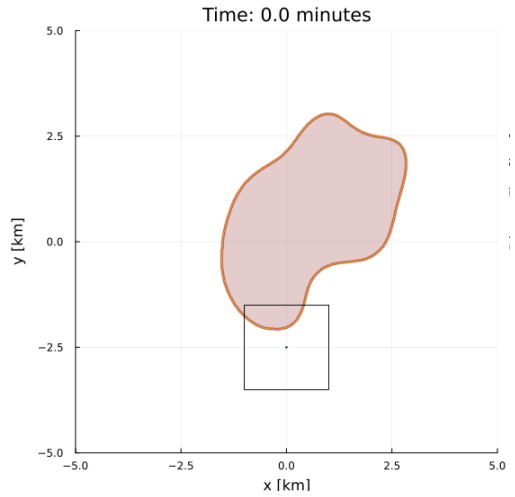


Ex. 2: Firewatch Mission

We want to ensure safety,	obstacle avoidance	helicopter doesn't enter the fire
subject to dynamics and input constraints,	quadrotor dynamics max speed of each motor	helicopter dynamics max engine thrust and max roll angle
while completing the mission,	navigate to a target location	map the boundary of the fires
and being robust to disturbances and uncertainties	wind, ground-effect, GPS-denied…	unknown fire spread rate, …

what are safety guarantees?

Ex. 1: Subterranean Navigation

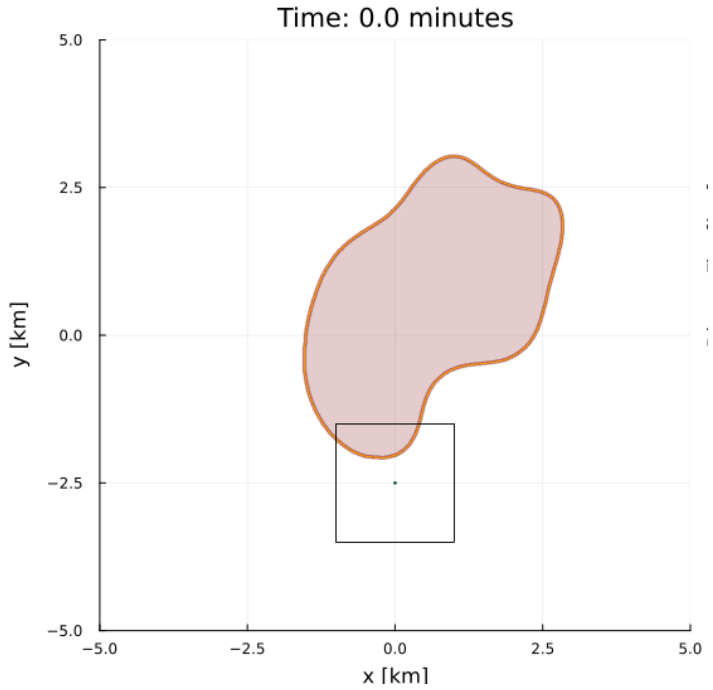


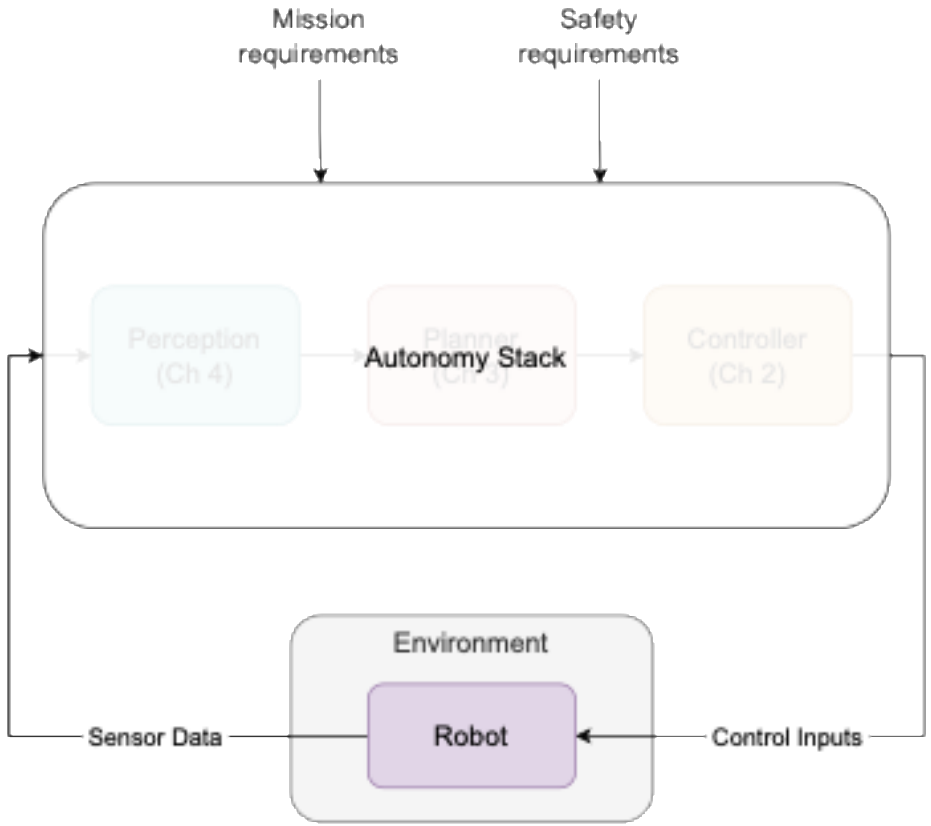
Ex. 2: Firewatch Mission

We want to ensure safety,	obstacle avoidance	helicopter doesn't enter the fire
subject to dynamics and input constraints,	quadrotor dynamics max speed of each motor	helicopter dynamics max engine thrust and max roll angle
while completing the mission,	navigate to a target location	map the boundary of the fires
and being robust to disturbances and uncertainties	wind, ground-effect, GPS-denied…	unknown fire spread rate, …

We want a **mathematical guarantee** that the robot will not become unsafe/behave unexpectedly.

how do we build (safe) autonomy?

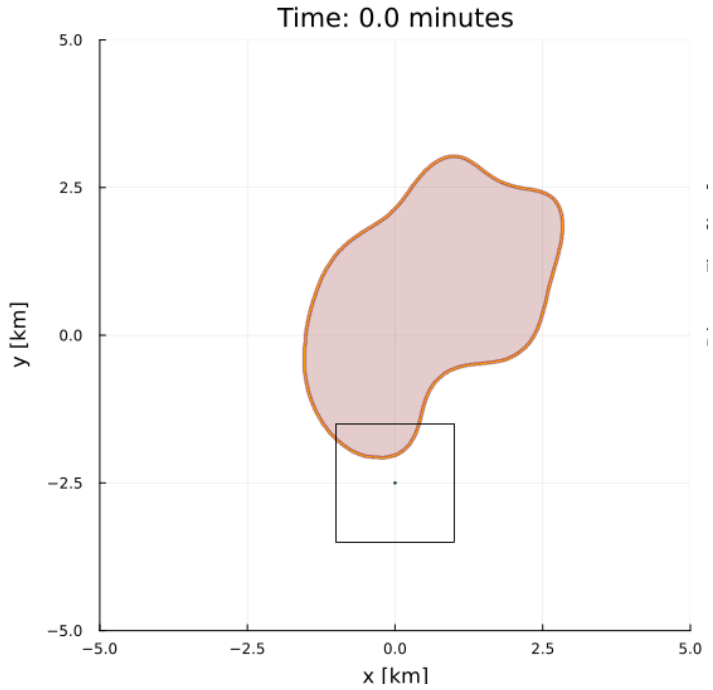




The “autonomy stack” consists of many modules.

The only interface to the robot is the control input and the sensor output.

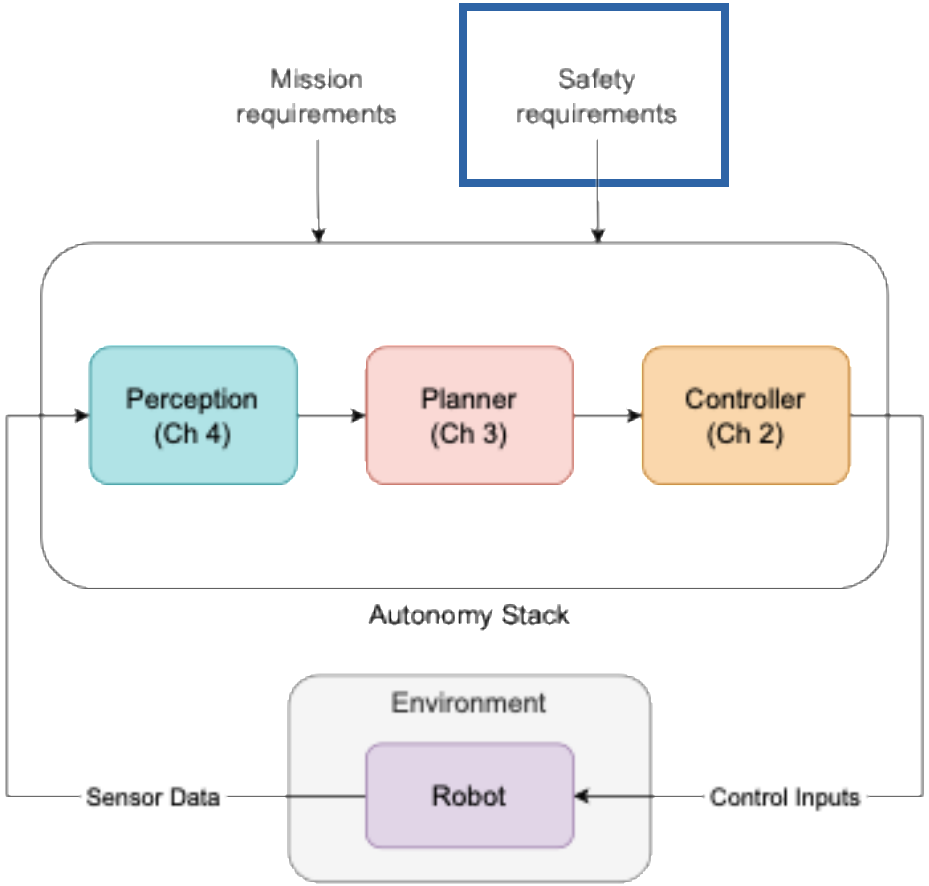
how do we build (safe) autonomy?



Perception: Estimate robot's state
Estimate fire map

Planner: Plan trajectories around fire perimeter

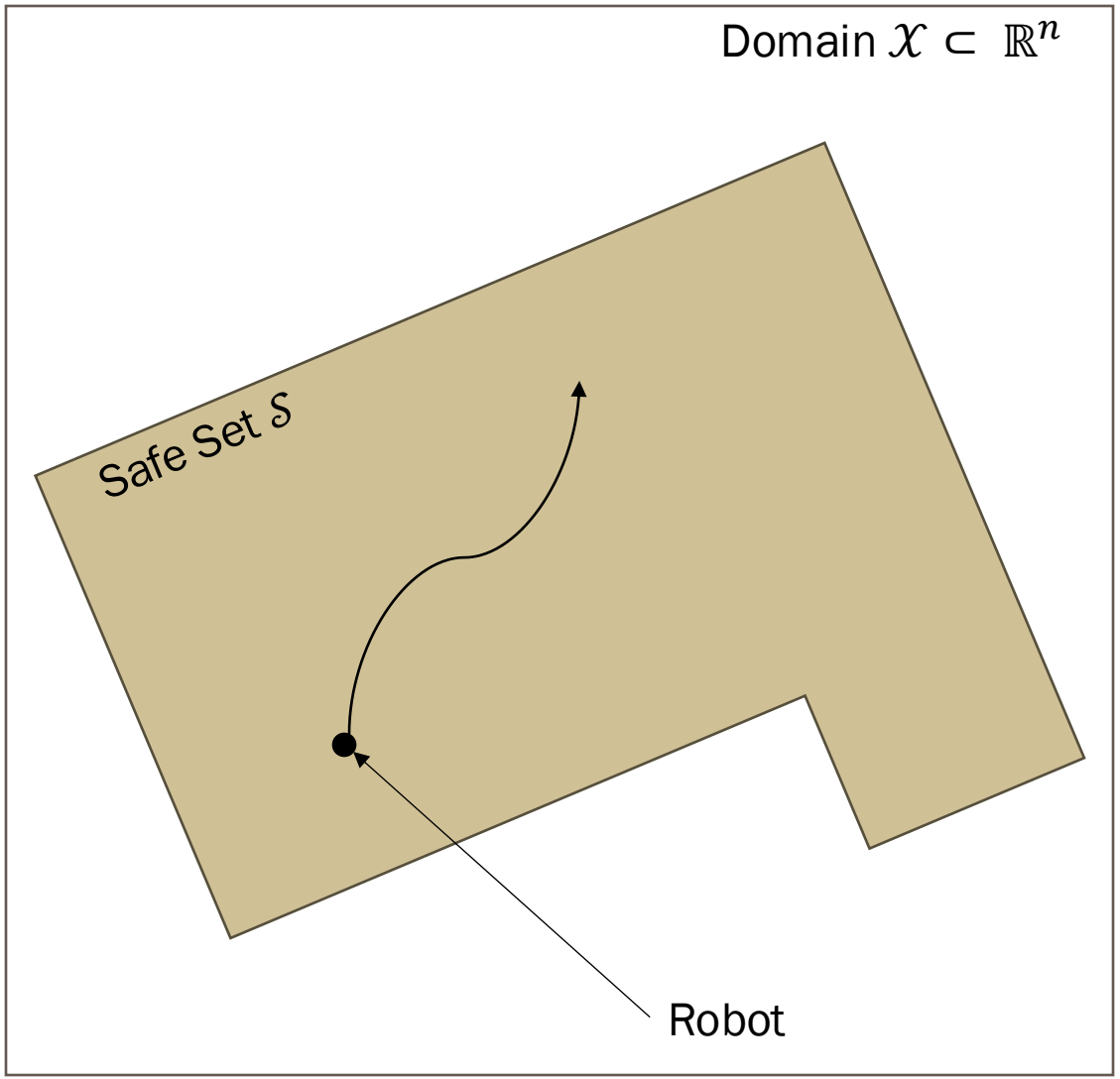
Controller: Compute control inputs to track the planned trajectory



The "autonomy stack" consists of many modules.

The only interface to the robot is the control input and the sensor output.

formalizing the notion of safety



Consider a dynamical system

$$\dot{x} = f(x, u)$$

state: $x \in \mathcal{X} \subset \mathbb{R}^n$ control input: $u \in \mathcal{U} \subset \mathbb{R}^m$

subject to constraints of the form

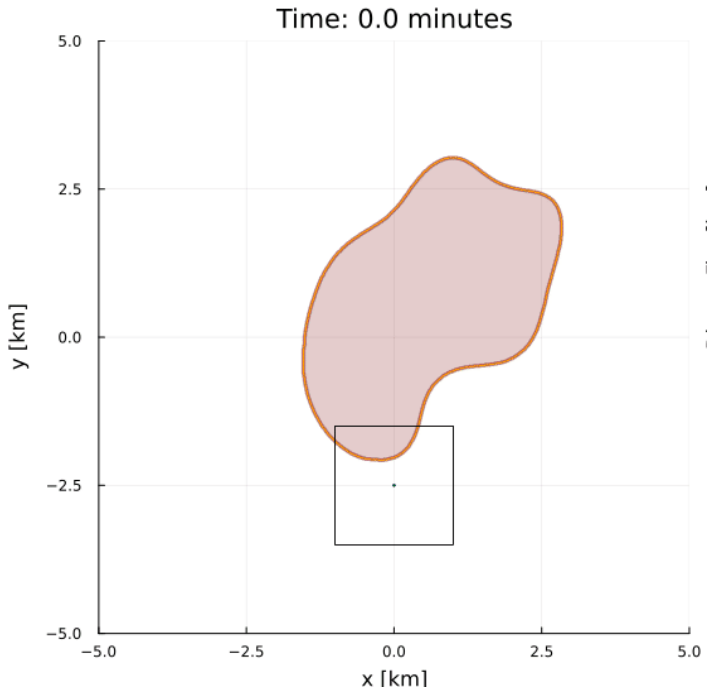
$$x \in \mathcal{S} = \{x : \underbrace{h_1(x) \geq 0}, \underbrace{h_2(x) \geq 0}, \ldots\}$$

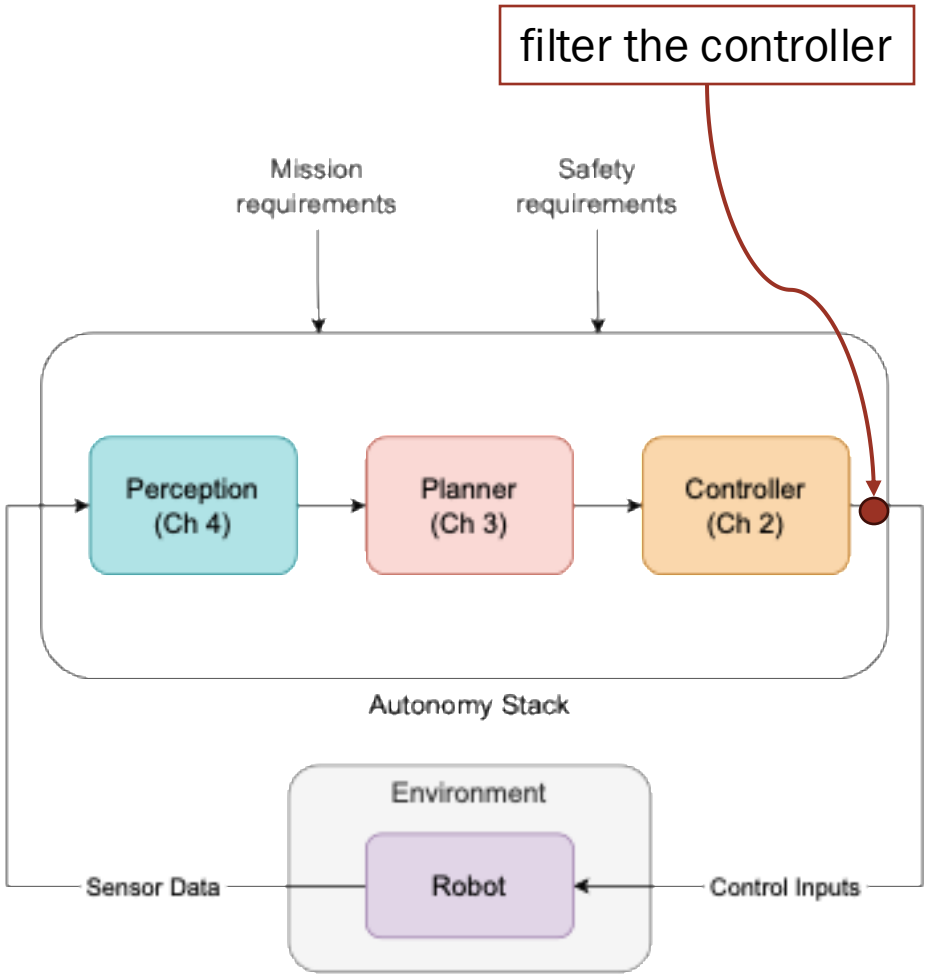
ex: avoid the walls, stay out of the fire

Goal is to ensure the closed-loop system satisfies

$$x(t) \in \mathcal{S} \quad \forall t \geq t_0.$$

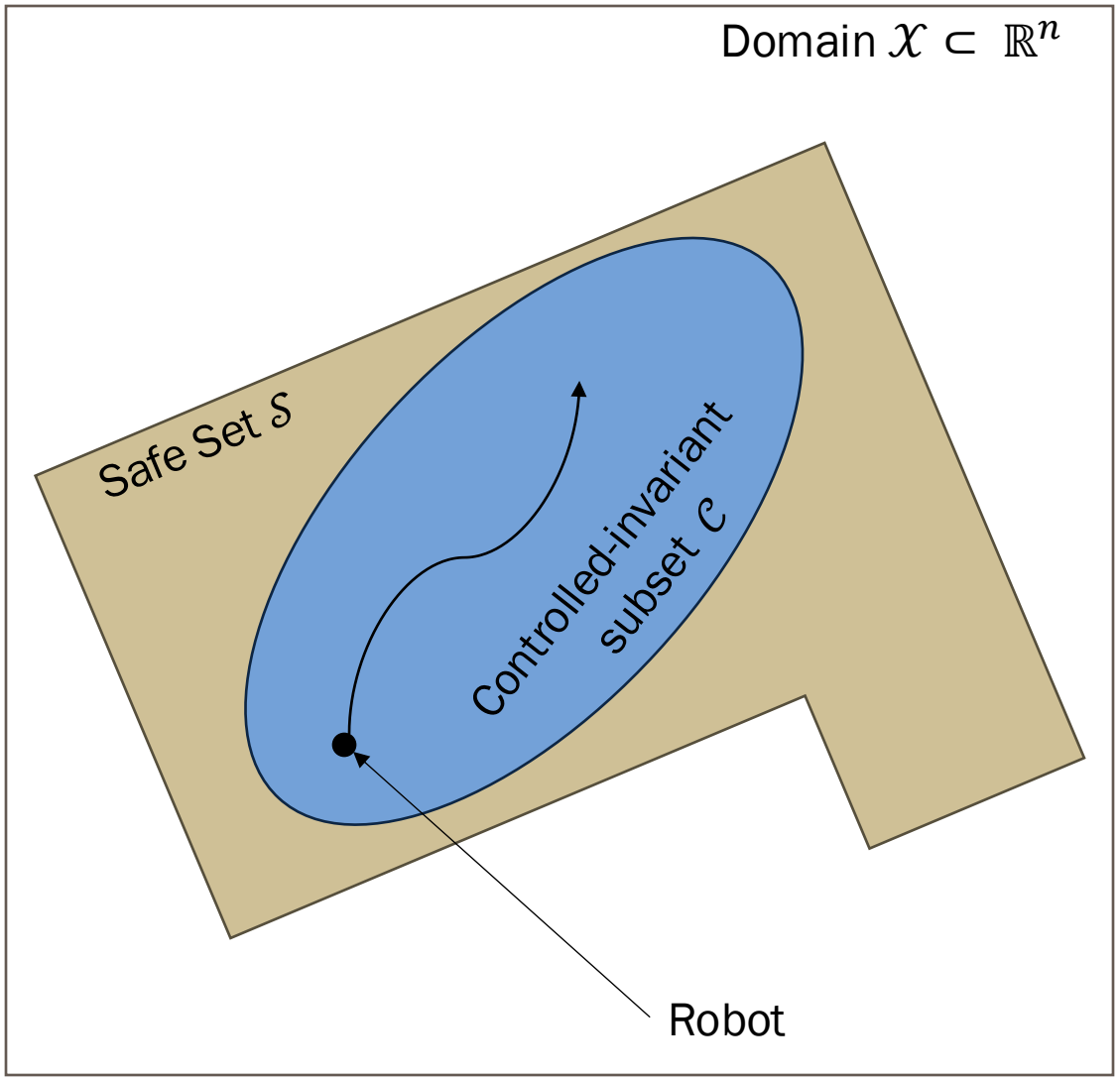
ways to achieve safety:





Since the control inputs determine the future trajectory, (and hence safety) can we “filter” the controller?

safety through filtering control inputs



Consider a (control-affine) dynamical system

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

with safety constraint $x(t) \in \mathcal{S} \quad \forall t \geq t_0$.
Suppose we can find $h : \mathcal{X} \to \mathbb{R}$,
such that

$$\mathcal{C} = \{x : h(x) \geq 0\} \subset \mathcal{S}.$$

If h is a CBF [1], then $\pi_{\text{CBF}} : \mathcal{X} \to \mathcal{U}$ (defined in [1]) ensures

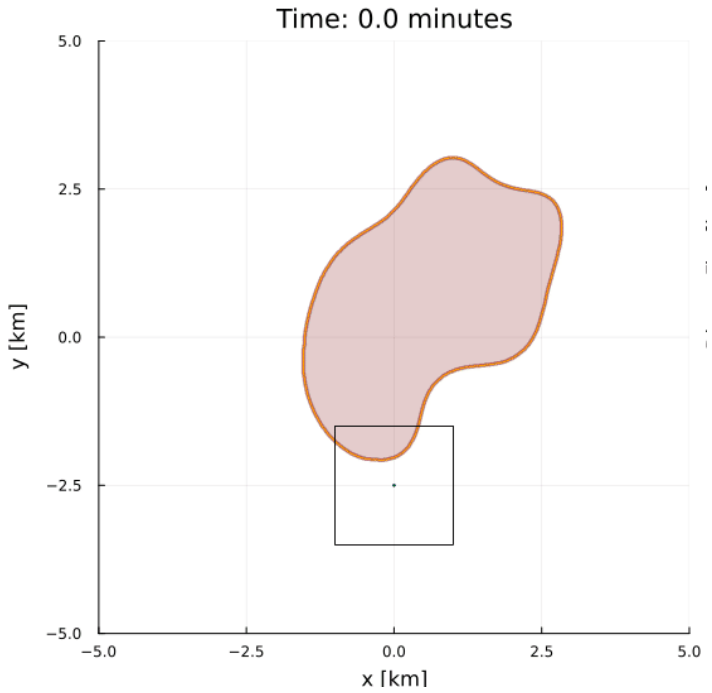
$$x(t_0) \in \mathcal{C} \implies x(t) \in \mathcal{C} \quad \forall t \geq t_0.$$

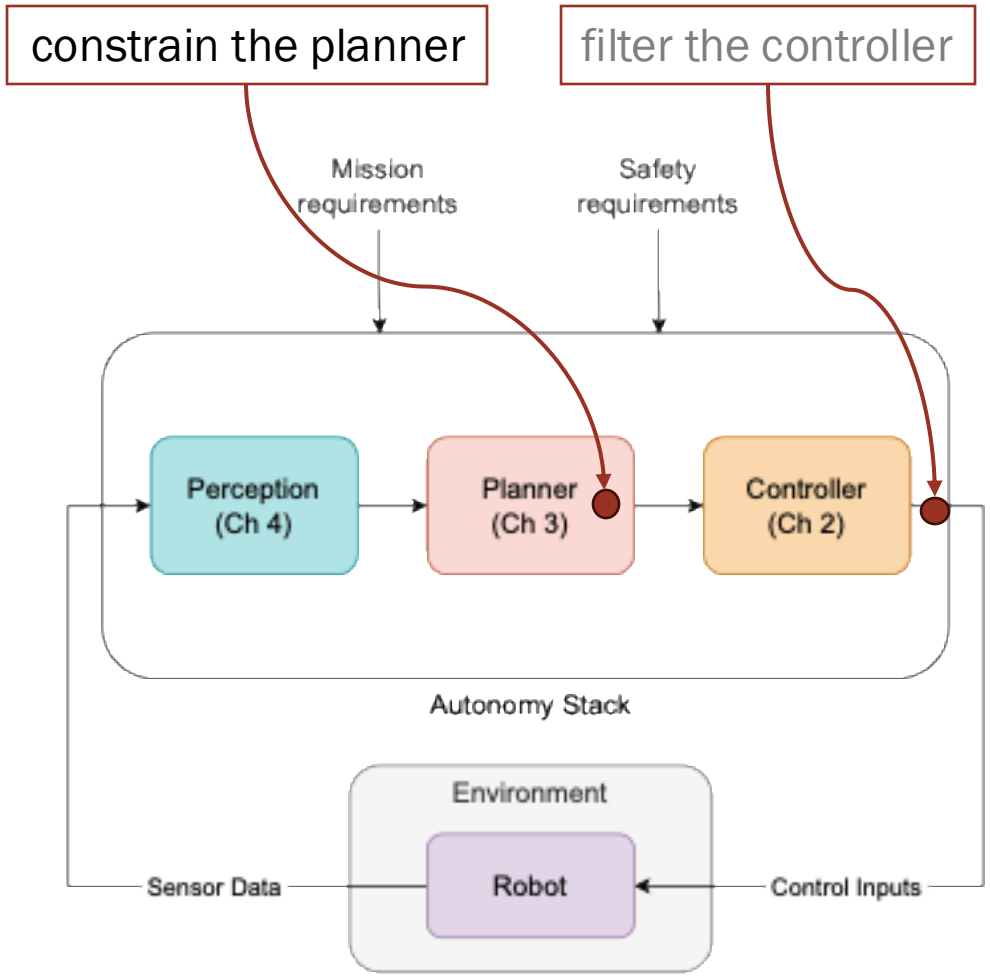
Therefore π_{CBF} is a **safe controller**.

But finding a CBF is hard.

[1] Ames et al. TAC 2017.

ways to achieve safety:

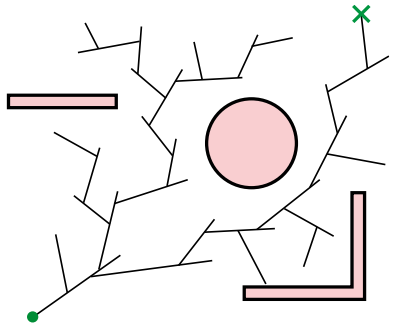




What if we made the safety constraints explicitly part of the planner?

safety through constraints in planner

Path Planning

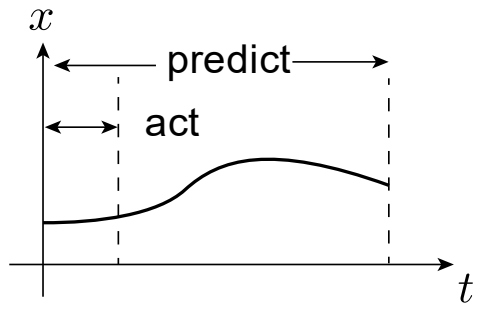


Ex. RRT*, A*, …

\+ can handle complex and dynamic environments

\- different models used by planner and controller leads to problems

Model Predictive Control



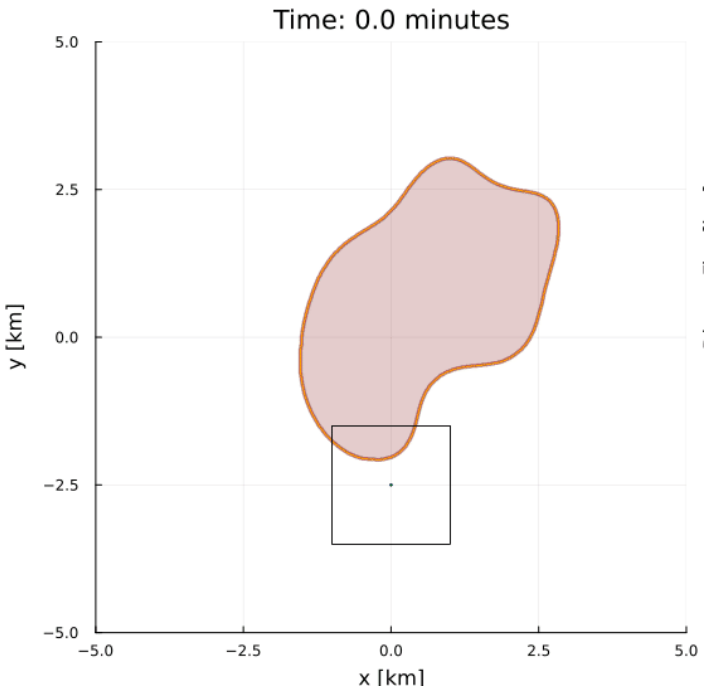
Ex. Nonlinear-MPC, MPPI, …

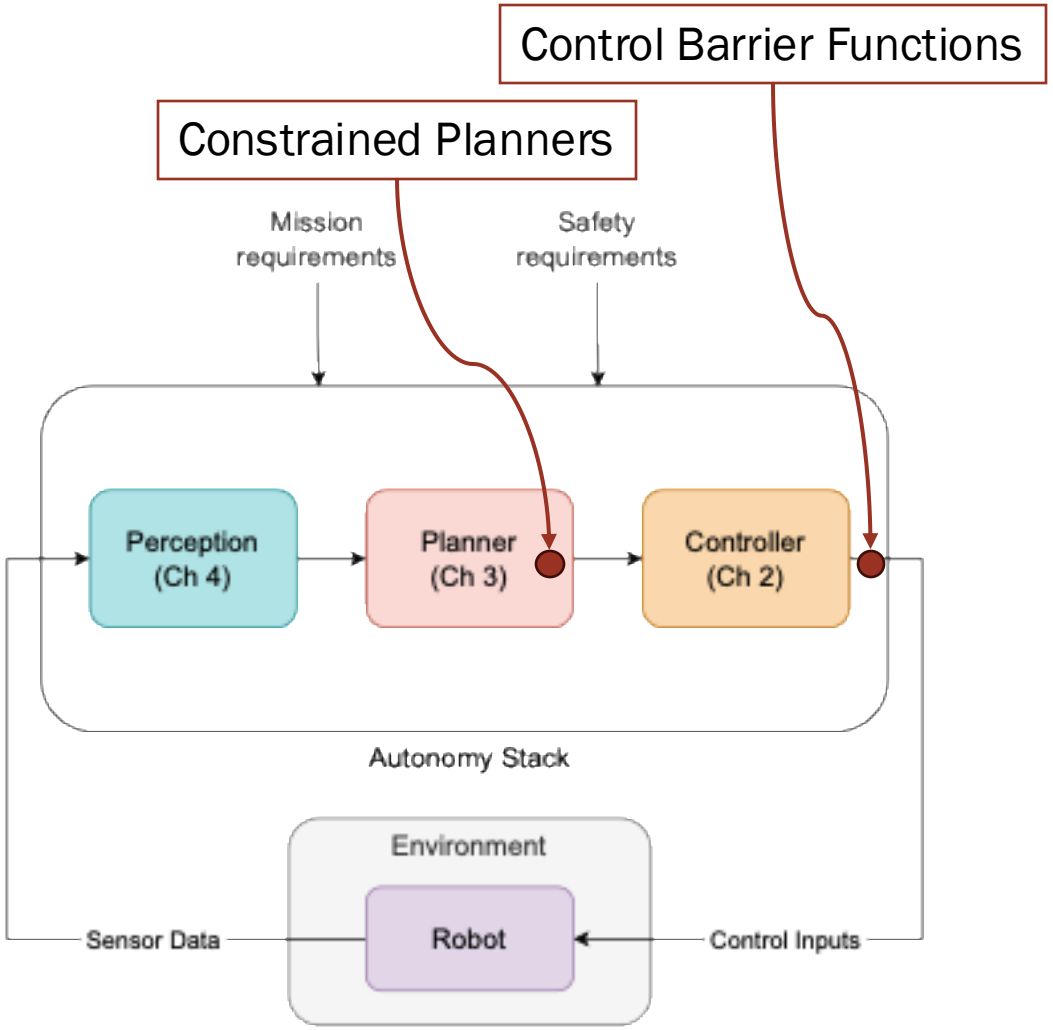
\+ considers future behavior of robot and environment

\- need to ensure recursive feasibility and stability

\- can be computationally challenging in nonconvex cases

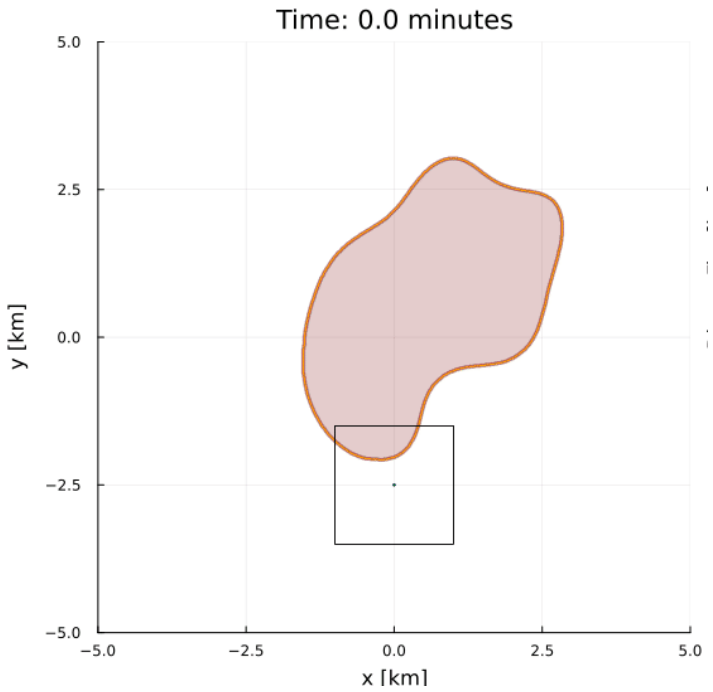
many ways to achieve safety:





How does a human achieve this complex mission?
By thinking about contingencies/fail-safes/backups.

perspective shift:



Instead of checking that a controller is safe for all initial states,

Design a controller $\pi : \mathcal{X} \to \mathcal{U}$ and $\mathcal{C} \subset \mathcal{S}$ such that

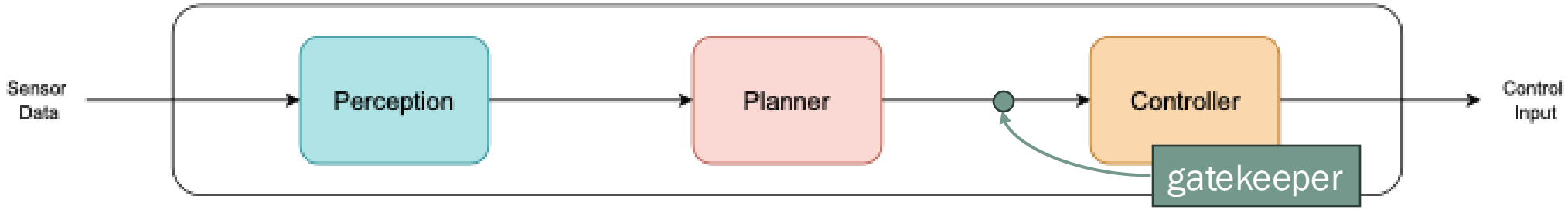
$$x(t_0) \in \mathcal{C} \implies x(t) \in \mathcal{C}\ \forall t \geq t_0.$$

What if we ensured that there exists a way to be safe from the current state?

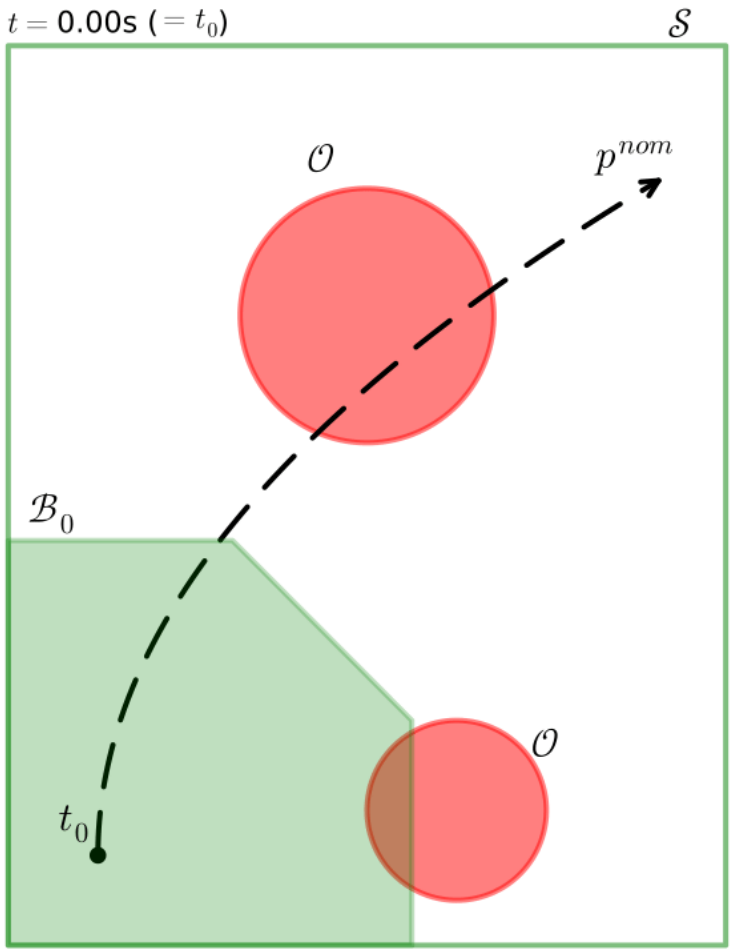
Ensure there exists a safe trajectory $p : [t_0, \infty) \to \mathcal{X}$ from the current state:

$$\exists p(t) \in \mathcal{S}\ \forall t \geq t_0, \quad p(t_0) = x(t_0)$$

Can we do this without compromising on the mission objectives?



gatekeeper



$\mathcal{S}$: safe set $\mathcal{O}$: obstacles $\mathcal{B}_k$: perceived safe set $\mathcal{C}_k$: backup set

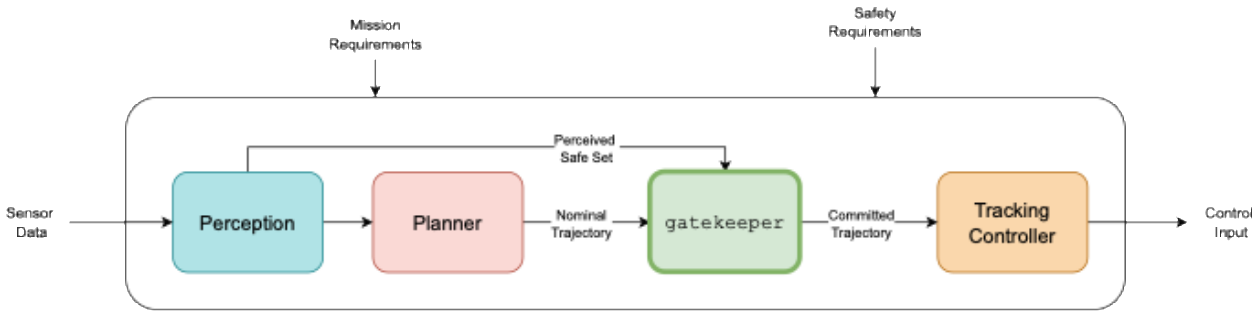
nominal traj candidate traj committed traj

Ensure there exists a safe trajectory $p : [t_0, \infty) \to \mathcal{X}$ from the current state:

$$\exists p(t) \in \mathcal{S}\ \forall t \geq t_0, \quad p(t_0) = x(t_0)$$

$\mathcal{S}(t)$ is the full safe set. Unknown, time-varying.

$\mathcal{B}_k(t)$ is the perceived safe set. Known, time-varying.

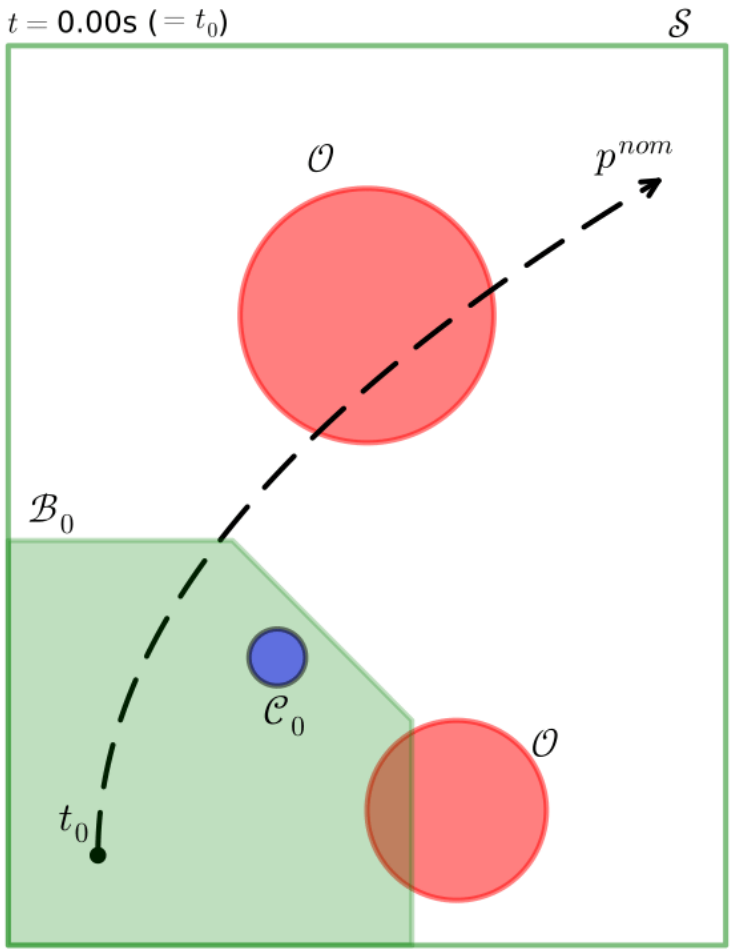


At the k-th iteration of `gatekeeper`,

- Construct p_k^{can}, a candidate trajectory.
- Check if p_k^{can} is valid.
- If valid, update the committed trajectory.

Controller tracks the last committed trajectory.

gatekeeper



$\mathcal{S}$: safe set $\mathcal{O}$: obstacles $\mathcal{B}_k$: perceived safe set $\mathcal{C}_k$: backup set

nominal traj candidate traj committed traj

Ensure there exists a safe trajectory $p : [t_0, \infty) \to \mathcal{X}$ from the current state:

$$\exists p(t) \in \mathcal{S}\ \forall t \geq t_0, \quad p(t_0) = x(t_0)$$

Core idea:

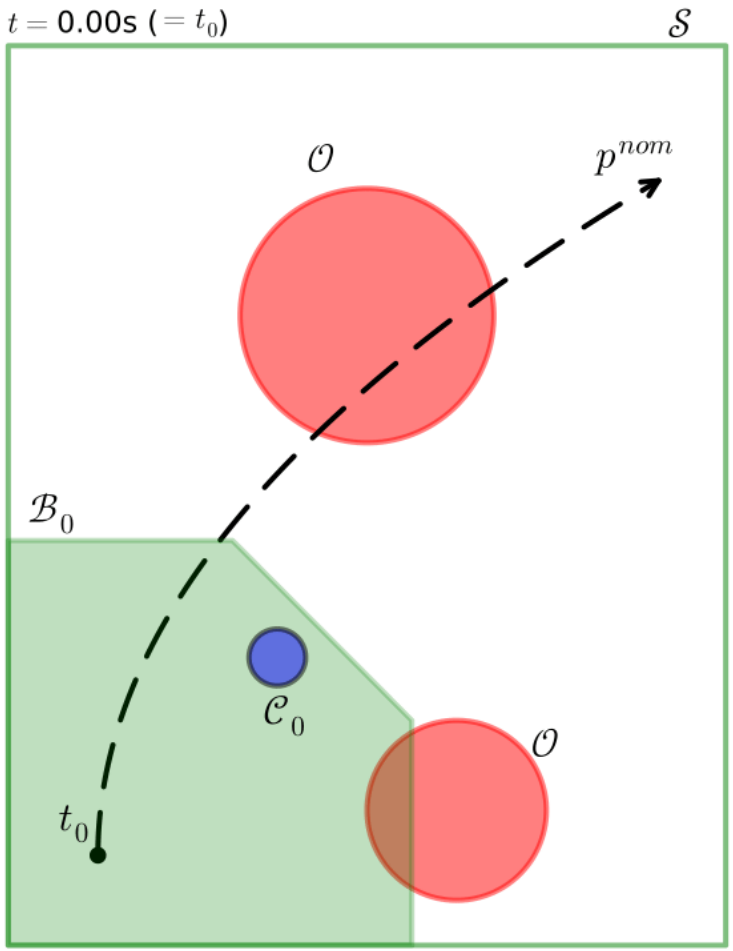
$$p_k^{\text{can}}(t) \in \mathcal{S}(t) \quad \forall t \geq t_k$$

$$\iff \begin{cases} p_k^{\text{can}}(t) \in \mathcal{S}(t) & \text{if } t \in [t_k, t_{kB}), \\ p_k^{\text{can}}(t) \in \mathcal{S}(t) & \text{if } t \in [t_{kB}, \infty) \end{cases}$$

$$\impliedby \begin{cases} p_k^{\text{can}}(t) \in \mathcal{B}_k(t) & \text{if } t \in [t_k, t_{kB}), \\ p_k^{\text{can}}(t_{kB}) \in \mathcal{C}_k(t_{kB}) \end{cases}$$

If p_k^{can} is valid, make it a committed trajectory.
Controller tracks the last committed trajectory.
Recursive guarantee of safety.

gatekeeper



$\mathcal{S}$: safe set $\mathcal{O}$: obstacles $\mathcal{B}_k$: perceived safe set $\mathcal{C}_k$: backup set

nominal traj candidate traj committed traj

Ensure there exists a safe trajectory $p : [t_0, \infty) \to \mathcal{X}$ from the current state:

$$\exists p(t) \in \mathcal{S}\ \forall t \geq t_0, \quad p(t_0) = x(t_0)$$

Core idea:

Split into finite horizon

$$p_k^{\text{can}}(t) \in \mathcal{S}(t) \quad \forall t \geq t_k$$

$$\iff \begin{cases} p_k^{\text{can}}(t) \in \mathcal{S}(t) & \text{if } t \in [t_k, t_{kB}), \\ p_k^{\text{can}}(t) \in \mathcal{S}(t) & \text{if } t \in [t_{kB}, \infty) \end{cases}$$

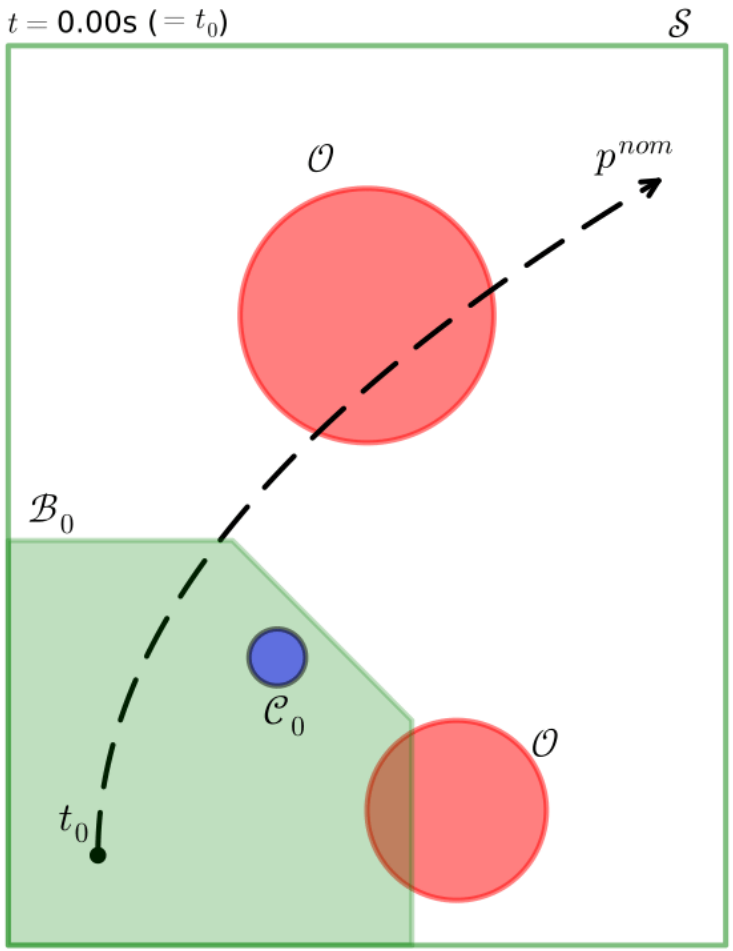
$$\impliedby \begin{cases} p_k^{\text{can}}(t) \in \mathcal{B}_k(t) & \text{if } t \in [t_k, t_{kB}), \\ p_k^{\text{can}}(t_{kB}) \in \mathcal{C}_k(t_{kB}) \end{cases}$$

If p_k^{can} is valid, make it a committed trajectory.
Controller tracks the last committed trajectory.
Recursive guarantee of safety.

gatekeeper

Ensure there exists a safe trajectory $p : [t_0, \infty) \to \mathcal{X}$ from the current state:

$$\exists p(t) \in \mathcal{S} \; \forall t \geq t_0, \quad p(t_0) = x(t_0)$$



$\mathcal{S}$: safe set $\mathcal{O}$: obstacles $\mathcal{B}_k$: perceived safe set $\mathcal{C}_k$: backup set

nominal traj candidate traj committed traj

Core idea:

$$p_k^{\text{can}}(t) \in \mathcal{S}(t) \quad \forall t \geq t_k$$

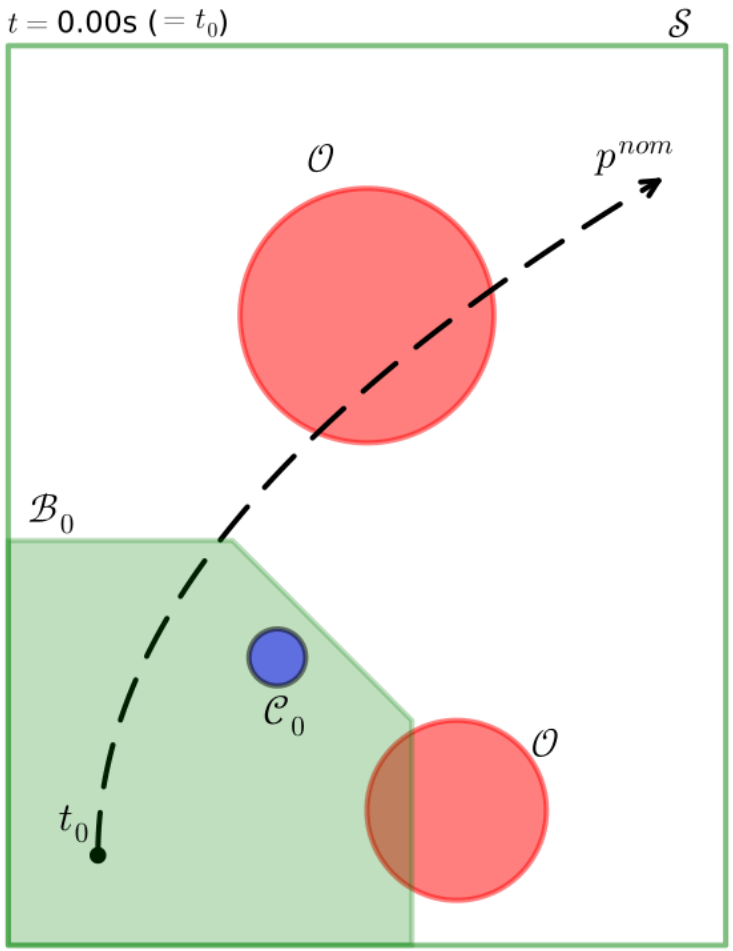
$$\impliedby \begin{cases} p_k^{\text{can}}(t) \in \mathcal{S}(t) & \text{if } t \in [t_k, t_{kB}), \\ p_k^{\text{can}}(t) \in \mathcal{S}(t) & \text{if } t \in [t_{kB}, \infty) \end{cases}$$

$$\impliedby \begin{cases} p_k^{\text{can}}(t) \in \mathcal{B}_k(t) & \text{if } t \in [t_k, t_{kB}), \\ p_k^{\text{can}}(t_{kB}) \in \mathcal{C}_k(t_{kB}) \end{cases}$$

Check that it safely reaches $\mathcal{C}_k$

If p_k^{can} is valid, make it a comm
Controller tracks the last committed trajectory.
Recursive guarantee of safety.

gatekeeper



$\mathcal{S}$: safe set $\mathcal{O}$: obstacles $\mathcal{B}_k$: perceived safe set $\mathcal{C}_k$: backup set

nominal traj candidate traj committed traj

Ensure there exists a safe trajectory $p : [t_0, \infty) \to \mathcal{X}$ from the current state:

$$\exists p(t) \in \mathcal{S}\ \forall t \geq t_0, \quad p(t_0) = x(t_0)$$

Core idea:

$$p_k^{\text{can}}(t) \in \mathcal{S}(t) \quad \forall t \geq t_k$$

$$\iff \begin{cases} p_k^{\text{can}}(t) \in \mathcal{S}(t) & \text{if } t \in [t_k, t_{kB}), \\ p_k^{\text{can}}(t) \in \mathcal{S}(t) & \text{if } t \in [t_{kB}, \infty) \end{cases}$$

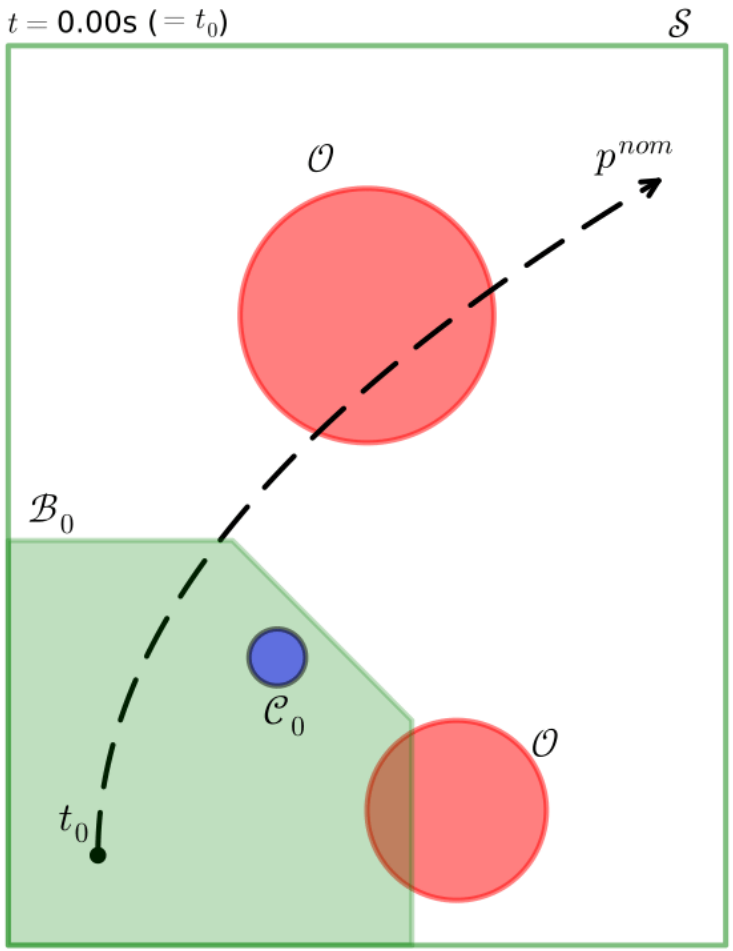
$$\impliedby \begin{cases} p_k^{\text{can}}(t) \in \mathcal{B}_k(t) & \text{if } t \in [t_k, t_{kB}), \\ p_k^{\text{can}}(t_{kB}) \in \mathcal{C}_k(t_{kB}) \end{cases}$$

If p_k^{can} is valid, make it a committed trajectory.
Controller tracks the last committed trajectory.
Recursive guarantee of safety.

gatekeeper

Ensure there exists a safe trajectory $p : [t_0, \infty) \to \mathcal{X}$ from the current state:

$$\exists p(t) \in \mathcal{S} \; \forall t \geq t_0, \quad p(t_0) = x(t_0)$$



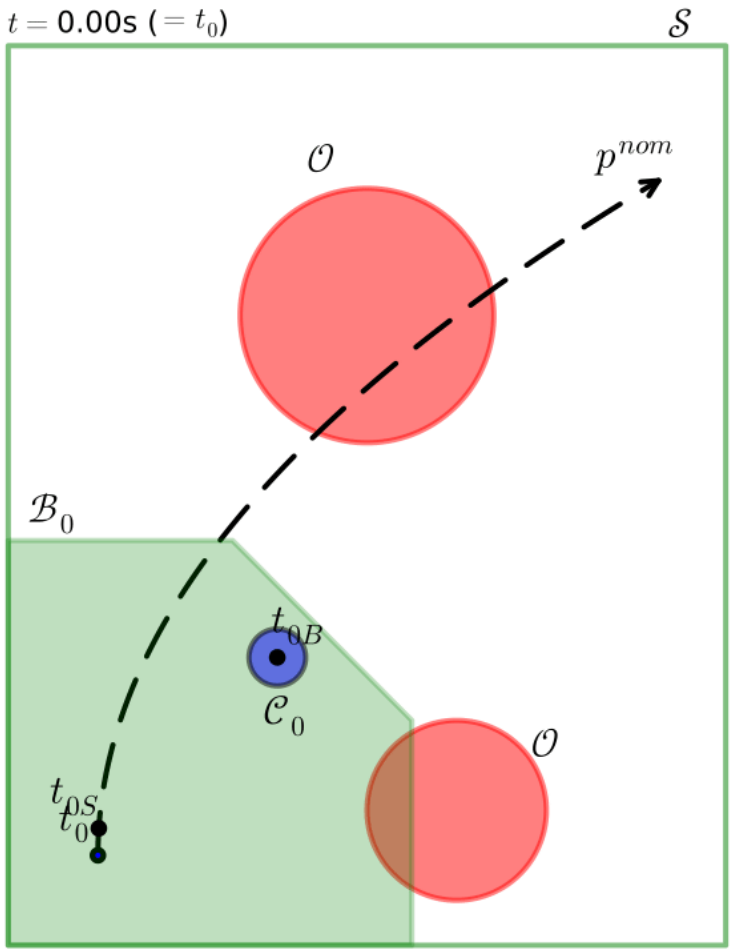
$\mathcal{S}$: safe set $\mathcal{O}$: obstacles $\mathcal{B}_k$: perceived safe set $\mathcal{C}_k$: backup set

nominal traj candidate traj committed traj

- *Step 1:* Identify $\mathcal{C}_k$
- *Step 2:* Construct a candidate trajectory
  - Choose a switch time t_{kS}
  - Forward propagate tracking nominal until t_{kS}
  - Forward propagate backup controller until t_{kB}
  - Choose largest t_{kS} such that candidate is valid
- *Step 3:* Update committed trajectory
  - If we found a candidate trajectory is valid, replace committed trajectory.
  - Else, keep old committed trajectory.

Controller always tracks the last committed trajectory.

gatekeeper



$\mathcal{S}$: safe set $\mathcal{O}$: obstacles $\mathcal{B}_k$: perceived safe set $\mathcal{C}_k$: backup set

nominal traj candidate traj committed traj

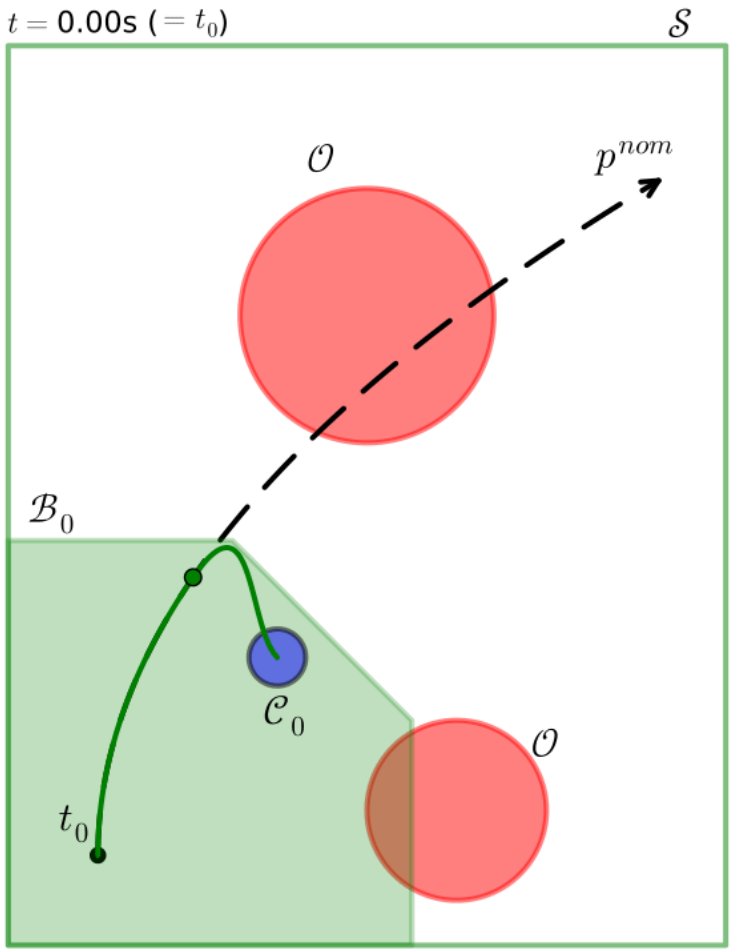
Ensure there exists a safe trajectory $p : [t_0, \infty) \to \mathcal{X}$ from the current state:

$$\exists p(t) \in \mathcal{S}\ \forall t \geq t_0, \quad p(t_0) = x(t_0)$$

- *Step 1:* Identify $\mathcal{C}_k$
- *Step 2:* Construct a candidate trajectory
  - Choose a switch time t_{kS}
  - Forward propagate tracking nominal until t_{kS}
  - Forward propagate backup controller until t_{kB}
  - Choose largest t_{kS} such that candidate is valid
- *Step 3:* Update committed trajectory
  - If we found a candidate trajectory is valid, replace committed trajectory.
  - Else, keep old committed trajectory.

Controller always tracks the last committed trajectory.

gatekeeper



$\mathcal{S}$: safe set $\mathcal{O}$: obstacles $\mathcal{B}_k$: perceived safe set $\mathcal{C}_k$: backup set

nominal traj candidate traj committed traj

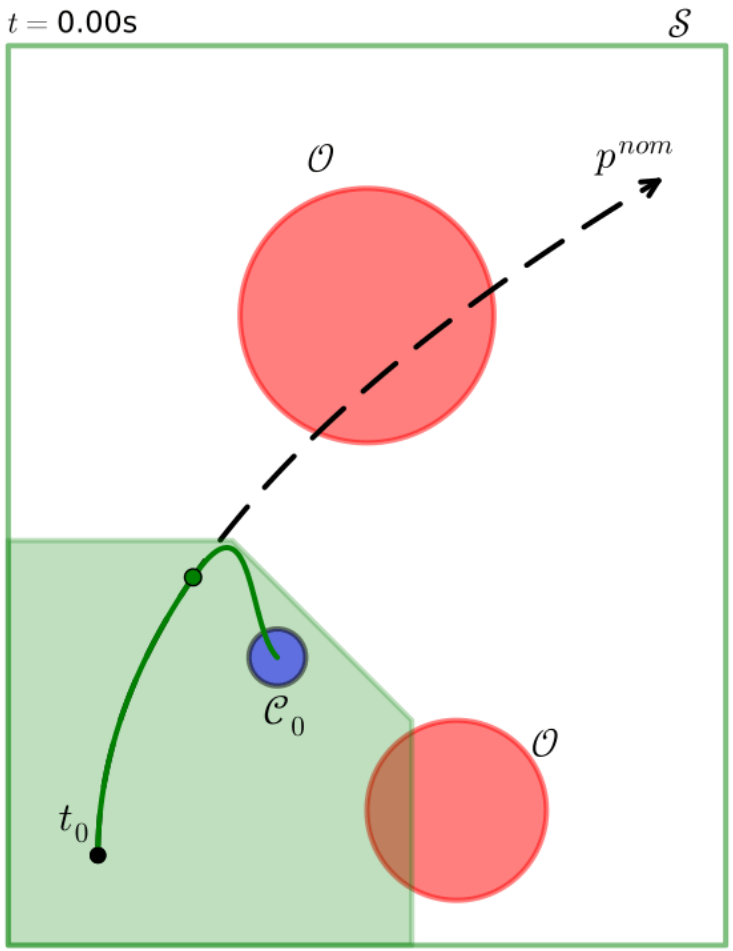
Ensure there exists a safe trajectory $p : [t_0, \infty) \to \mathcal{X}$ from the current state:

$$\exists p(t) \in \mathcal{S}\ \forall t \geq t_0, \quad p(t_0) = x(t_0)$$

- *Step 1:* Identify $\mathcal{C}_k$
- *Step 2:* Construct a candidate trajectory
  - Choose a switch time t_{kS}
  - Forward propagate tracking nominal until t_{kS}
  - Forward propagate backup controller until t_{kB}
  - Choose largest t_{kS} such that candidate is valid
- *Step 3:* Update committed trajectory
  - If we found a candidate trajectory is valid, replace committed trajectory.
  - Else, keep old committed trajectory.

Controller always tracks the last committed trajectory.

gatekeeper



$\mathcal{S}$: safe set $\mathcal{O}$: obstacles $\mathcal{B}_k$: perceived safe set $\mathcal{C}_k$: backup set

nominal traj candidate traj committed traj

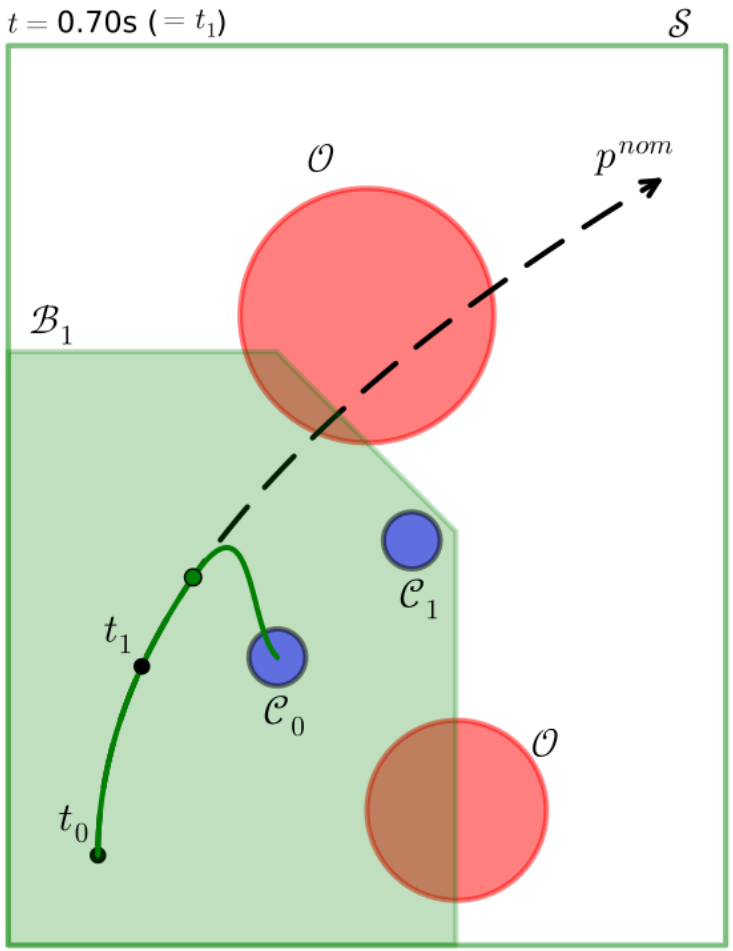
Ensure there exists a safe trajectory $p : [t_0, \infty) \to \mathcal{X}$ from the current state:

$$\exists p(t) \in \mathcal{S} \; \forall t \geq t_0, \quad p(t_0) = x(t_0)$$

- *Step 1:* Identify $\mathcal{C}_k$
- *Step 2:* Construct a candidate trajectory
  - Choose a switch time t_{kS}
  - Forward propagate tracking nominal until t_{kS}
  - Forward propagate backup controller until t_{kB}
  - Choose largest t_{kS} such that candidate is valid
- *Step 3:* Update committed trajectory
  - If we found a candidate trajectory is valid, replace committed trajectory.
  - Else, keep old committed trajectory.

Controller always tracks the last committed trajectory.

gatekeeper



$\mathcal{S}$: safe set $\mathcal{O}$: obstacles $\mathcal{B}_k$: perceived safe set $\mathcal{C}_k$: backup set

nominal traj candidate traj committed traj

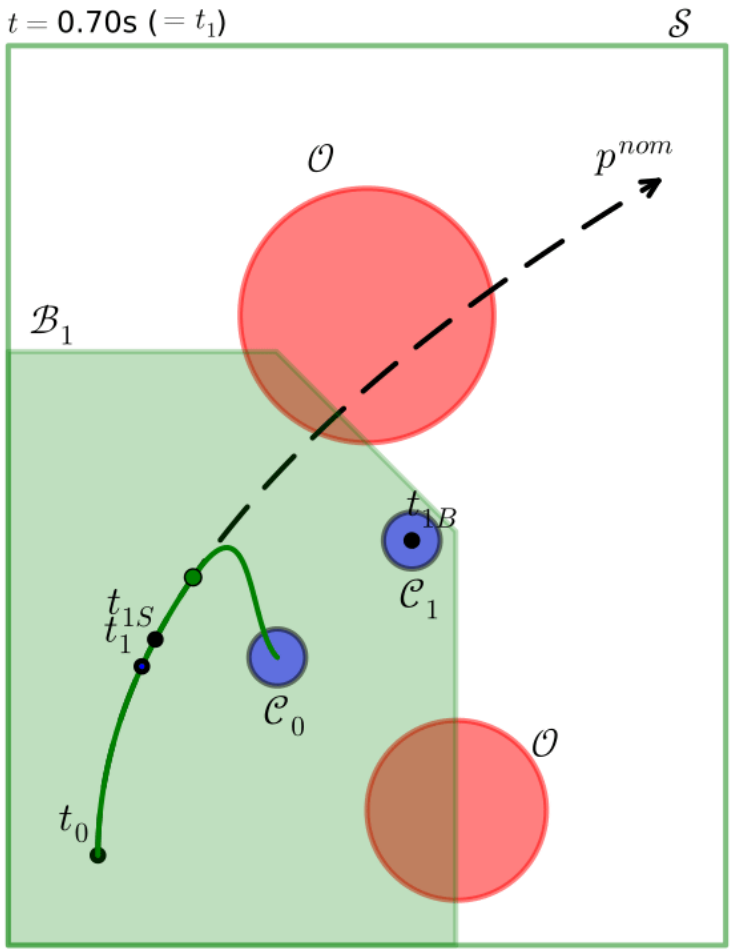
Ensure there exists a safe trajectory $p : [t_0, \infty) \to \mathcal{X}$ from the current state:

$$\exists p(t) \in \mathcal{S}\ \forall t \geq t_0, \quad p(t_0) = x(t_0)$$

- *Step 1:* Identify $\mathcal{C}_k$
- *Step 2:* Construct a candidate trajectory
  - Choose a switch time t_{kS}
  - Forward propagate tracking nominal until t_{kS}
  - Forward propagate backup controller until t_{kB}
  - Choose largest t_{kS} such that candidate is valid
- *Step 3:* Update committed trajectory
  - If we found a candidate trajectory is valid, replace committed trajectory.
  - Else, keep old committed trajectory.

Controller always tracks the last committed trajectory.

gatekeeper



$\mathcal{S}$: safe set $\mathcal{O}$: obstacles $\mathcal{B}_k$: perceived safe set $\mathcal{C}_k$: backup set

nominal traj candidate traj committed traj

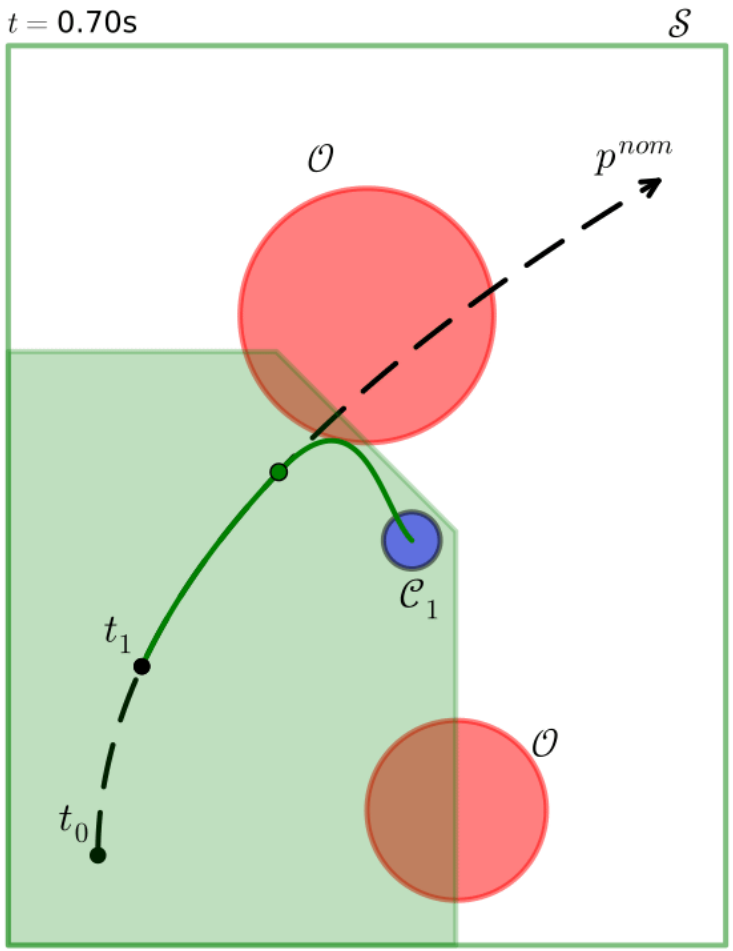
Ensure there exists a safe trajectory $p : [t_0, \infty) \to \mathcal{X}$ from the current state:

$$\exists p(t) \in \mathcal{S}\ \forall t \geq t_0, \quad p(t_0) = x(t_0)$$

- *Step 1:* Identify $\mathcal{C}_k$
- *Step 2:* Construct a candidate trajectory
  - Choose a switch time t_{kS}
  - Forward propagate tracking nominal until t_{kS}
  - Forward propagate backup controller until t_{kB}
  - Choose largest t_{kS} such that candidate is valid
- *Step 3:* Update committed trajectory
  - If we found a candidate trajectory is valid, replace committed trajectory.
  - Else, keep old committed trajectory.

Controller always tracks the last committed trajectory.

gatekeeper



$\mathcal{S}$: safe set $\mathcal{O}$: obstacles $\mathcal{B}_k$: perceived safe set $\mathcal{C}_k$: backup set

nominal traj candidate traj committed traj

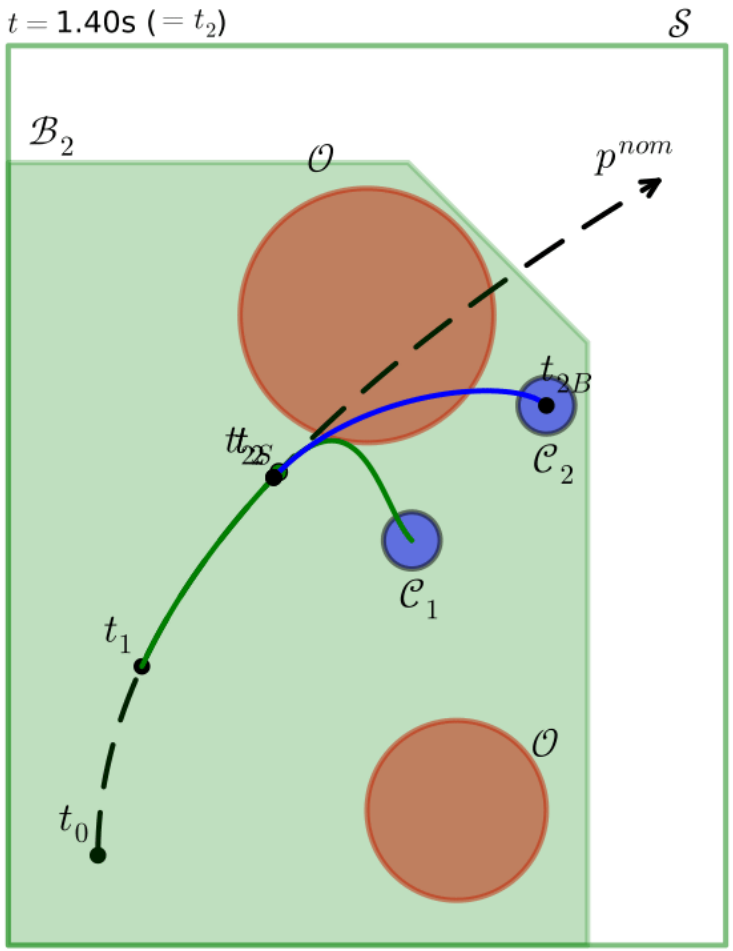
Ensure there exists a safe trajectory $p : [t_0, \infty) \to \mathcal{X}$ from the current state:

$$\exists p(t) \in \mathcal{S}\ \forall t \geq t_0, \quad p(t_0) = x(t_0)$$

- *Step 1:* Identify $\mathcal{C}_k$
- *Step 2:* Construct a candidate trajectory
  - Choose a switch time t_{kS}
  - Forward propagate tracking nominal until t_{kS}
  - Forward propagate backup controller until t_{kB}
  - Choose largest t_{kS} such that candidate is valid
- *Step 3:* Update committed trajectory
  - If we found a candidate trajectory is valid, replace committed trajectory.
  - Else, keep old committed trajectory.

Controller always tracks the last committed trajectory.

gatekeeper



$\mathcal{S}$: safe set $\mathcal{O}$: obstacles $\mathcal{B}_k$: perceived safe set $\mathcal{C}_k$: backup set

nominal traj candidate traj committed traj

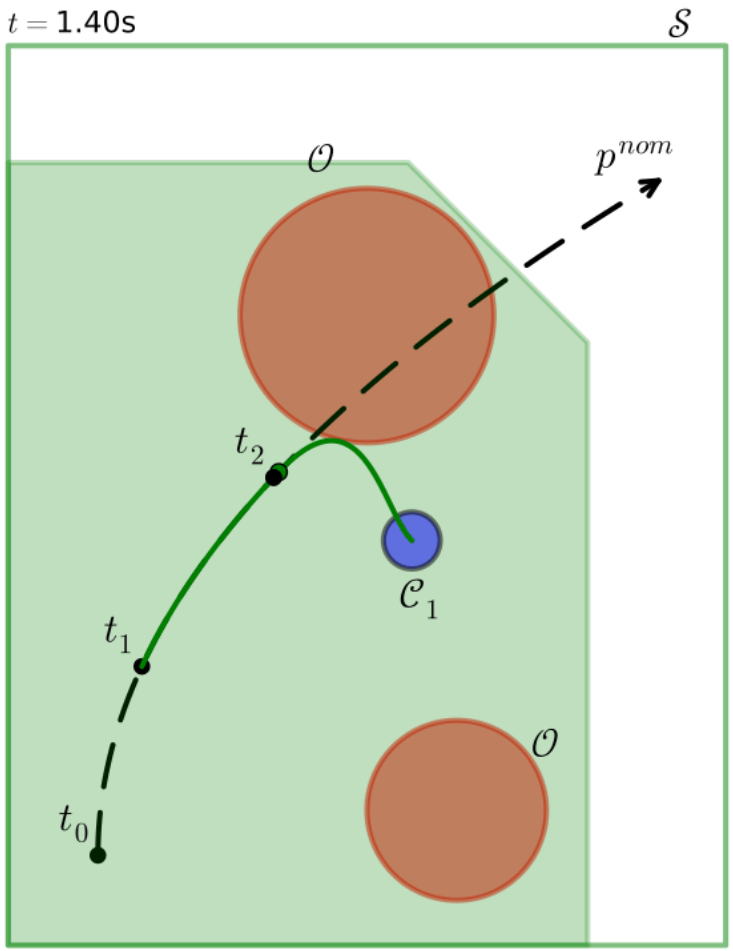
Ensure there exists a safe trajectory $p : [t_0, \infty) \to \mathcal{X}$ from the current state:

$$\exists p(t) \in \mathcal{S}\ \forall t \geq t_0, \quad p(t_0) = x(t_0)$$

- *Step 1:* Identify $\mathcal{C}_k$
- *Step 2:* Construct a candidate trajectory
  - Choose a switch time t_{kS}
  - Forward propagate tracking nominal until t_{kS}
  - Forward propagate backup controller until t_{kB}
  - Choose largest t_{kS} such that candidate is valid
- *Step 3:* Update committed trajectory
  - If we found a candidate trajectory is valid, replace committed trajectory.
  - Else, keep old committed trajectory.

Controller always tracks the last committed trajectory.

gatekeeper



$\mathcal{S}$: safe set $\mathcal{O}$: obstacles $\mathcal{B}_k$: perceived safe set $\mathcal{C}_k$: backup set

nominal traj candidate traj committed traj

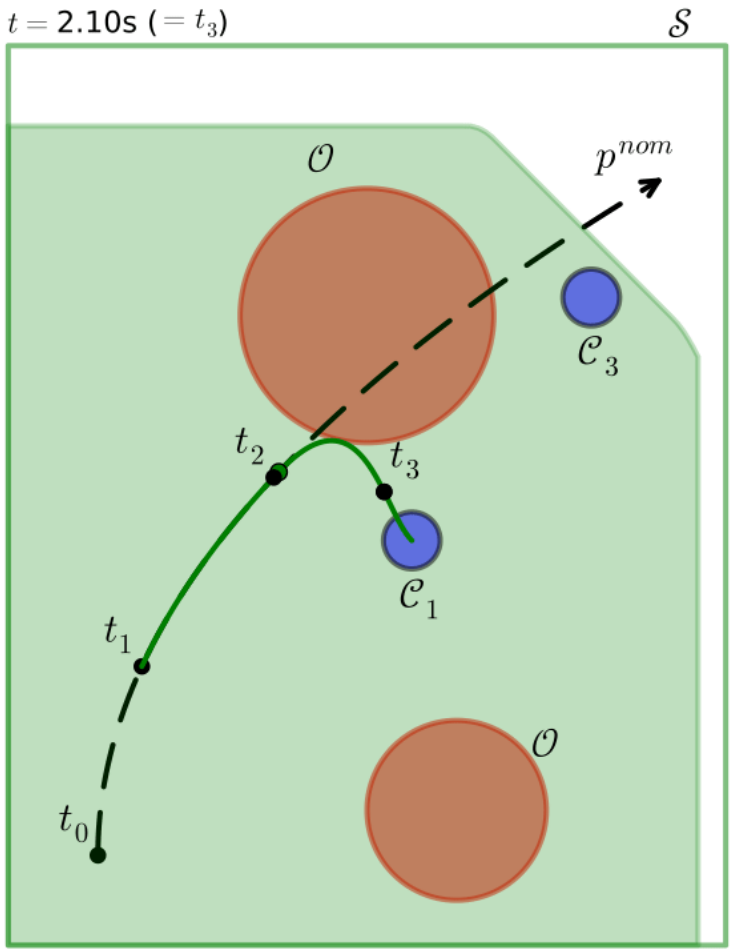
Ensure there exists a safe trajectory $p : [t_0, \infty) \to \mathcal{X}$ from the current state:

$$\exists p(t) \in \mathcal{S}\ \forall t \geq t_0, \quad p(t_0) = x(t_0)$$

- *Step 1:* Identify $\mathcal{C}_k$
- *Step 2:* Construct a candidate trajectory
  - Choose a switch time t_{kS}
  - Forward propagate tracking nominal until t_{kS}
  - Forward propagate backup controller until t_{kB}
  - Choose largest t_{kS} such that candidate is valid
- *Step 3:* Update committed trajectory
  - If we found a candidate trajectory is valid, replace committed trajectory.
  - Else, keep old committed trajectory.

Controller always tracks the last committed trajectory.

gatekeeper



$\mathcal{S}$: safe set $\mathcal{O}$: obstacles $\mathcal{B}_k$: perceived safe set $\mathcal{C}_k$: backup set

nominal traj candidate traj committed traj

Ensure there exists a safe trajectory $p : [t_0, \infty) \to \mathcal{X}$ from the current state:

$$\exists p(t) \in \mathcal{S}\ \forall t \geq t_0, \quad p(t_0) = x(t_0)$$

`gatekeeper` provides recursive guarantee of safety

- for nonlinear dynamics
- for multiple constraints
- with inputs bounds
- possibly time-varying safe sets or dynamics
- partially known safe sets
- very low compute cost
- robust to disturbances and observer error

It **does** assume

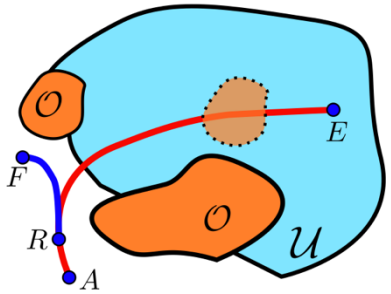
- known backup controller and backup safe set
- known tracking controller

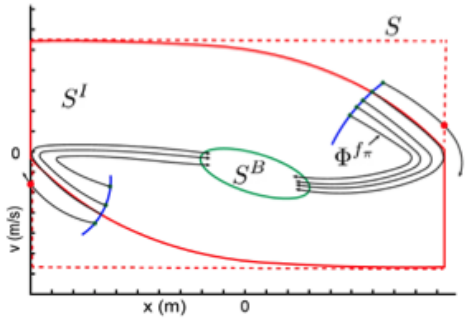
It **does not** assume:

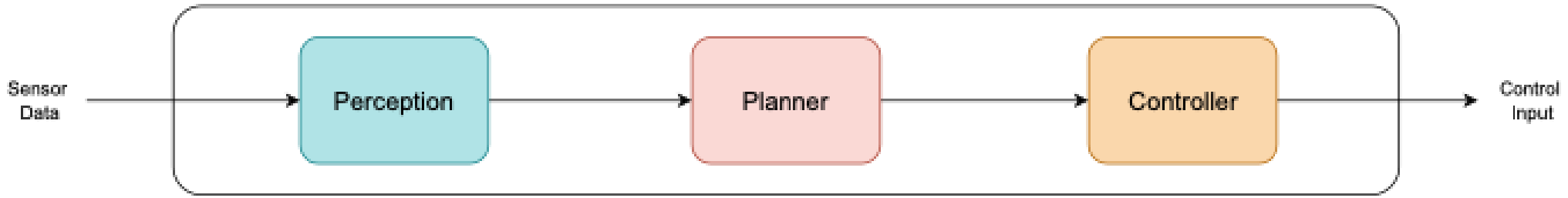
- convexity of safe sets/dynamics
- nominal plan is dynamically feasible

related approaches:

FASTER	Backup filters
Tordesillas et al, TRO 2021.	Singletary et al, RAL 2022





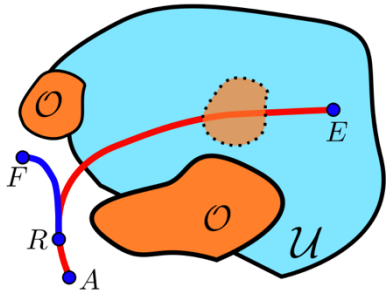


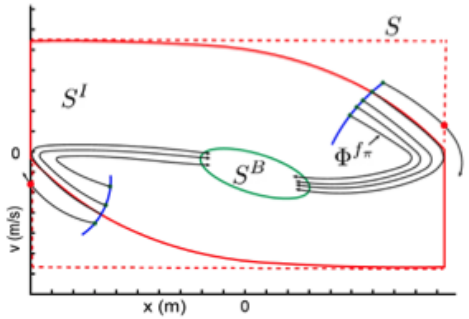
related approaches:

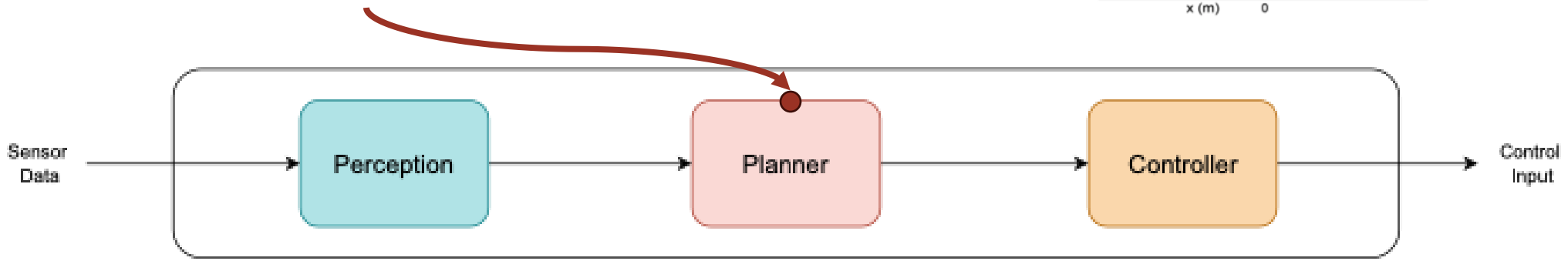
FASTER	Backup filters
Tordesillas et al, TRO 2021.	Singletary et al, RAL 2022

Use MPC to simultaneously solve for both nominal trajectory and a backup trajectory.

But assumes linear dynamics and static convex safe sets (or SFCs) to maintain recursive feasibility.







related approaches:

FASTER

Tordesillas et al, TRO 2021.

Use MPC to simultaneously solve for both nominal trajectory and a backup trajectory.

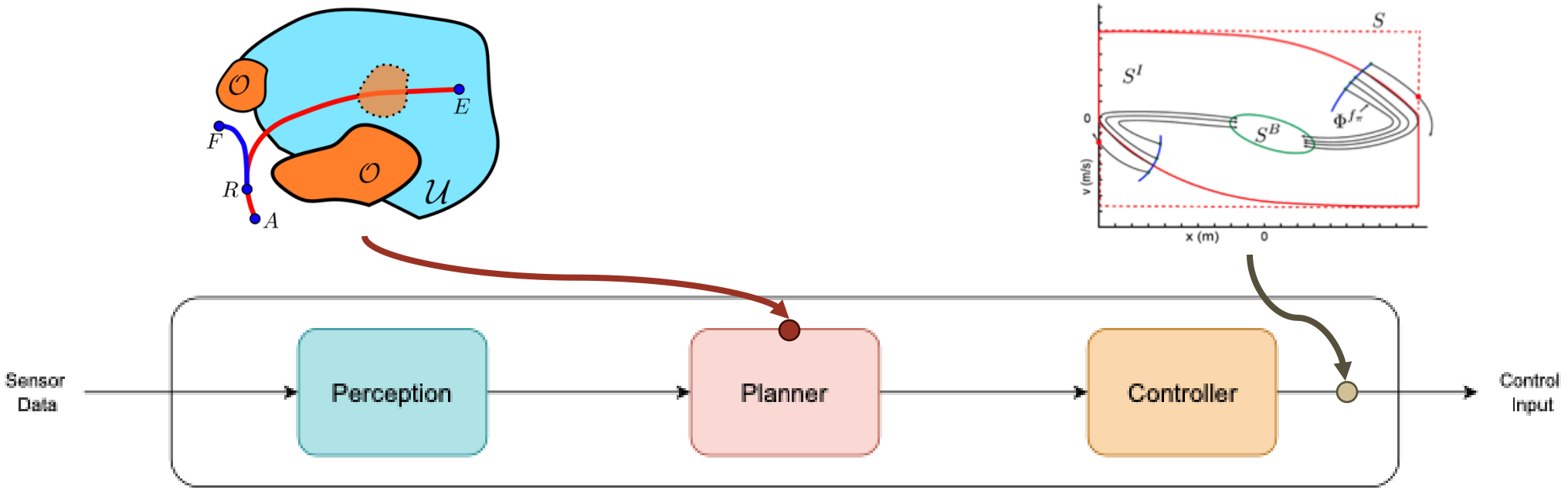
But assumes linear dynamics and static convex safe sets (or SFCs) to maintain recursive feasibility.

Backup filters

Singletary et al, RAL 2022

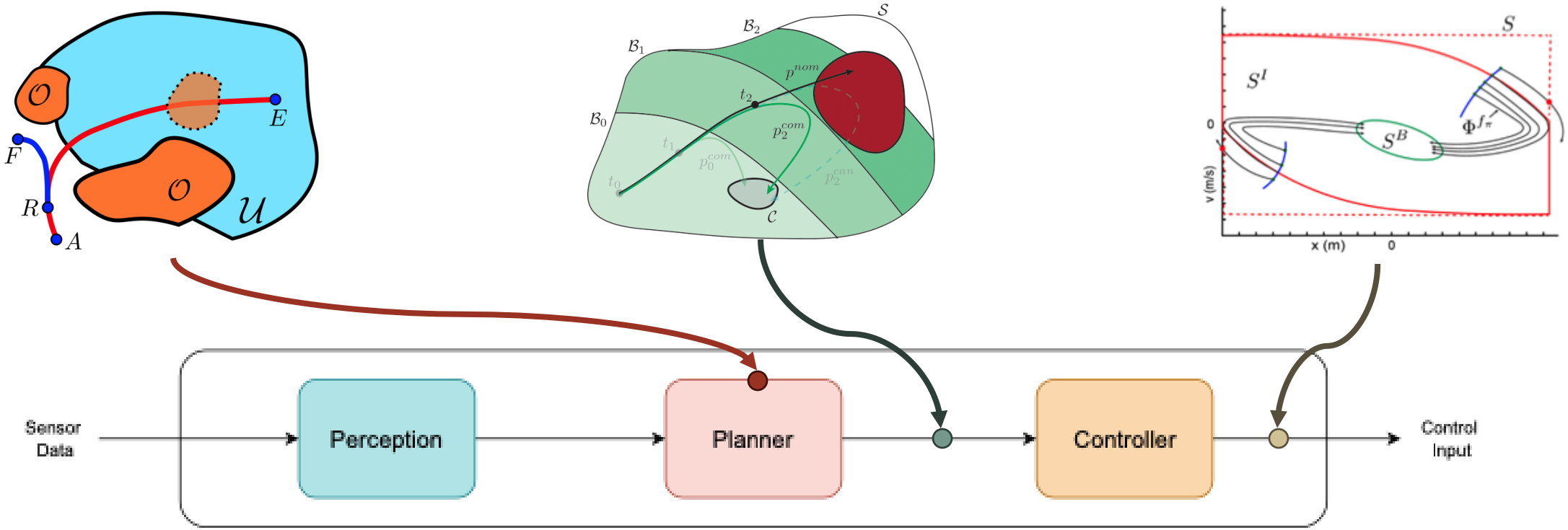
Forward propagates the backup controller. Mixes nominal and backup based on how close backup trajectory gets to unsafety.

Only checks safety of backup trajectory. Always mixing means nominal is never executed (even if it were safe).

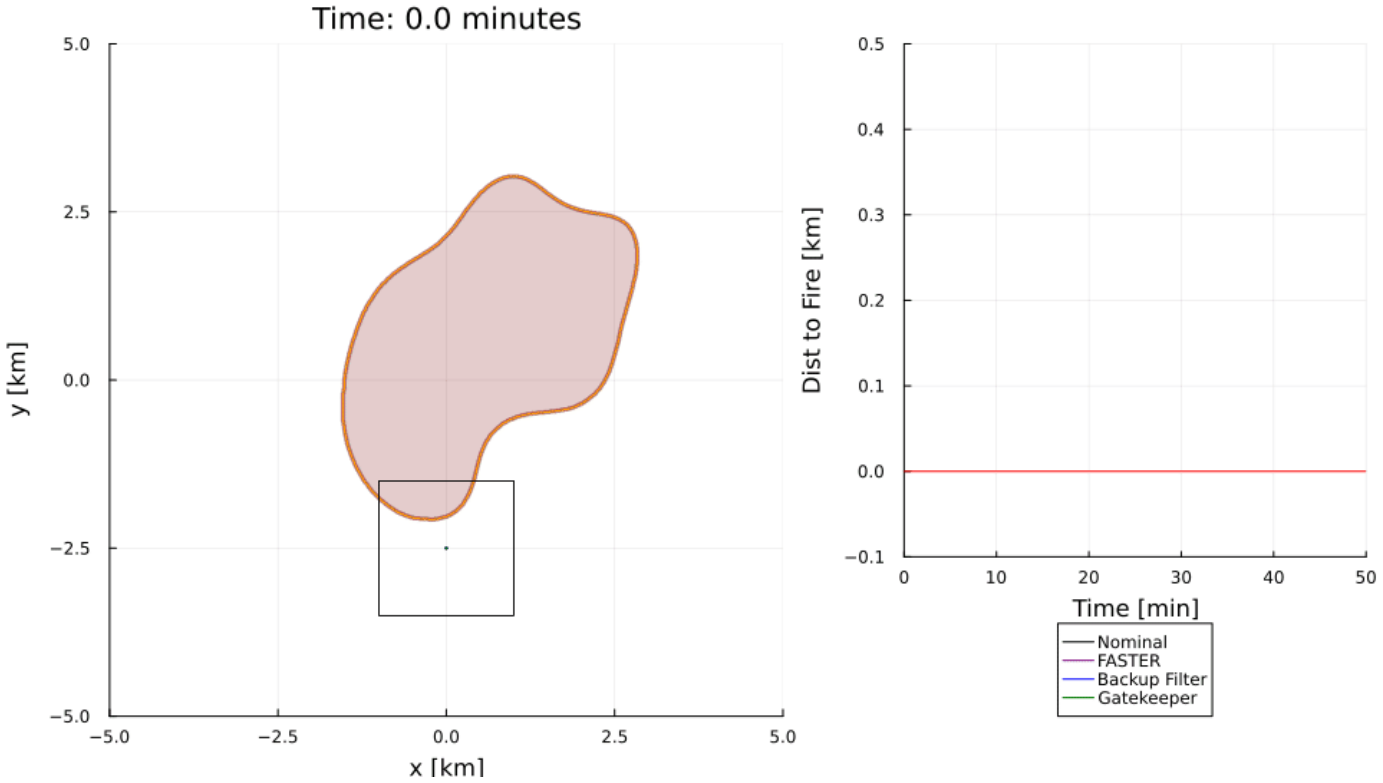


related approaches:

FASTER	`gatekeepeer`	Backup filters
Tordesillas et al, TRO 2021.	Agrawal et al, TRO 2024.	Singletary et al, RAL 2022
Use MPC to solve for both nominal trajectory and a backup trajectory simultaneously.	Track the nominal for as long as possible, before switching to backup. Always track the last committed trajectory.	Forward propagates the backup controller. Mixes nominal and backup based on how close backup trajectory gets to unsafety.



gatekeeper results:

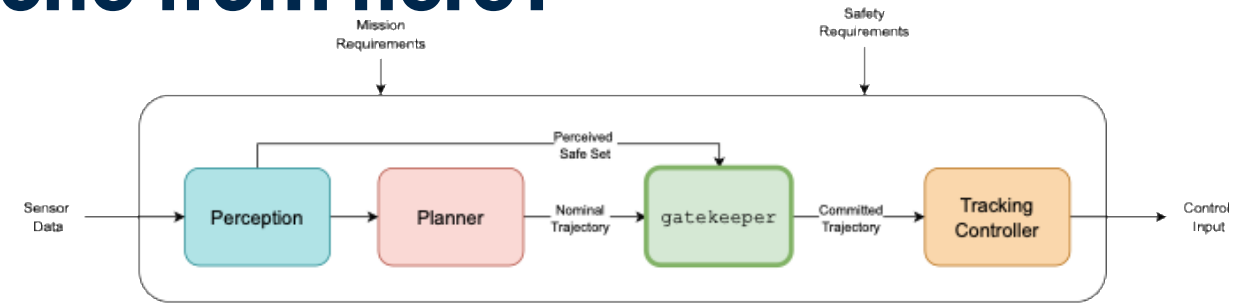


	Distance to Fire [km]			Velocity [m/s]		Comp. time [ms]	
	Minimum	Mean	Std.	Mean	Std.	Median	IQR
Target	≥ 0	0.100	-	15.0	-	-	-
Nominal Planner	-0.032	0.098	0.032	15.14	0.73	27.32	4.37
FASTER [14]	0.040	0.101	0.030	12.60	2.08	78.50	20.64
Backup Filters [15]	0.081	0.240	0.054	10.11	3.52	0.87*	0.05
Gatekeeper (proposed)	0.049	0.108	0.034	14.91	1.35	3.39	0.11

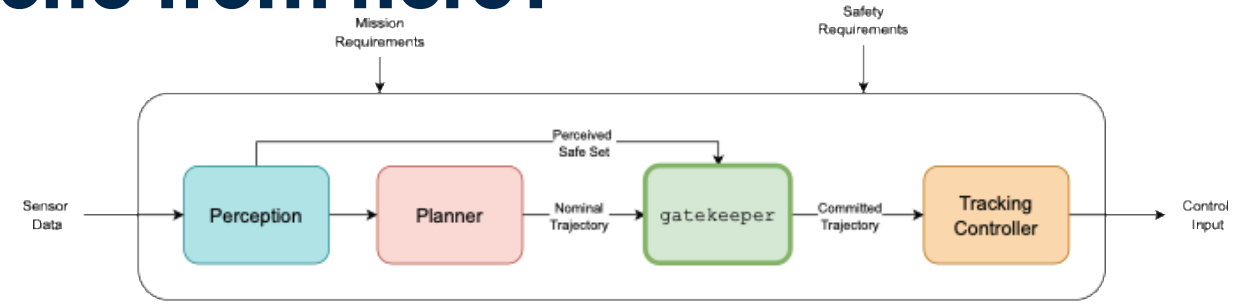
- gatekeeper stays very close to the boundary, but stays safe
- Compared to FASTER:
  - Lower compute time
    3 ms vs 78 ms
- Compared to Backup Filters:
  - Closer to fire perimeter
  - Travels at higher speeds

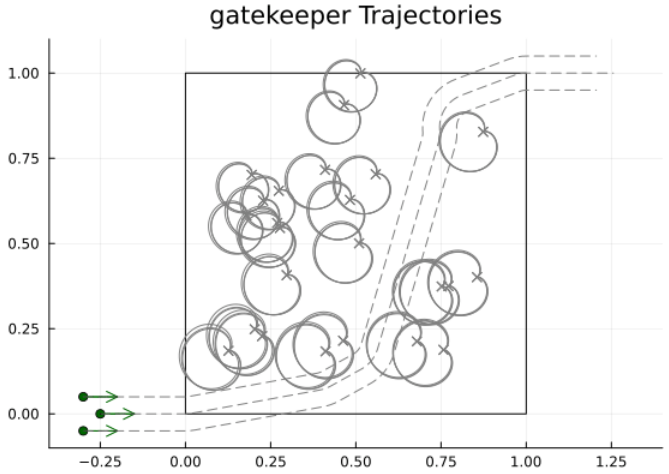
gatekeeper experiments

where have we gone from here?



where have we gone from here?

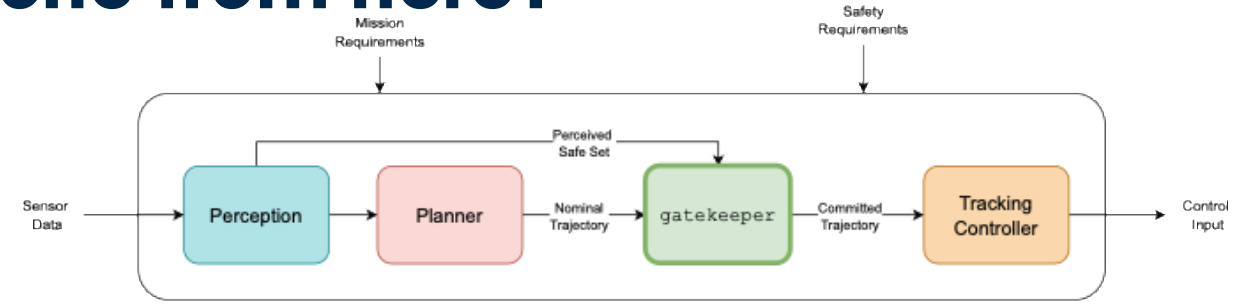


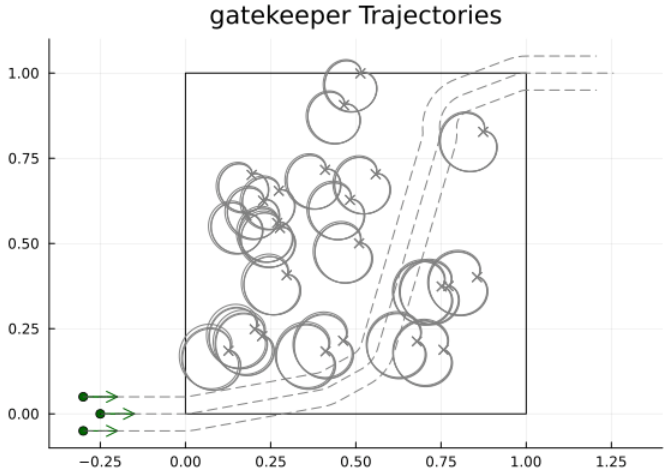


1. Dev + AFRL: Multiagent Wez Avoidance

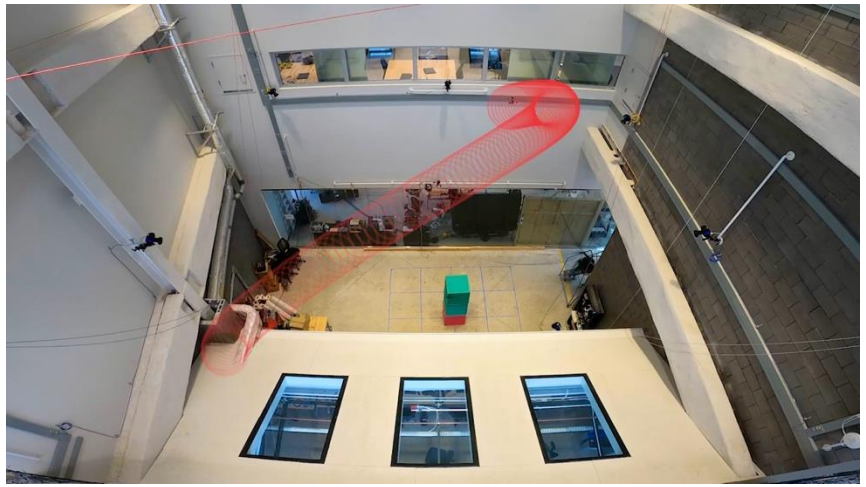
TopGun2

where have we gone from here?





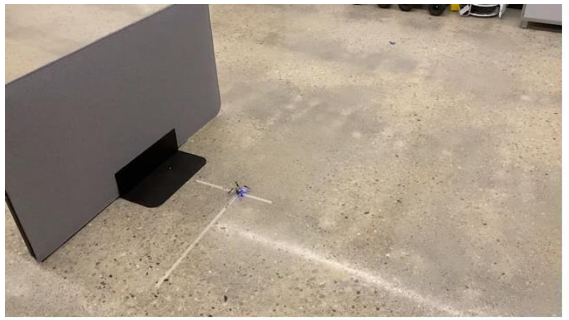
1. Dev + AFRL: Multiagent Wez Avoidance

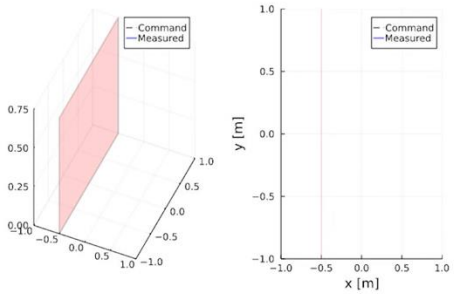


2. Kaleb: Persistent exploration algorithms

3. Daniel: Budget constrained planning

closing the loop on safety guarantees:

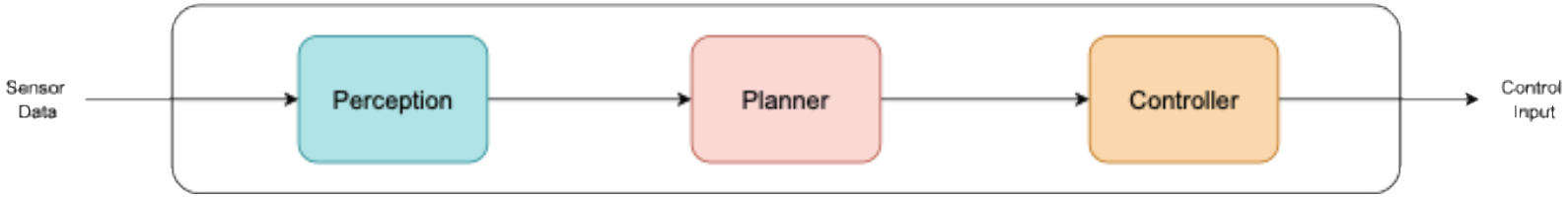




Baseline CBF Controller

Certified Perception:
Modified mapping algorithm to be robust to visual odometry drift.

Observer-Controllers:
Devised CBF-QP controllers to handle state estimation uncertainty.



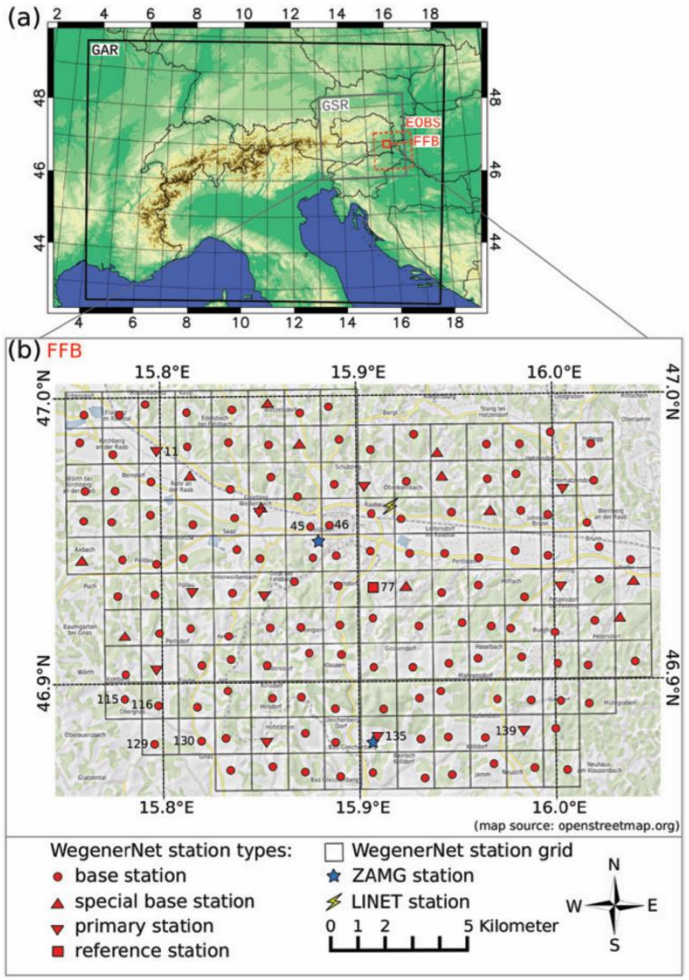
two main questions



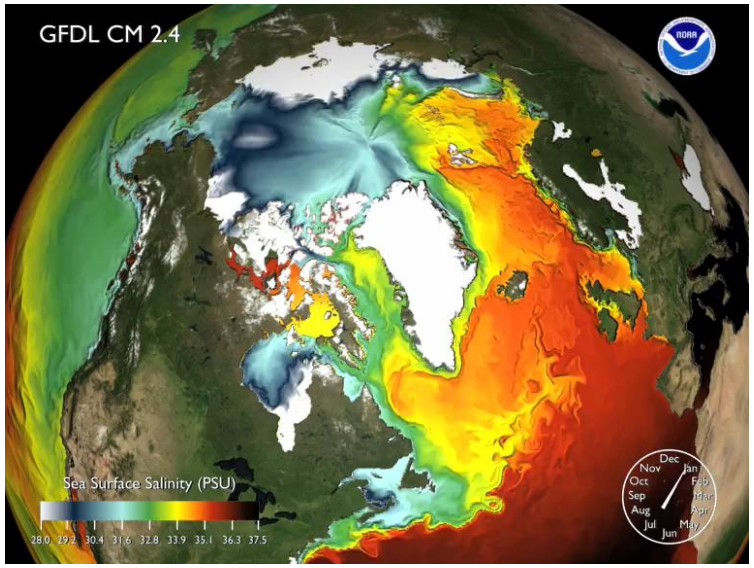

How can we **ensure** safety constraints are not violated during operation?

What are the limits of the information gathering ability of robots?

the information gathering problem

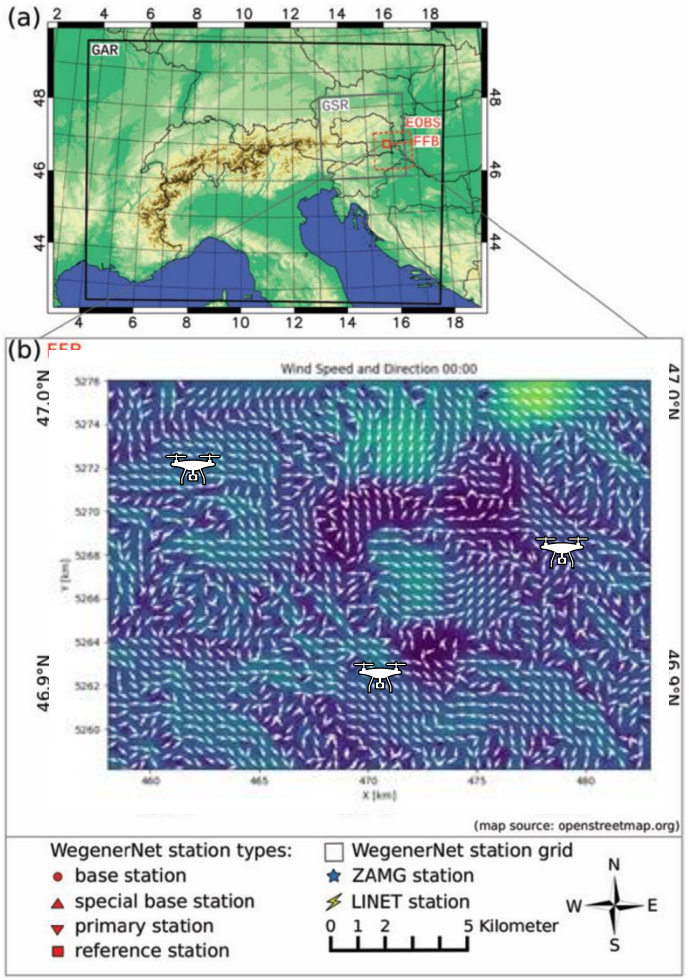


Southeast Austria weather data
WegenerNet [1]



[1] Kirchengast et al. 2013

the information gathering problem



Southeast Austria weather data
WegenerNet [1]

Say we have a team of robots exploring this environment to collect information.

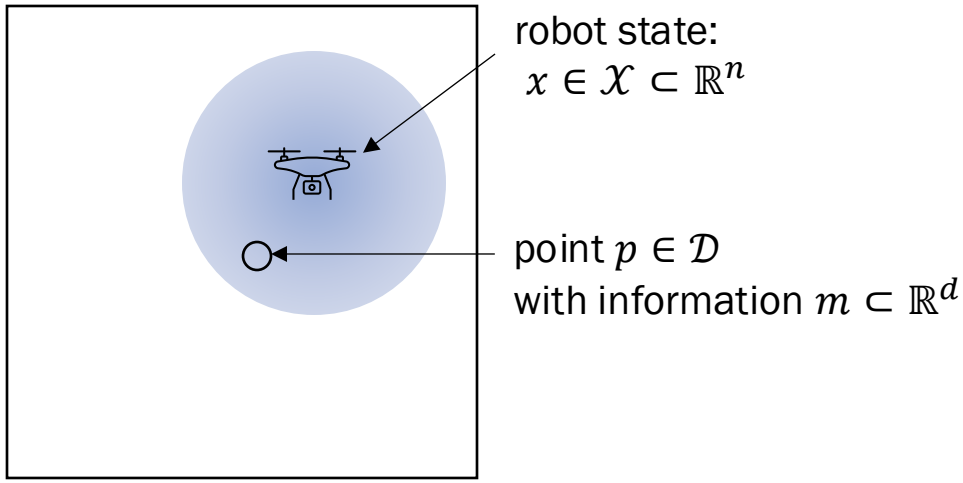
How do we control the robots to collect the maximum quality of information efficiently?

What does efficiently mean?
What does “quality of information” mean?

Can the robots collect the information in the first place?
(especially since the information itself is changing and the robots are moving)
What are fundamental limits of information gathering?

[1] Kirchengast et al. 2013

quantifying information



Given a dynamical system, e.g.

$$\dot{x} = f(x, u) \qquad x \in \mathcal{X} \subset \mathbb{R}^n, u \in \mathcal{U} \subset \mathbb{R}^m$$

a spatio-temporal environment model, e.g.

$$\dot{m} = w(t) \qquad w(t) \sim \mathcal{N}(0, Q)$$

and a sensing model, e.g.:

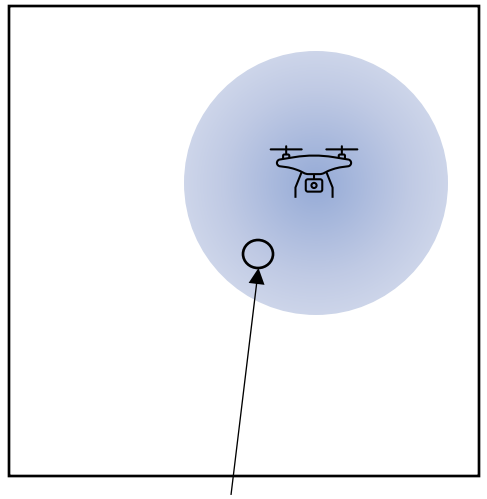
$$y = C(x)m + v(t) \qquad v(t) \sim \mathcal{N}(0, R(x))$$

robot state affects ability to measure m

design a controller $u : [0, T] \rightarrow \mathcal{U}$ to maximize quality of information collected

We need a definition for “quality of information collected”

quantifying information: clarity



a point p has information $m \in \mathbb{R}^d$, modeled as a random variable

we need to quantify the quality of the information possessed, i.e., level of uncertainty about m

Differential Entropy:

$$h[m] = \mathbb{E}[-\log f(m)]$$

probability density function

But:

$$h[m] \in [-\infty, \infty]$$

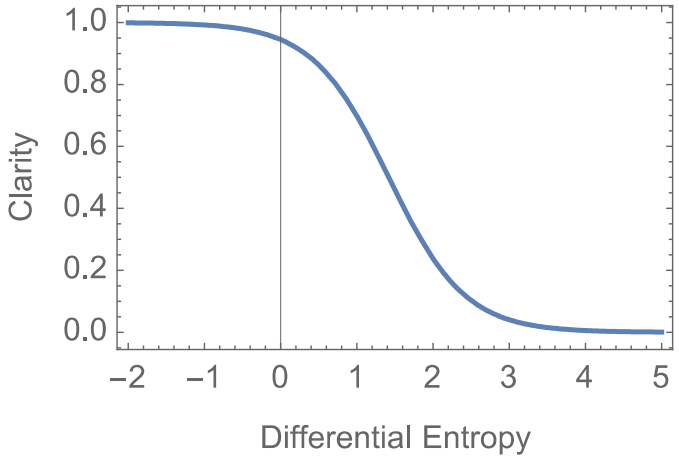
can be negative,
perfect information as $h[m] \to -\infty$

Def: The **clarity** of a d-dimensional continuous random variable m with differential entropy $h[m]$ is

$$q[m] = \left(1 + \frac{e^{2h[m]}}{(2\pi e)^d}\right)^{-1}$$

For $X \sim \mathcal{N}(\mu, \Sigma)$,

$$q[X] = \frac{1}{1 + \det \Sigma}$$



Kalman filtering and clarity

Given a dynamical system, e.g.

$$\dot{x} = f(x, u) \qquad x \in \mathcal{X} \subset \mathbb{R}^n, u \in \mathcal{U} \subset \mathbb{R}^m$$

a spatiotemporal environment model, e.g.

$$\dot{m} = w(t) \qquad w(t) \sim \mathcal{N}(0, Q)$$

and a sensing model, e.g.:

$$y = C(x)m + v(t) \qquad v(t) \sim \mathcal{N}(0, R(x))$$

Using the Kalman Filter to assimilate measurements: estimate has dynamics

$$\dot{\mu} = PC(x)R(x)^{-1}(y - C(x)\mu)$$
$$\dot{P} = Q - \frac{C(x)^2}{R(x)}P^2$$

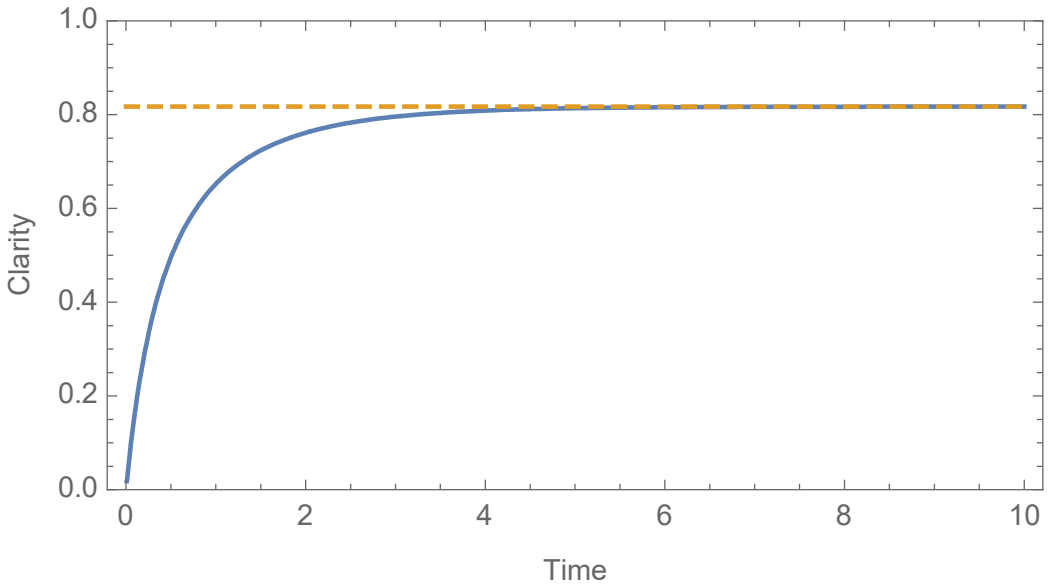
We can derive the clarity dynamics:

Since $q = 1/(1 + P)$, we have

$$\dot{q} = \boxed{\frac{C(x)^2}{R(x)}(1 - q)^2} - \boxed{Qq^2}$$

measurements increase clarity

stochasticity decreases clarity



Kalman filtering and clarity

Given a dynamical system, e.g.

$$\dot{x} = f(x, u) \qquad x \in \mathcal{X} \subset \mathbb{R}^n, u \in \mathcal{U} \subset \mathbb{R}^m$$

a spatiotemporal environment model, e.g.

$$\dot{m} = w(t) \qquad w(t) \sim \mathcal{N}(0, Q)$$

and a sensing model, e.g.:

$$y = C(x)m + v(t) \qquad v(t) \sim \mathcal{N}(0, R(x))$$

Using the Kalman Filter to assimilate measurements: estimate has dynamics

$$\dot{\mu} = PC(x)R(x)^{-1}(y - C(x)\mu)$$
$$\dot{P} = Q - \frac{C(x)^2}{R(x)}P^2$$

[1] B. Haydon, et al. CDC 2021.
[2] Panagou, et al. TCNS 2016.
[3] Bentz, et al. Automatica 2019.

We can derive the clarity dynamics:

Since $q = 1/(1 + P)$, we have

$$\dot{q} = \boxed{\frac{C(x)^2}{R(x)}(1 - q)^2} - \boxed{Qq^2}$$

measurements increase clarity

stochasticity decreases clarity

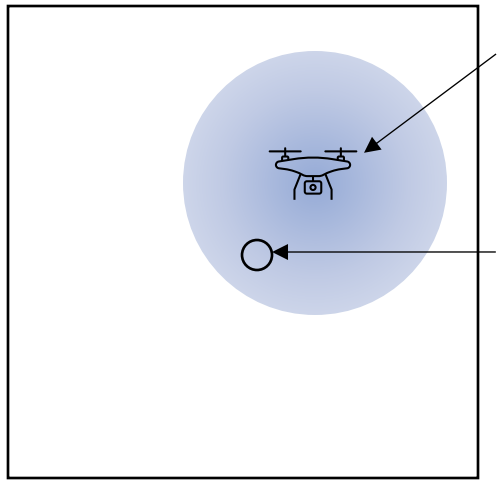
Compare to Coverage Control [1-3]:

$$\dot{q} = \boxed{S(x)(1 - q)} - \boxed{\alpha q}$$

where S, α are tuned heuristically.

Clarity dynamics capture the information collection mechanism based on the env. and sensing model.

maximizing information (i.e., clarity)



robot state:
$x \in \mathcal{X} \subset \mathbb{R}^n$

point $p \in \mathcal{D}$
with information $m \subset \mathbb{R}^d$

Given a dynamical system: $\dot{x} = f(x, u)$

an environment model: $\dot{m} = w(t)$

and a sensing model: $y = C(x)m + v(t)$

The clarity dynamics are $\dot{q} = \frac{C(x)^2}{R(x)}(1-q)^2 - Qq^2$

Perceivability

Construct an augmented system

$$\dot{x} = f(x, u)$$
$$\dot{q} = \frac{C(x)^2}{R(x)}(1-q)^2 - Qq^2$$

Solve an optimal control problem to reach target clarity.

Dynamic Coverage

Define m for each point in a spatial domain

Quantify the rate of increase of clarity based on robot locations

Design feedback controllers to maximize rate of increase of clarity

perceivability

Definition:
A quantity $m \in \mathbb{R}$ is **perceivable** by the system
to a level $q^* \in [0, 1]$ at time T
from an initial condition (x_0, q_0)
if there exists a controller $\pi : [t_0, t_0 + T] \to \mathcal{U}$
s.t. $q(t_0 + T) \geq q^*$.

Theorem:
Define

$$V(t_0, x_0, q_0) = \max_{\pi \in \mathcal{L}(\mathcal{U})} \quad q(t_0 + T)$$
$$\text{s.t.} \quad \dot{x} = f(x, u), \quad x(t_0) = x_0,$$
$$\dot{q} = \dot{q}(x, u, q), \quad q(t_0) = q_0.$$

i.e., the maximum clarity reachable.

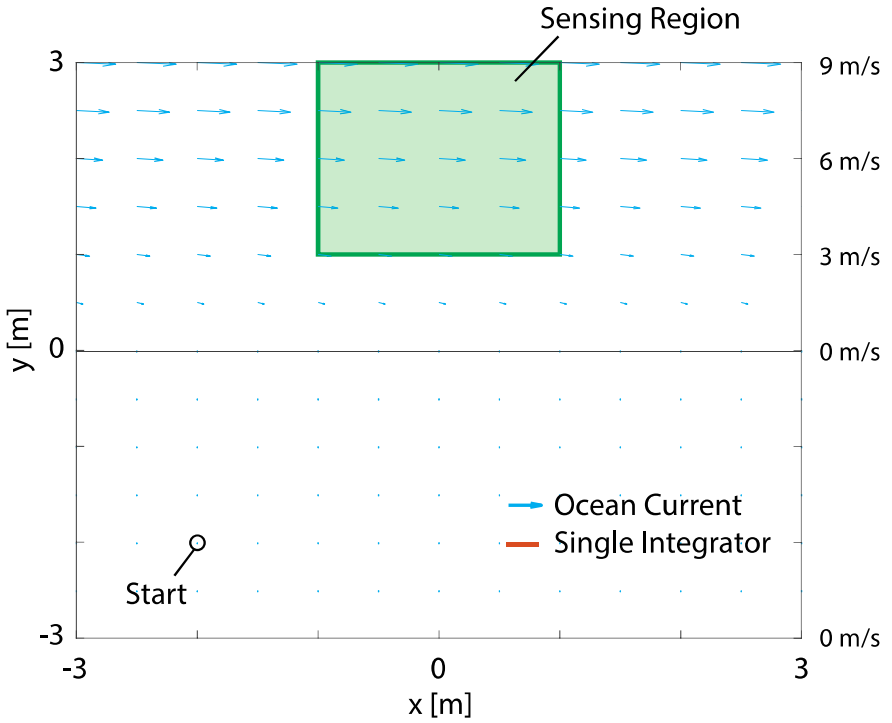
m is **perceivable** if $V(t_0, x_0, q_0) \geq q^*$.

Perceivability depends on the
- environment model
- the robot's sensing capabilities **and**
- robot's actuation capabilities

Perceivability is an optimal control problem that can be solved using reachability analysis (i.e., the HJB equations)

Once we have solved for V, it also gives the optimal controller as a feedback controller.

ex: salinity measurements

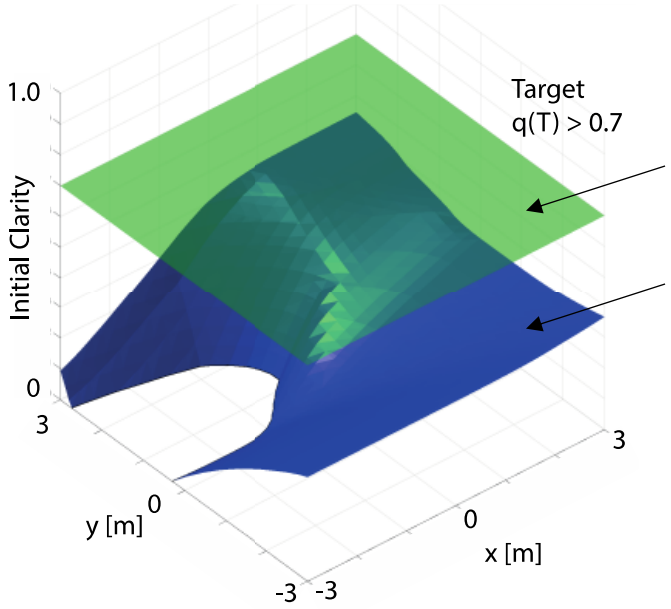


$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}u_1\\ u_2\end{bmatrix} + \begin{bmatrix}w_1(x)\\ w_2(x)\end{bmatrix} \qquad u_1, u_2 \in [-2, 2] \text{ m/s}$$

$$\dot{q} = \frac{C(x)^2}{R(x)}(1-q)^2 - Qq^2$$

Optimal strategy requires going through the sensing region many times, due to limited max speed

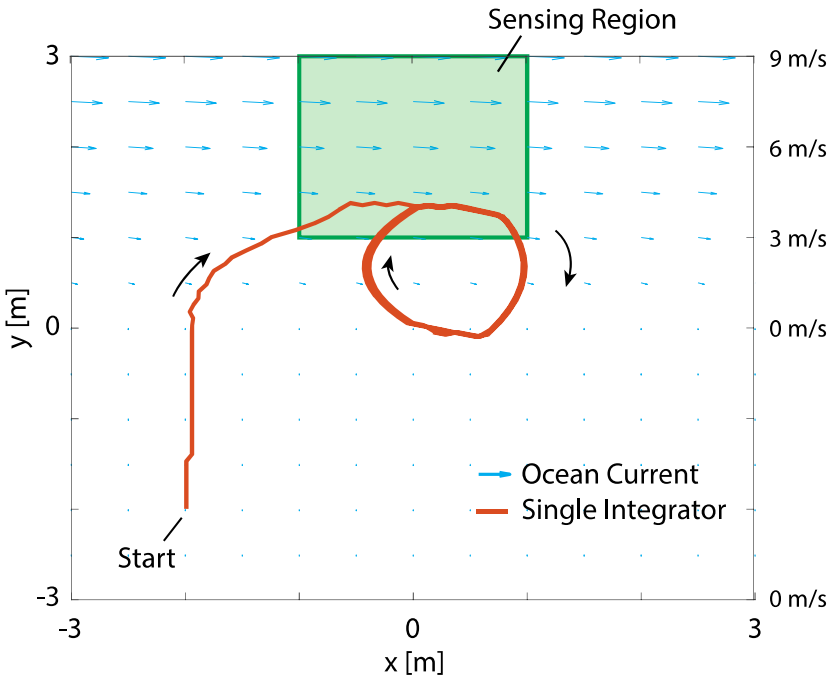
Solve the HJB PDE, and determine the perceivability domain



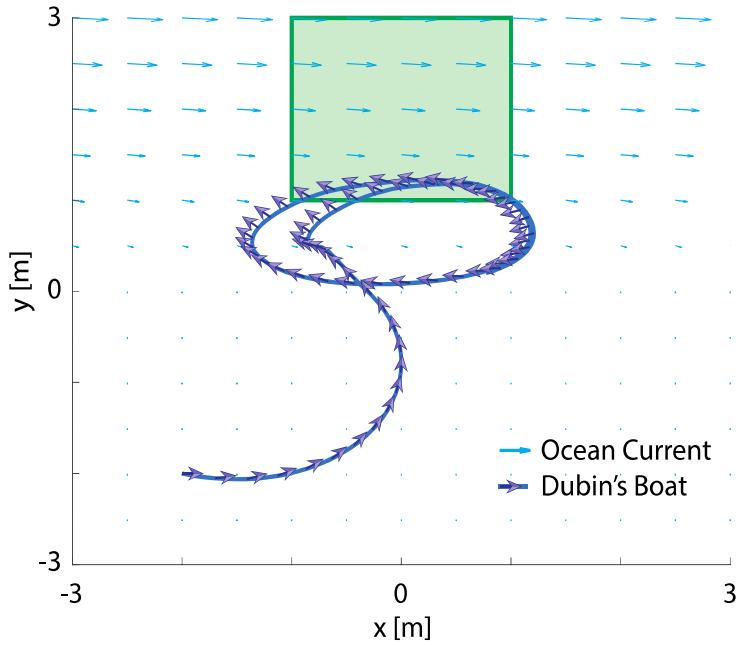
above green = target domain

above blue = perceivability domain

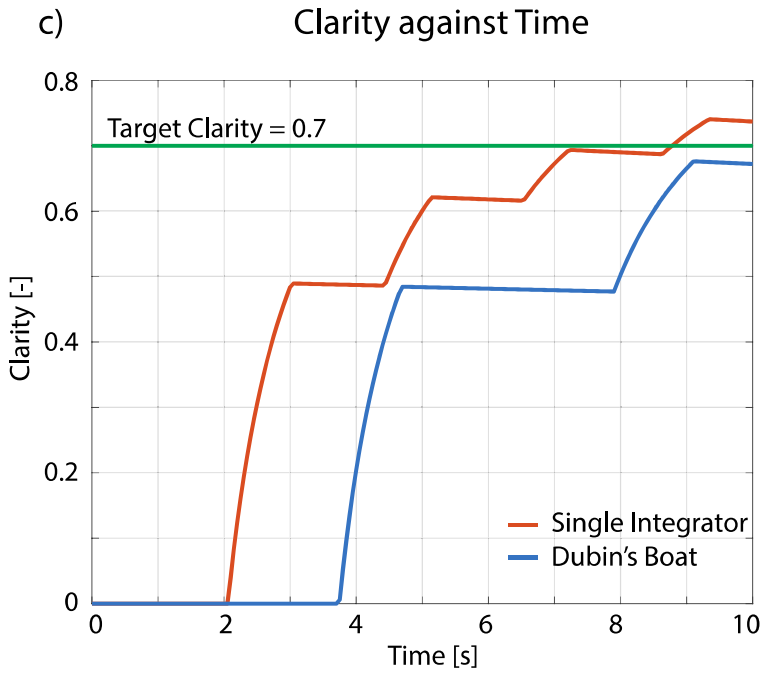
ex: salinity measurements



Single Integrator

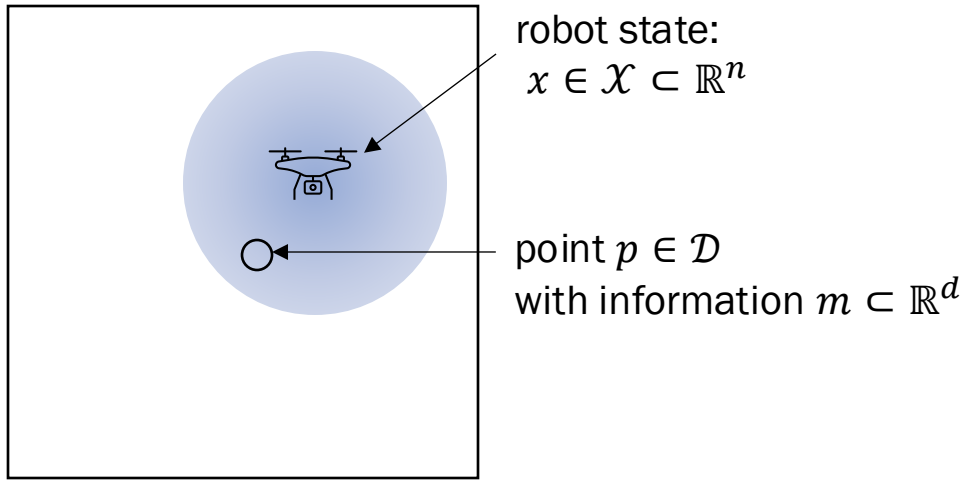


Dubin's Boat



The perceivability of the environment depends on the robot's actuation capabilities.

maximizing information



Given a dynamical system: $\dot{x} = f(x, u)$

an environment model: $\dot{m} = w(t)$

and a sensing model: $y = C(x)m + v(t)$

The clarity dynamics are $\dot{q} = \frac{C(x)^2}{R(x)}(1-q)^2 - Qq^2$

Perceivability

Construct an augmented system

$$\dot{x} = f(x, u)$$
$$\dot{q} = \frac{C(x)^2}{R(x)}(1-q)^2 - Qq^2$$

Solve an optimal control problem to reach target clarity.

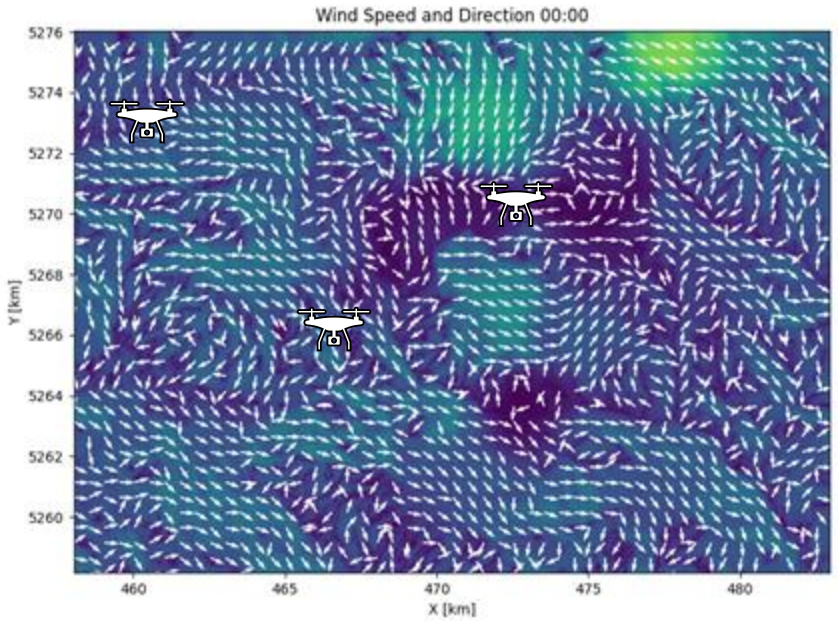
Dynamic Coverage

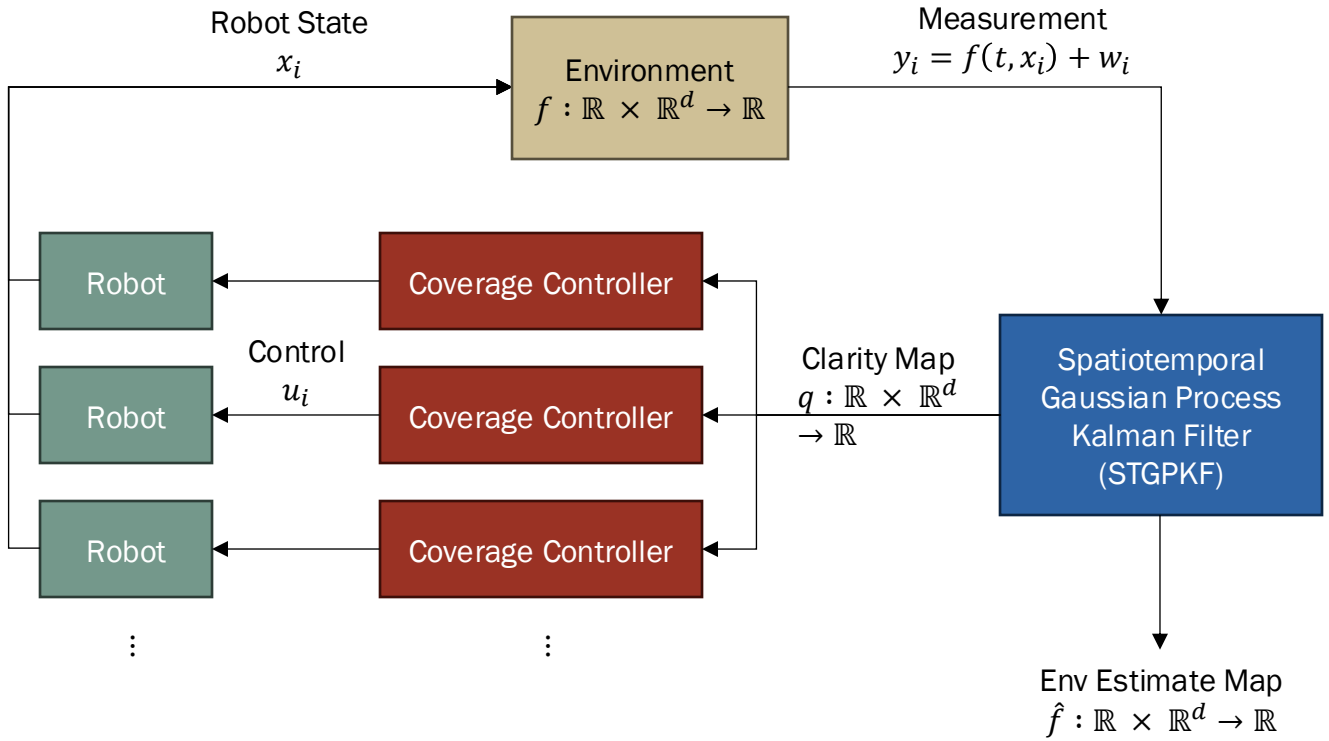
Define m for each point in a spatial domain

Quantify the rate of increase of clarity based on robot locations

Design feedback controllers to maximize rate of increase of clarity

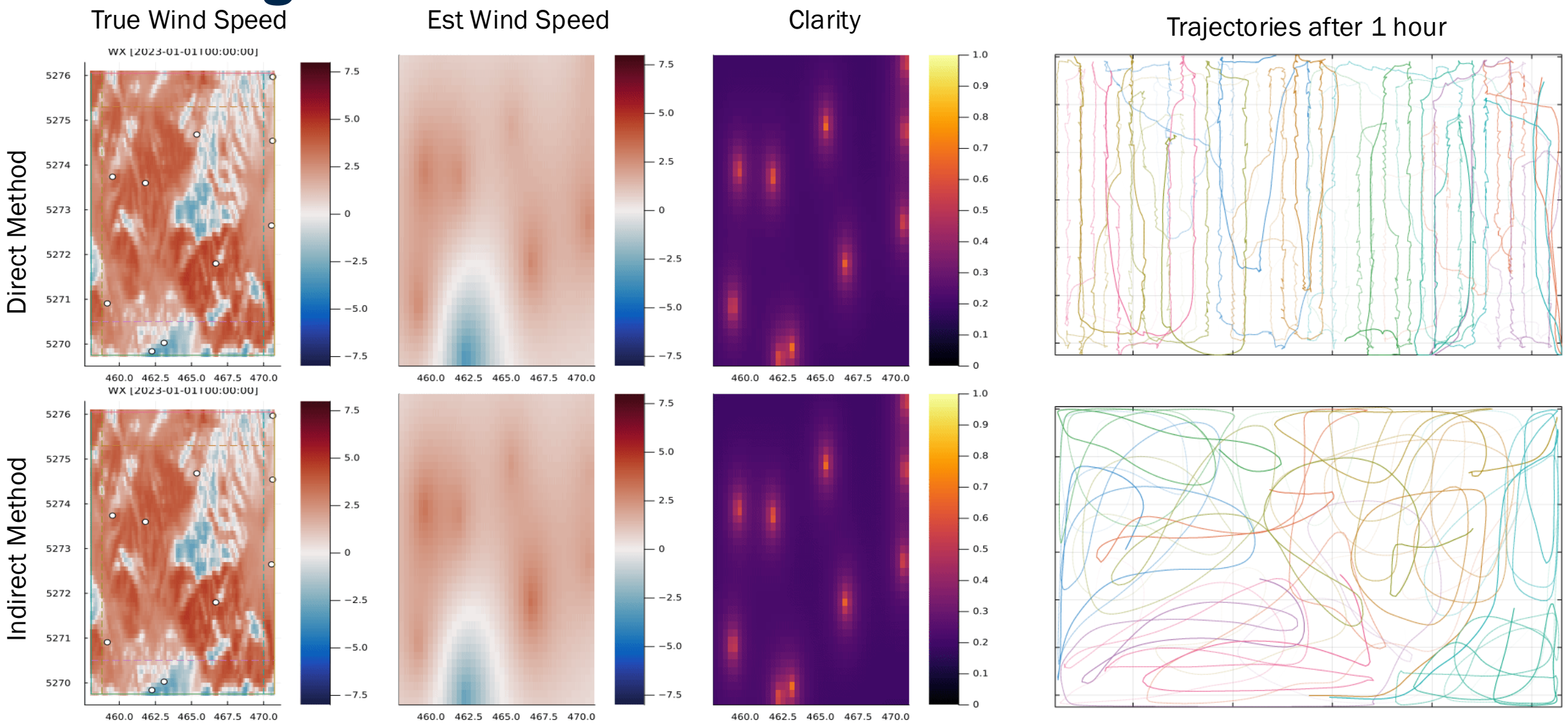
multiagent dynamic coverage





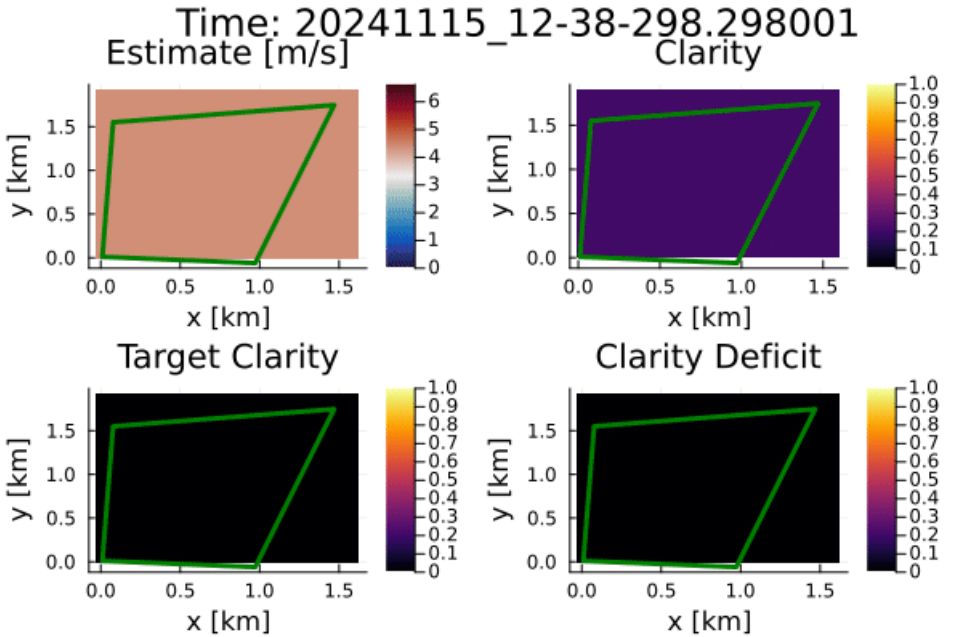
Model information to be collected as a GP
Convert GP into SDE model
Estimate information using a spatiotemporal KF
Quantify the clarity gain rate given robot locations
Design coverage controllers to maximize clarity
(two methods proposed)

two coverage controllers



where have we gone from here?

Kavin and Kaleb have run real-world experiments of this framework.
They have extended it to handle solar and battery constraints, the boat's dynamics and target clarities that depend on the information collected so far.



(real experimental data!)

Kaleb is extending the method to handle non-stationary GPs.
Kavin is extending these ideas to try to quantify the information value of energy.

conclusions




How can we **ensure** safety constraints are not violated during operation?

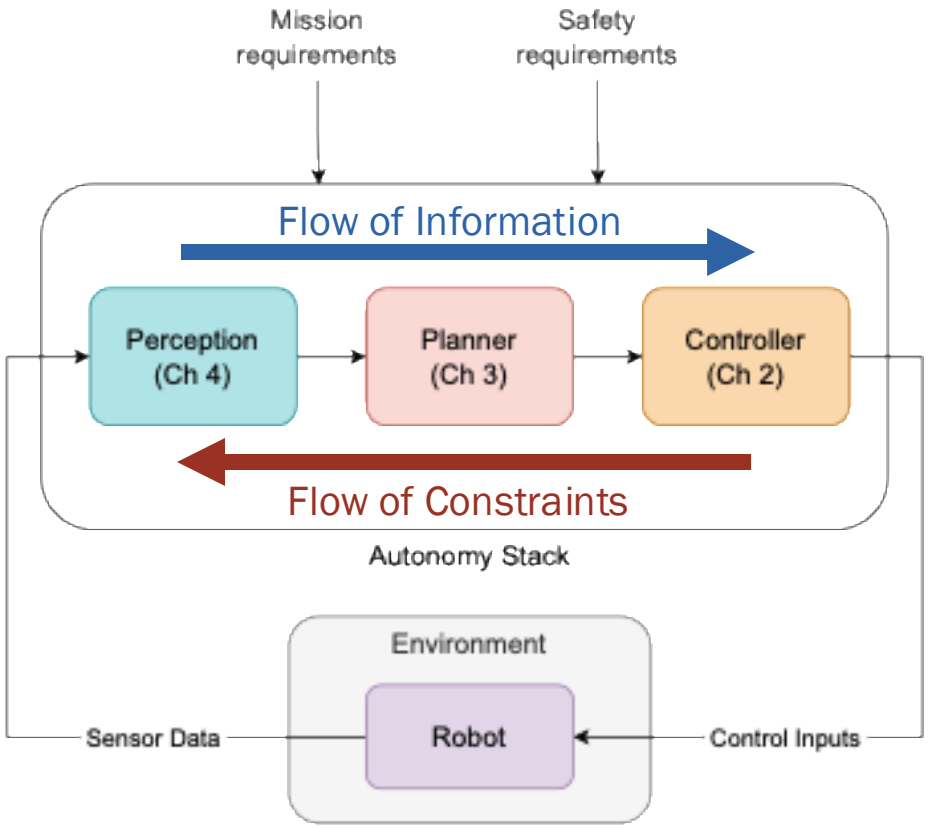
What are the limits of the information gathering ability of robots?

conclusions

- Why is designing safe autonomy hard?

"Information" flows left-to-right

Each module computes quantities used by downstream modules

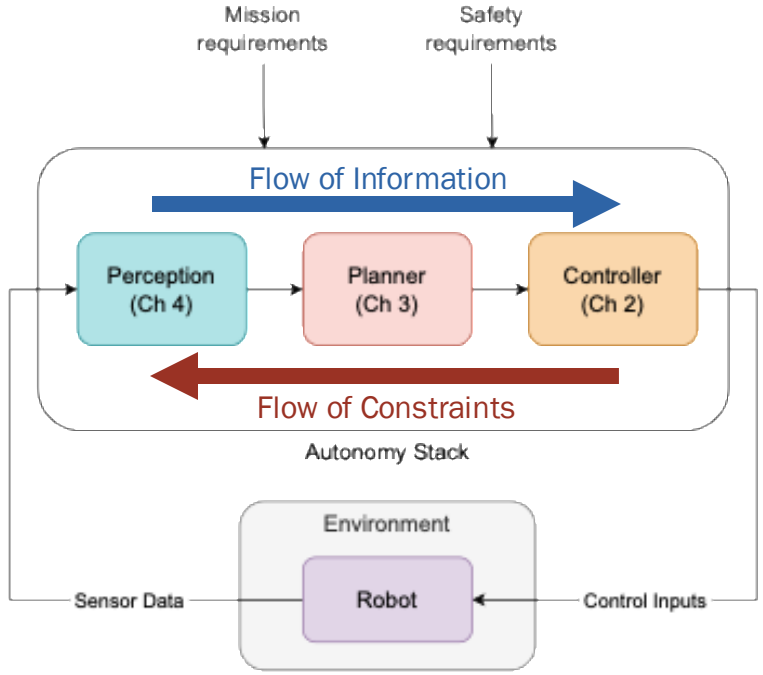


"Constraints" flow right-to-left

To guarantee safety the downstream modules impose constraints on the upstream modules

conclusions

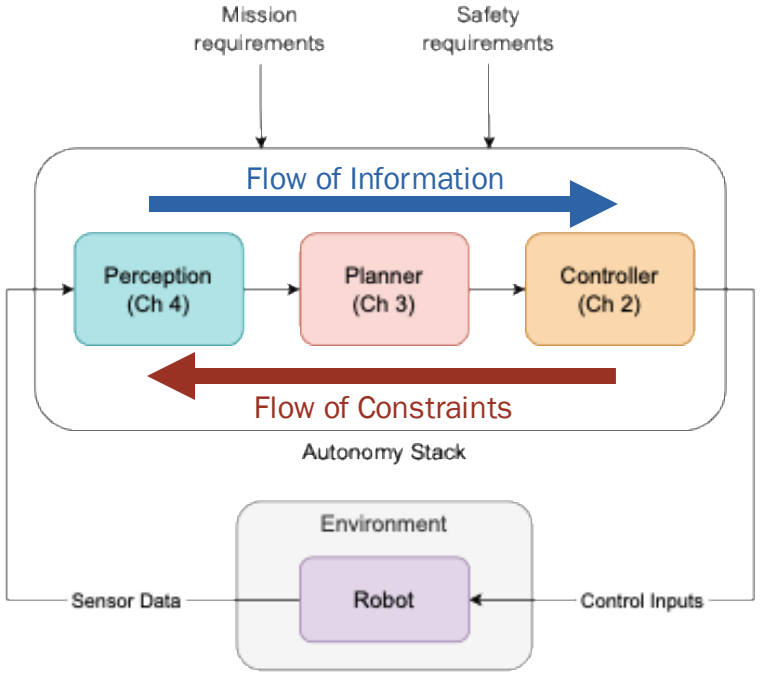
- Why is designing safe autonomy hard?
  - Flow of “safety constraints” is not well understood
    Often “tacked on” rather than a core part of the design of autonomous systems
  - Resilience vs robustness
  - “Mission requirements” is often implemented for a specific situation, and general mathematical frameworks are cumbersome/complicated



conclusions

- Why is designing safe autonomy hard?
  - Flow of “safety constraints” is not well understood
    Often “tacked on” rather than a core part of the design of autonomous systems
  - Resilience vs robustness
  - “Mission requirements” is often implemented for a specific situation, and general mathematical frameworks are cumbersome/complicated

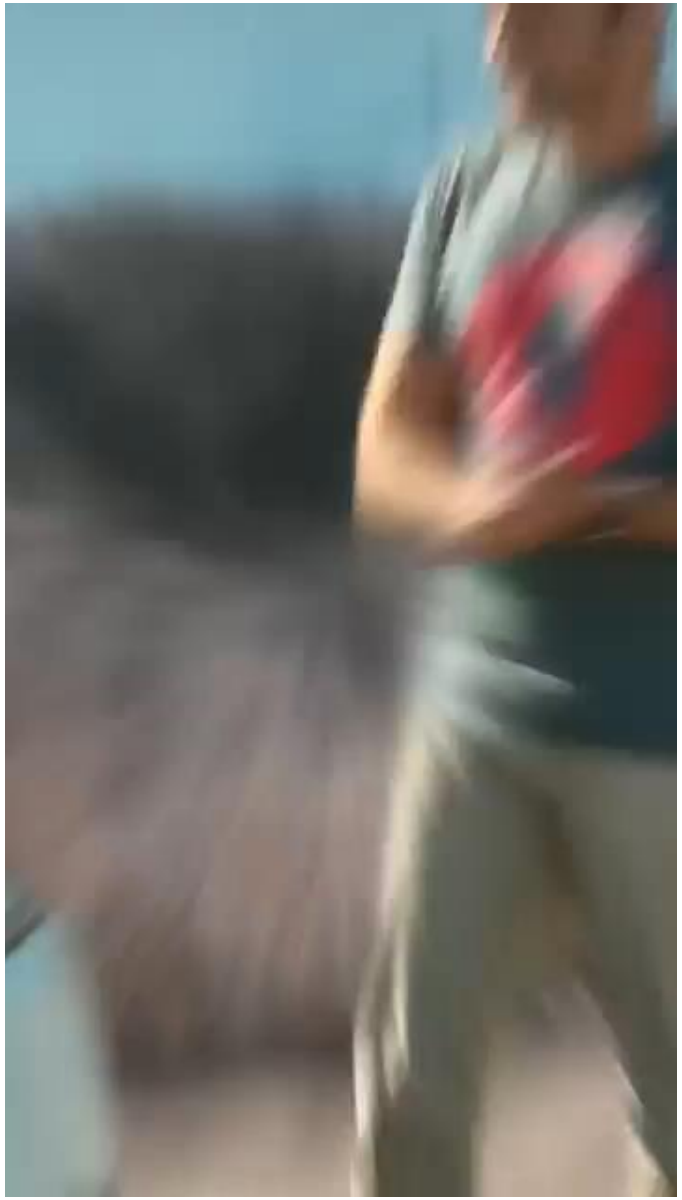
- What will we need going forward?
  - How to handle stochasticity?



conclusions

- Why is designing safe autonomy hard?
  - Flow of “safety constraints” is not well understood
    Often “tacked on” rather than a core part of the design of autonomous systems
  - Resilience vs robustness
  - “Mission requirements” is often implemented for a specific situation, and general mathematical frameworks are cumbersome/complicated

- What will we need going forward?
  - How to handle stochasticity?
  - Better understanding of the limits of our sensors
  - Better integration of perception with planning/controls

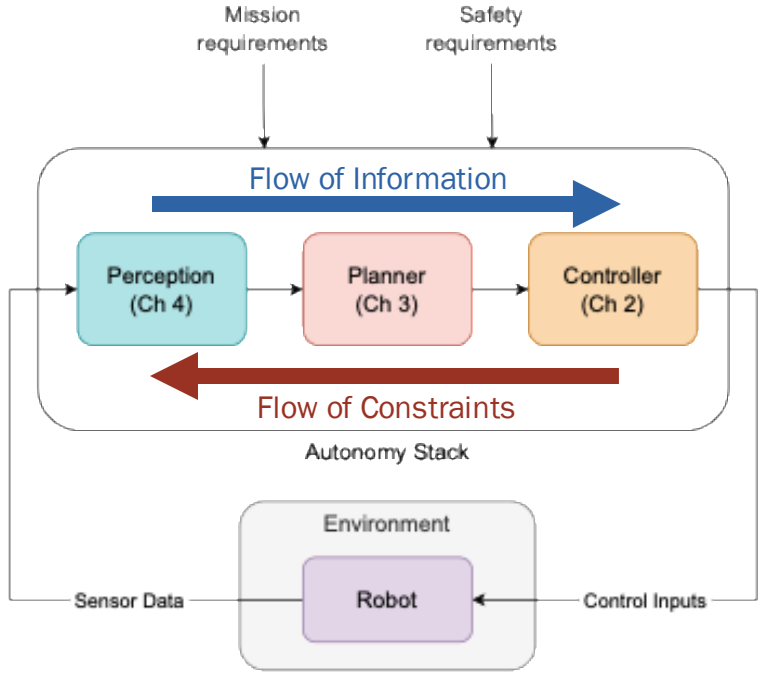


Mark Rober, 2025

conclusions

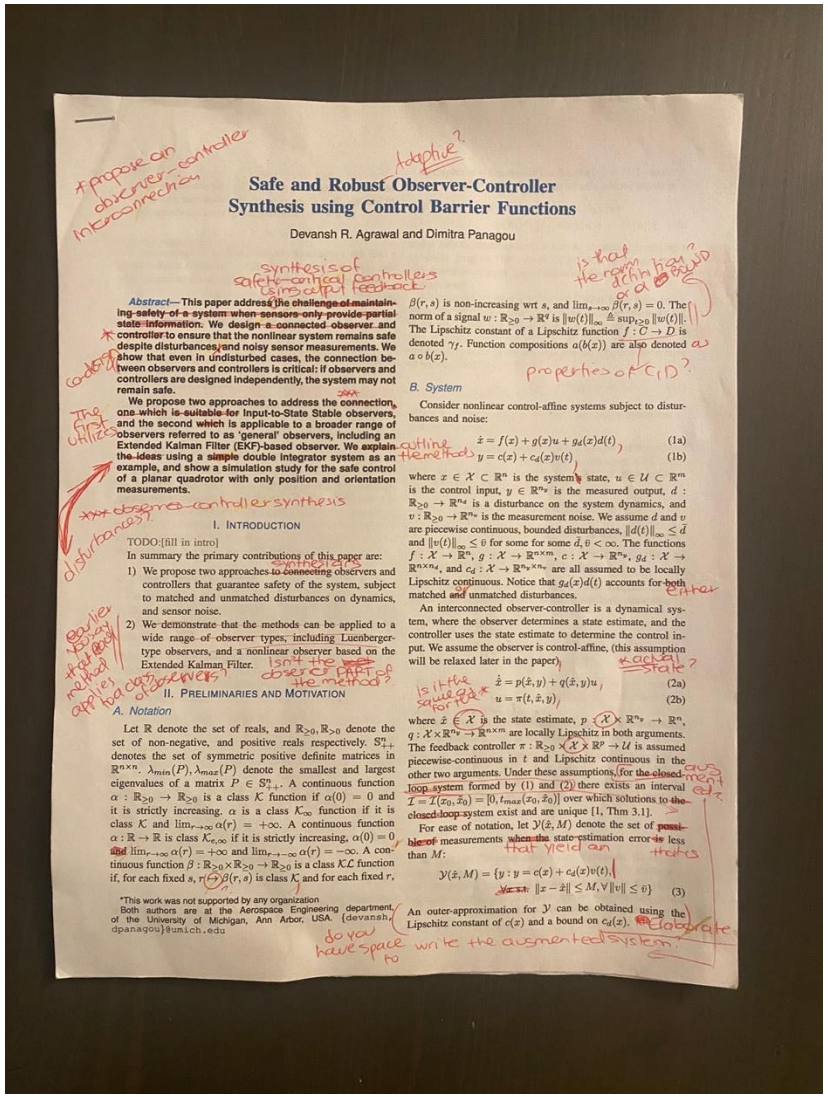
- Why is designing safe autonomy hard?
  - Flow of “safety constraints” is not well understood
    Often “tacked on” rather than a core part of the design of autonomous systems
  - Resilience vs robustness
  - “Mission requirements” is often implemented for a specific situation, and general mathematical frameworks are cumbersome/complicated

- What will we need going forward?
  - How to handle stochasticity?
  - Better understanding of the limits of our sensors
  - Better integration of perception with planning/controls
  - More abstract tools for mission requirements and interactions between multiagent systems



acknowledgements

Prof Dimitra Panagou



Safe and Robust Observer-Controller Synthesis using Control Barrier Functions

Devansh R. Agrawal and Dimitra Panagou

CDC 2021	Jeju Island (but covid)
CDC 2022	Cancun, Mexico
CDC 2023	Singapore
CDC 2024	Milan, Italy
ACC 2021	Atlanta, Georgia
IROS 2023	Detroit, USA
ICRA 2024	Yokohama, Japan
CAAMS 2024	San Diego, USA
CAAMS 2024	Boston, USA
CAAMS 2025	Hawaii, USA
CPS Training	Marblehead, USA

acknowledgements

Prof Dimitra Panagou

Prof Vasileios Tzoumas

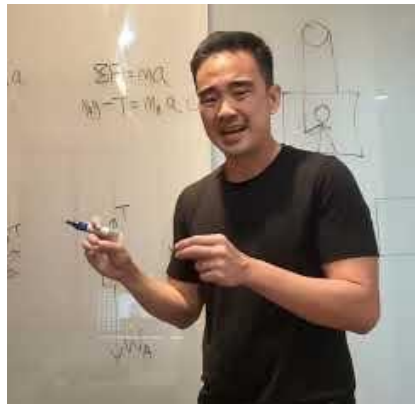
Prof Ilya Kolmanovsky

Prof Necmiye Ozay

Prof Chris Vermillion

Prof Aaron Ames

Prof John Ho



Mr Terence Chiew

Mr Damen Provost

Rotor

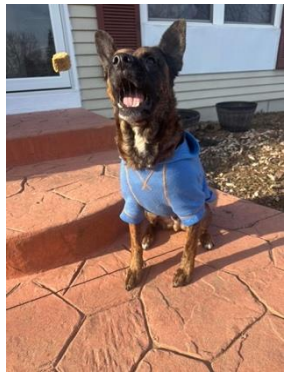
acknowledgements









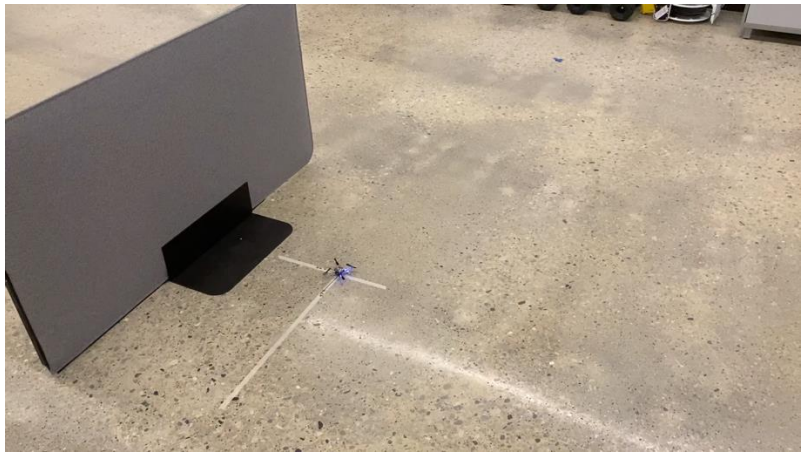





acknowledgements

my lab coffee setup

The L&B Kitchen



Architectures for Safe Autonomy:
Provable Guarantees Across Control, Planning, and Perception

Questions?

Backup Slides

Contributions

Clarity and Perceivability
(LCSS/CDC 2023)

Certifiably-Correct Mapping despite Visual Odometry Drift
(RSS 2025, under review)

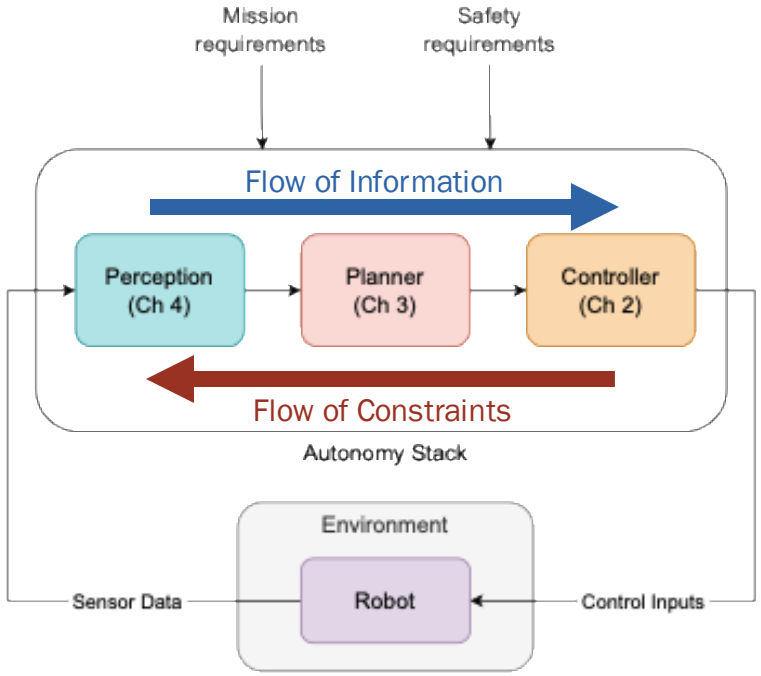
Multirate Planner-Controllers using Differential Flatness
(LCSS/ACC 2022)

Input Constrained Control Barrier Functions
(CDC 2021)

Multiagent Dynamic Coverage with GPs
(CDC 2024)

`gatekeeper`
(TRO 2024, IROS 2023)

Observer-Controller Interconnections
(LCSS/CDC 2022)



Advances in the Theory of CBFs
(Annual Reviews 2024)

Energy-Aware Ergodic Search
(ICRA 2024)

Algorithm

(notation)

dynamics: $\dot{x} = f(x, u)$

(unknown) safe set: $\mathcal{S}(t)$

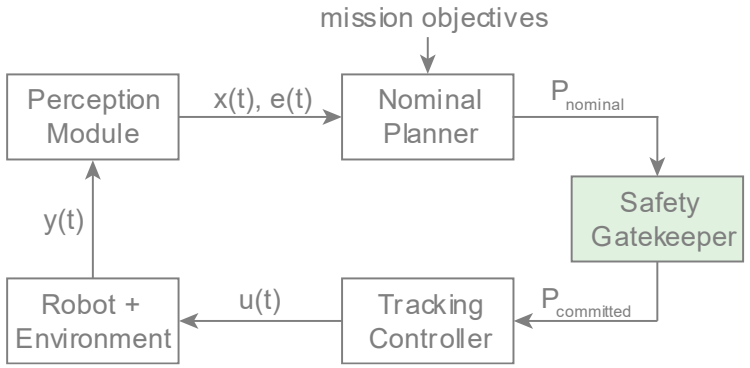
perceived safe set: $\mathcal{B}_k(t)$

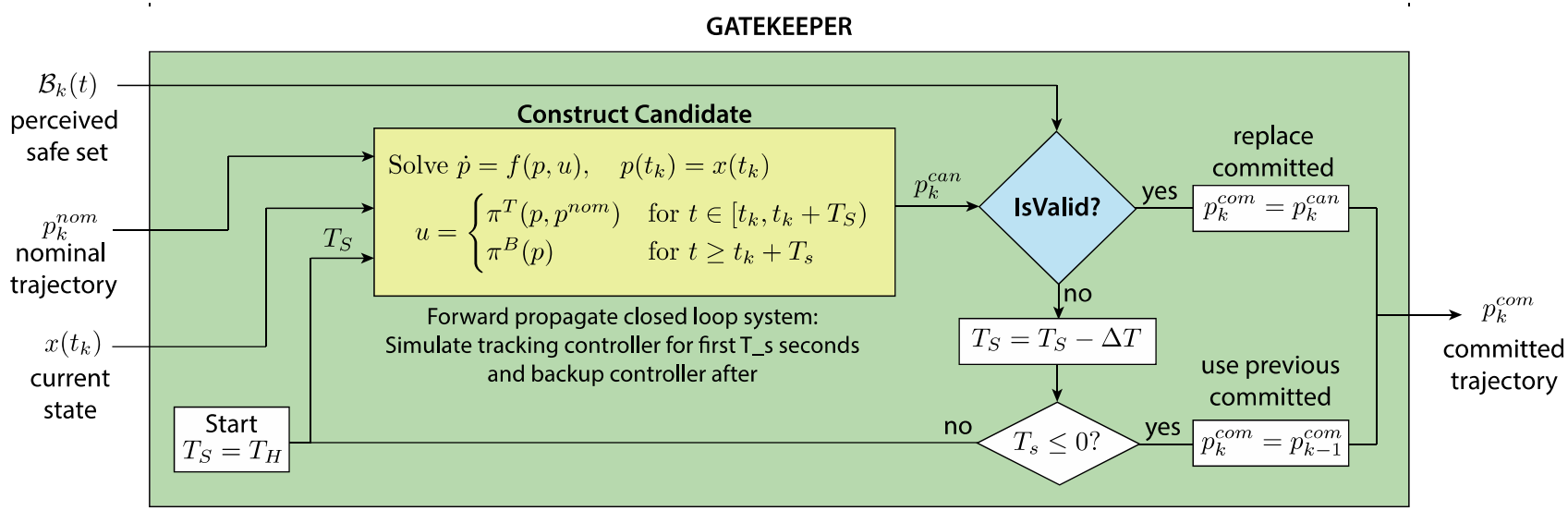
controlled-inv. set: $\mathcal{C}_k(t)$

nominal trajectory: $p_k^{nom}(t)$

candidate trajectory: $p_k^{can}(t)$

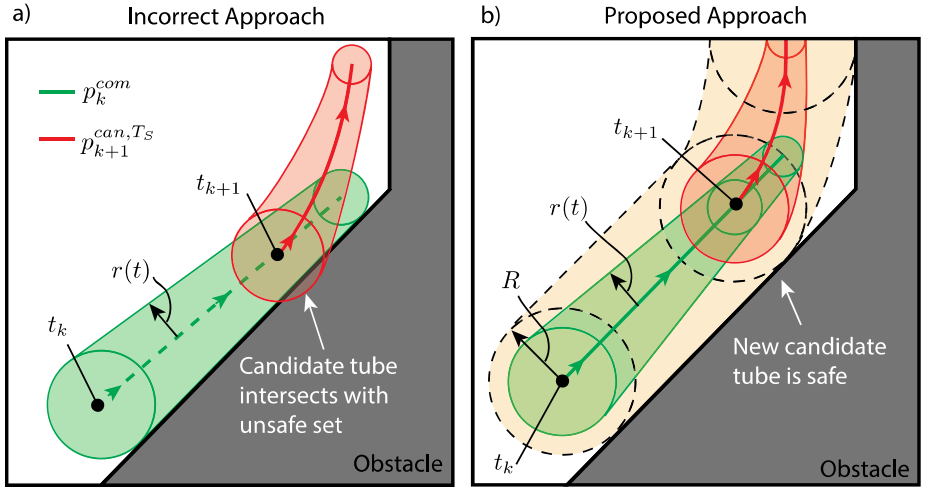
committed trajectory: $p_k^{com}(t)$





See the paper for the full proof and algorithm description.

Small modification to “isValid” needed to robustify against disturbances.



Clarity: useful properties

$q[m] \in [0, 1]$	Clarity is bounded, 'perfect' information as $q[m] \to 1$
$q[m + c] = q[m]$	Clarity is shift-invariant
$q[am] \neq q[m]$	Clarity is *not* scale-invariant
$m \sim \mathcal{N}(\mu, \sigma^2) \implies q[m] = \frac{1}{1+\sigma^2}$	Clarity of Gaussian RV

Clarity lower-bounds the expected estimation error:

Theorem 1. *For any* 1*-D continuous random variable* m *and any* $\hat{m} \in \mathbb{R}$*, the expected estimation error is lower-bounded as*

$$E[(m - \hat{m})^2] \geq \frac{1}{q[m]} - 1$$

with equality if and only if m *is Gaussian and* $\hat{m} = E[m]$.

Perceivability as a reachability problem

Define V as the maximum clarity reachable from (t, x, q):

$$V(t, x(t), q(t)) = \max_{\pi \in \mathcal{L}([t,T],\mathcal{U})} q(T)$$
$$\text{s.t. } \dot{x} = f(x, u),\ \dot{q} = g(x, u)$$

Perceivability as a reachability problem

Define V as the maximum clarity reachable from (t, x, q):

$$V(t, x(t), q(t)) = \max_{\pi \in \mathcal{L}([t,T],\mathcal{U})} q(T)$$
$$\text{s.t. } \dot{x} = f(x, u),\ \dot{q} = g(x, u)$$

By the principle of dynamic programming, for any $\delta > 0$,

$$V(t, x(t), q(t)) = \max_{\pi \in \mathcal{L}([t,t+\delta],\mathcal{U})} V(t + \delta, x(t + \delta), q(t + \delta))$$

Perceivability as a reachability problem

Define V as the maximum clarity reachable from (t, x, q):

$$V(t, x(t), q(t)) = \max_{\pi \in \mathcal{L}([t,T],\mathcal{U})} q(T)$$
$$\text{s.t. } \dot{x} = f(x, u),\ \dot{q} = g(x, u)$$

By the principle of dynamic programming, for any $\delta > 0$,

$$V(t, x(t), q(t)) = \max_{\pi \in \mathcal{L}([t,t+\delta],\mathcal{U})} V(t + \delta, x(t + \delta), q(t + \delta))$$

Using a Taylor expansion about $\delta = 0$, as $\delta \to 0$, we arrive at the HJB PDE

$$\frac{\partial V}{\partial t} + \max_{u \in \mathcal{U}} \left(\frac{\partial V}{\partial x} f(x, u) \right) + \frac{\partial V}{\partial q} g(x, q) = 0,$$
$$V(T, x, q) = q$$

Perceivability as a reachability problem

Theorem. *Let* $V : [0, T] \times \mathcal{X} \times [0, 1] \to \mathbb{R}$ *solve*

$$\frac{\partial V}{\partial t} + \max_{u \in \mathcal{U}} \left(\frac{\partial V}{\partial x} f(x, u) \right) + \frac{\partial V}{\partial q} g(x, q) = 0,$$
$$V(T, x, q) = q \quad \forall x \in \mathcal{X}, q \in [0, 1].$$

Then the (q^*, T)*- perceivability domain of* $m \in \mathbb{R}$ *is*

$$\mathcal{D}(q^*, T) = \left\{ [x_0^T, q]^T : V(0, x_0, q_0) \geq q^* \right\}.$$

Moreover, the optimal controller is

$$\pi(t, x, q) = \text{argmax}_{u \in \mathcal{U}} \left(\frac{\partial V}{\partial x} f(x, u) \right)$$

To summarize:

Given a robot model, env model, the clarity dynamics are well defined.

We can solve the HJB PDE for the value function $V(t, x, q)$

Super-level sets of V determine the perceivability domain

Value function determines the optimal controller.