Autonomous Robotics

Winter 2025

Abhishek Gupta

TAs: Carolina Higuera, Entong Su, Bernie Zhu

Class Outline

State Estimation

- Robotic System Design
- Filtering
- Localization
- SLAM

Control

- Feedback Control
- PID Control
- MPC
- LQR

Planning

- Search
- Heuristic Search
- Motion Planning
- Lazy Search

Learning

- Imitation Learning
- Policy Gradient
- Actor-Critic
- Model-Based RL

Logistics

- HW2 Due Tue Feb 4
- HW3 out Wed Feb 5
- Reading discussions due a week from Monday on EdStem

- Post questions, discuss any issues you are having on Ed.
- Students with **no** access to 002, e-mail us with your student ID.
- Students that have not been added to the class, email abhgupta@cs.washington.edu with the subject-line “Waitlisted for CSE478”

Recap

Original Kalman Filter Algorithm

Initial Prior

$$p(x_0)$$

Dynamics/Prediction
(given some u)

Estimate $\overline{Bel}(x_{t+1})$

$$p(x_{t+1}|u_{0:t+1}, z_{0:t}) = \mathcal{N}(A\mu_{t|0:t} + Bu_{t+1}, A\Sigma_{t|0:t}A^T + Q_{t+1})$$

Measurement/Correction
(given some z)

Estimate $Bel(x_{t+1})$

$$p(x_{t+1}|u_{0:t+1}, z_{0:t+1})$$
$$= \mathcal{N}(\mu_{t+1|0:t} + K_{t+1}(z_{t+1} - C\mu_{t+1|0:t}), (I - K_{t+1}C)\Sigma_{t+1|0:t})$$

EKF Algorithm – linearize non-linear functions

Initial Prior

$$p(x_0)$$

Linearize dynamics

$$x_{t+1} = g(x_t, u_t) + \epsilon_t \;\approx g(\mu_t, u_t) + \left.\frac{\partial g(x_t, u_t)}{\partial x_t}\right|_{x_t=\mu_t} (x_t - \mu_t) + \epsilon_t$$

Dynamics/Prediction (given some u)

Estimate $\overline{Bel}(x_t)$

$$p(x_{t+1} | z_{0:t}, u_{0:t}) \sim \mathcal{N}(g(\mu_t, u_t), G\Sigma_{t|0:t}G^T + Q_t)$$

Linearize measurement

$$z_t = h(x_t) + \delta_t \;\approx h(\bar{\mu}_t) + \left.\frac{\partial h(x_t)}{\partial x_t}\right|_{x_t=\bar{\mu}_t} (x_t - \bar{\mu}_t) + \delta_t$$

Measurement/Correction (given some z)

Estimate $Bel(x_t)$

$$p(x_{t+1} | z_{0:t+1}, u_{0:t}) = \mathcal{N}(\mu_{t+1|0:t} + K_{t+1}(z_{t+1} - h(\bar{\mu}_t), (I - K_{t+1}H)\Sigma_{t+1|0:t})$$

Ok so what have we learned

Bayesian Filtering!

Key Idea: Apply Markov to get a recursive update!

Step 0. Start with the belief at time step t-1

$$bel(x_{t-1})$$

Step 1: Prediction - push belief through dynamics given action

$$\overline{bel}(x_t) = \sum P(x_t|u_t, x_{t-1})bel(x_{t-1})$$

Step 2: Correction - apply Bayes rule given measurement

$$bel(x_t) = \eta P(z_t|x_t)\overline{bel}(x_t)$$

Motion and Measurement Model

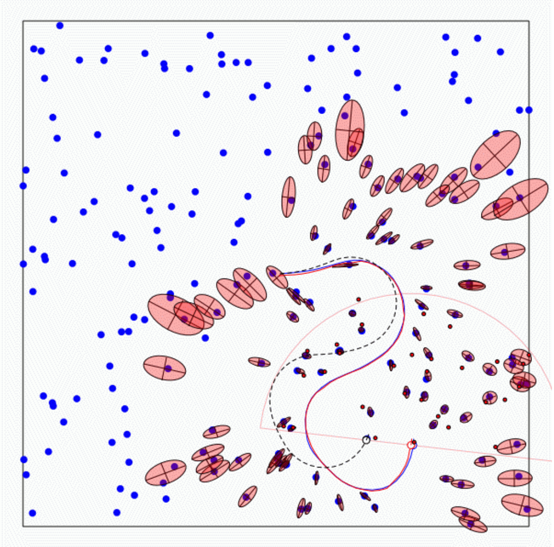
Linear Gaussian
– Kalman Filter

Nonlinear Gaussian
– Extended Kalman Filter

Nonlinear non-gaussian
– Particle Filter

What if we didn‘t know the map?

- So far, the maps have been assumed to be known → often untrue → SLAM problem
- A robot is exploring an unknown, static environment.



Given:

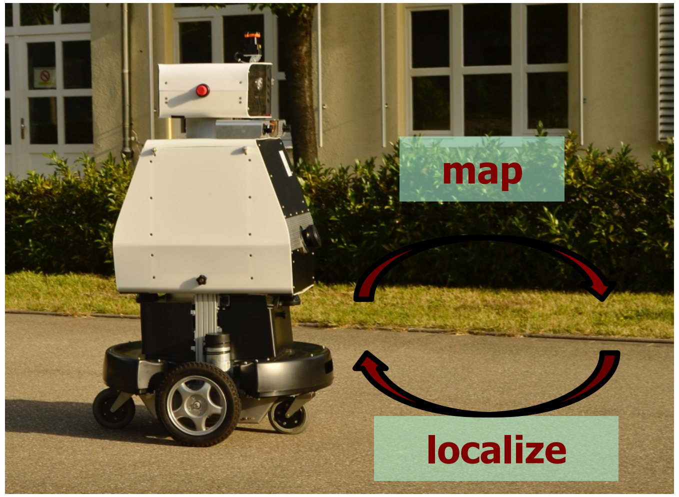
The robot’s controls (u)

Observations of nearby features (z)

Estimate:

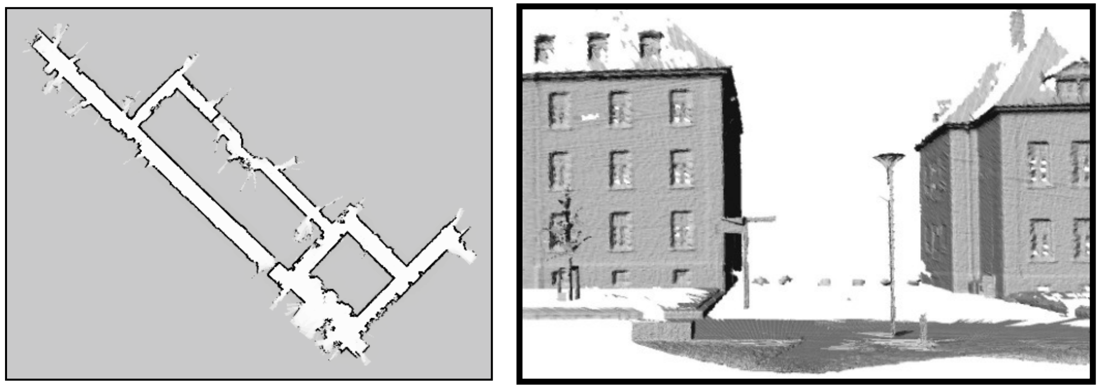
Map of features (x)

Path of the robot (x)

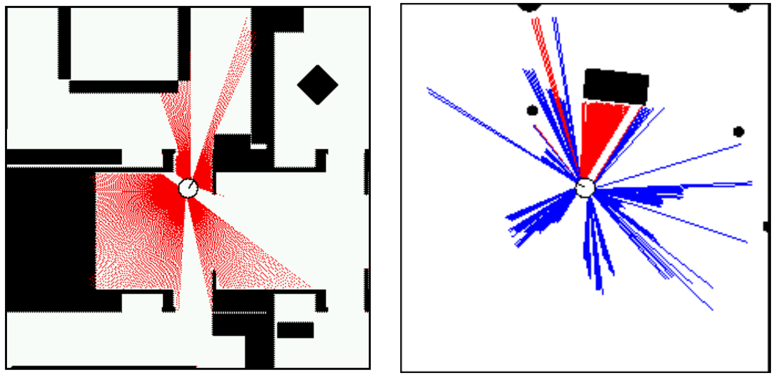


Why is SLAM difficult?

- Localization assumed map was perfectly known in the sensor/motion
- Mapping assumes position is fully known
- Doing both jointly is hard!

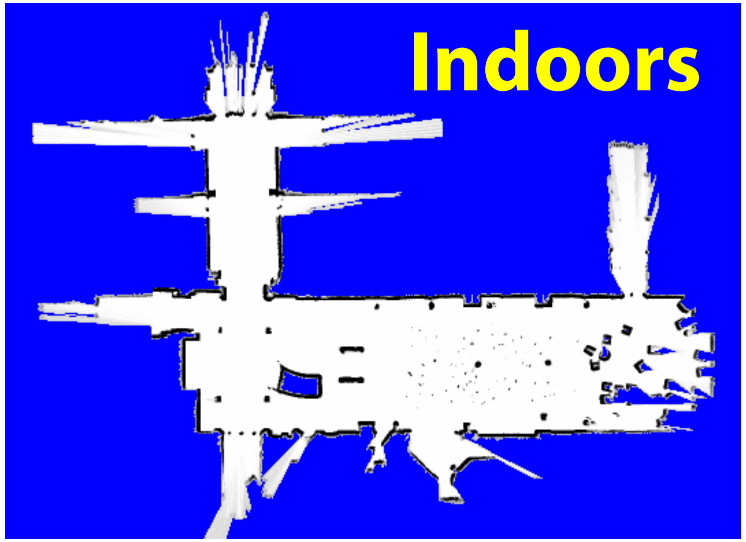


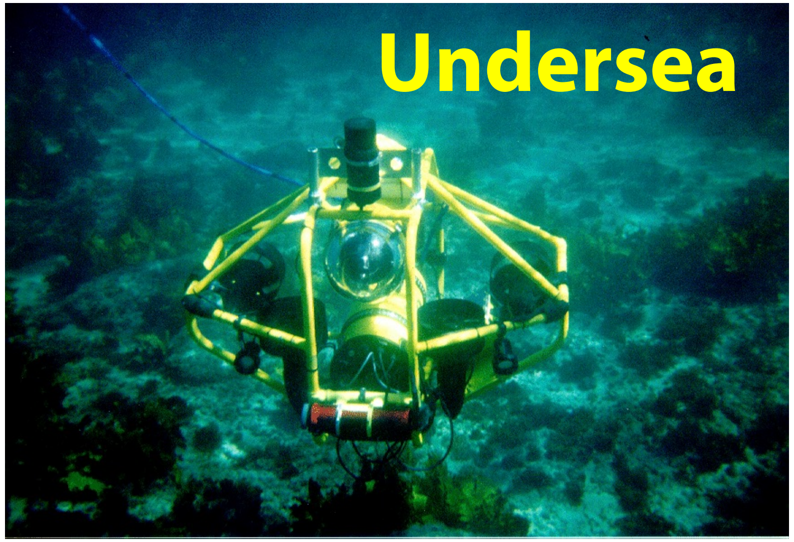
Mapping

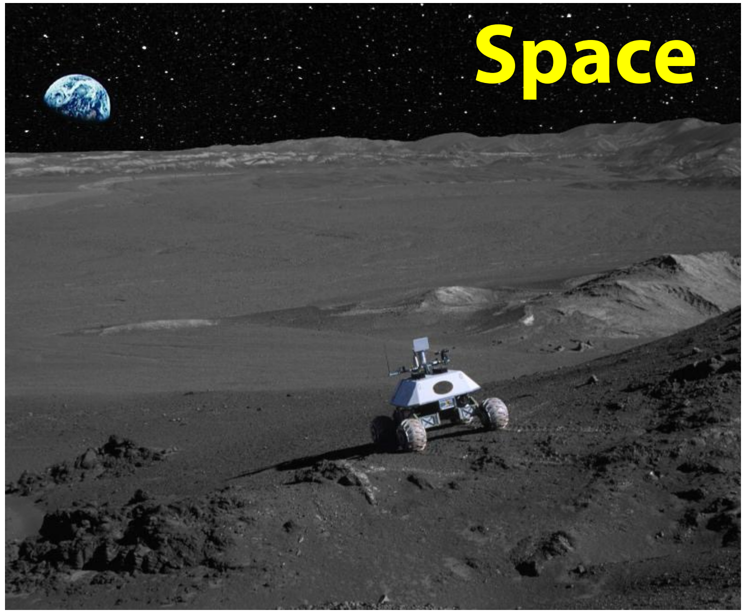


Localization

SLAM Applications







Definition of the SLAM Problem

Given

- The robot's controls

$$u_{1:T} = \{u_1, u_2, u_3, \ldots, u_T\}$$

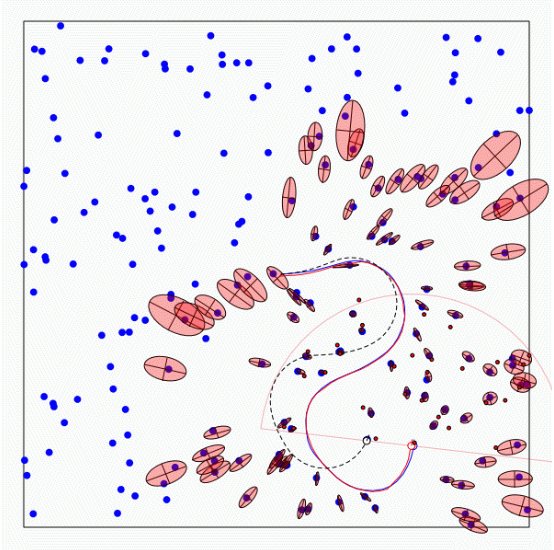
- Observations

$$z_{1:T} = \{z_1, z_2, z_3, \ldots, z_T\}$$

Wanted

- Map of the environment m

- Path of the robot $x_{0:T} = \{x_0, x_1, x_2, \ldots, x_T\}$



Courtesy: Cyrill Stachniss

Bayes Filter

- Recursive filter with prediction and correction step

- Prediction

$$\overline{Bel}(x_t) = \int p(x_t | x_{t-1}, u_{t-1}) Bel(x_{t-1}) dx_{t-1}$$

- Correction

$$Bel(x_t) = \eta p(z_t | x_t) \overline{Bel}(x_t)$$

- EKF Slam sets x to be (position of robot, position of landmarks)

EKF SLAM

- Application of the EKF to SLAM
- Estimate robot's pose and locations of landmarks in the environment
- Assumption: known correspondences
- State space (for the 2D plane) is

$$x_t = (\underbrace{x, y, \theta}_{\text{robot's pose}}, \underbrace{m_{1,x}, m_{1,y}}_{\text{landmark 1}}, \ldots, \underbrace{m_{n,x}, m_{n,y}}_{\text{landmark n}})^T$$

Courtesy: Cyrill Stachniss

EKF SLAM: Filter Cycle

1. State prediction
2. Measurement prediction
3. Measurement + Data Association
4. Update

Courtesy: Cyrill Stachniss

Why is this useful - SLAM

Lecture Outline

Where does control fit in the roadmap

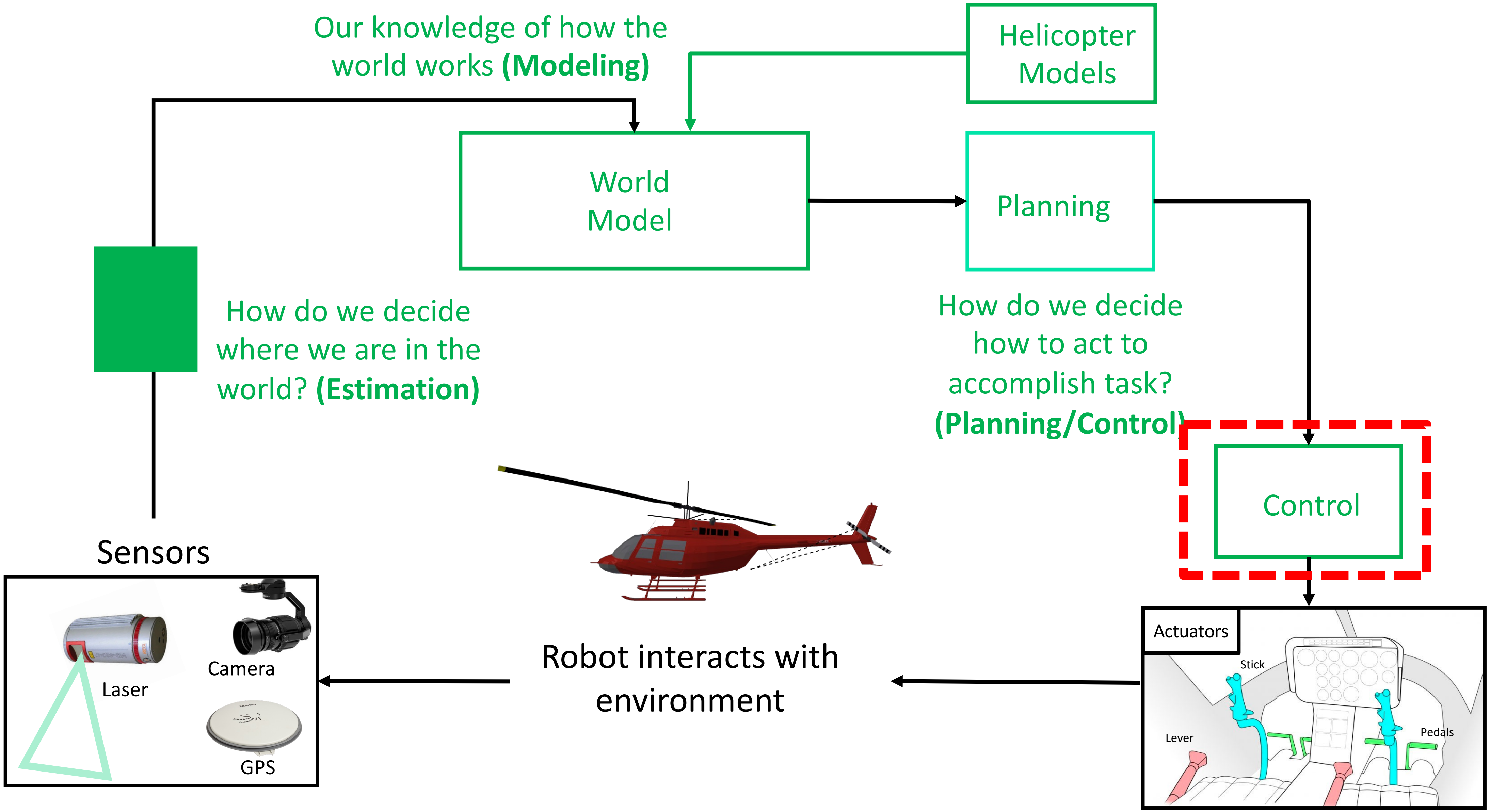
↓

Why control problems are hard

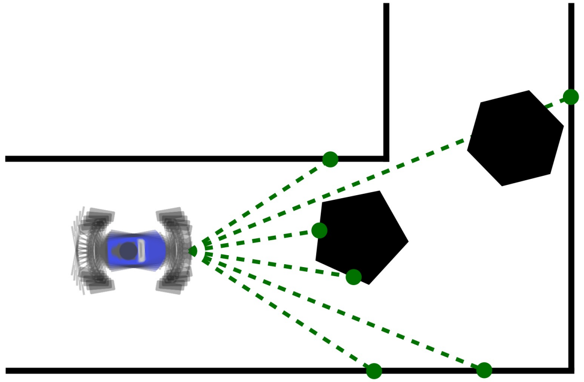
↓

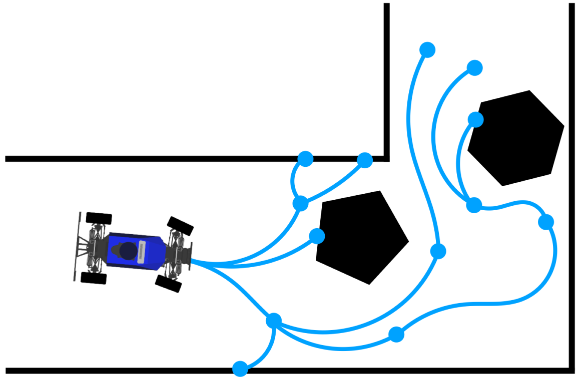
How to formulate control problems

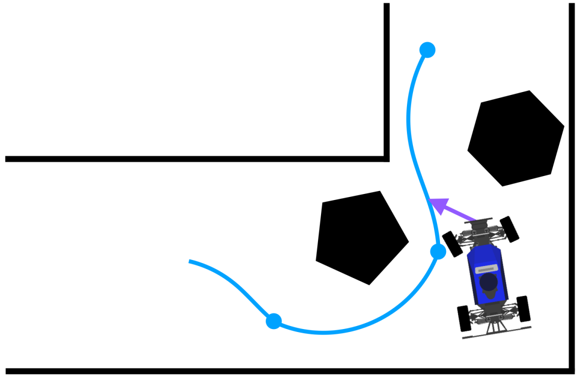
Let’s zoom back out



The Sense-Plan-Act Paradigm







Estimate robot state

Solved over last 4 weeks

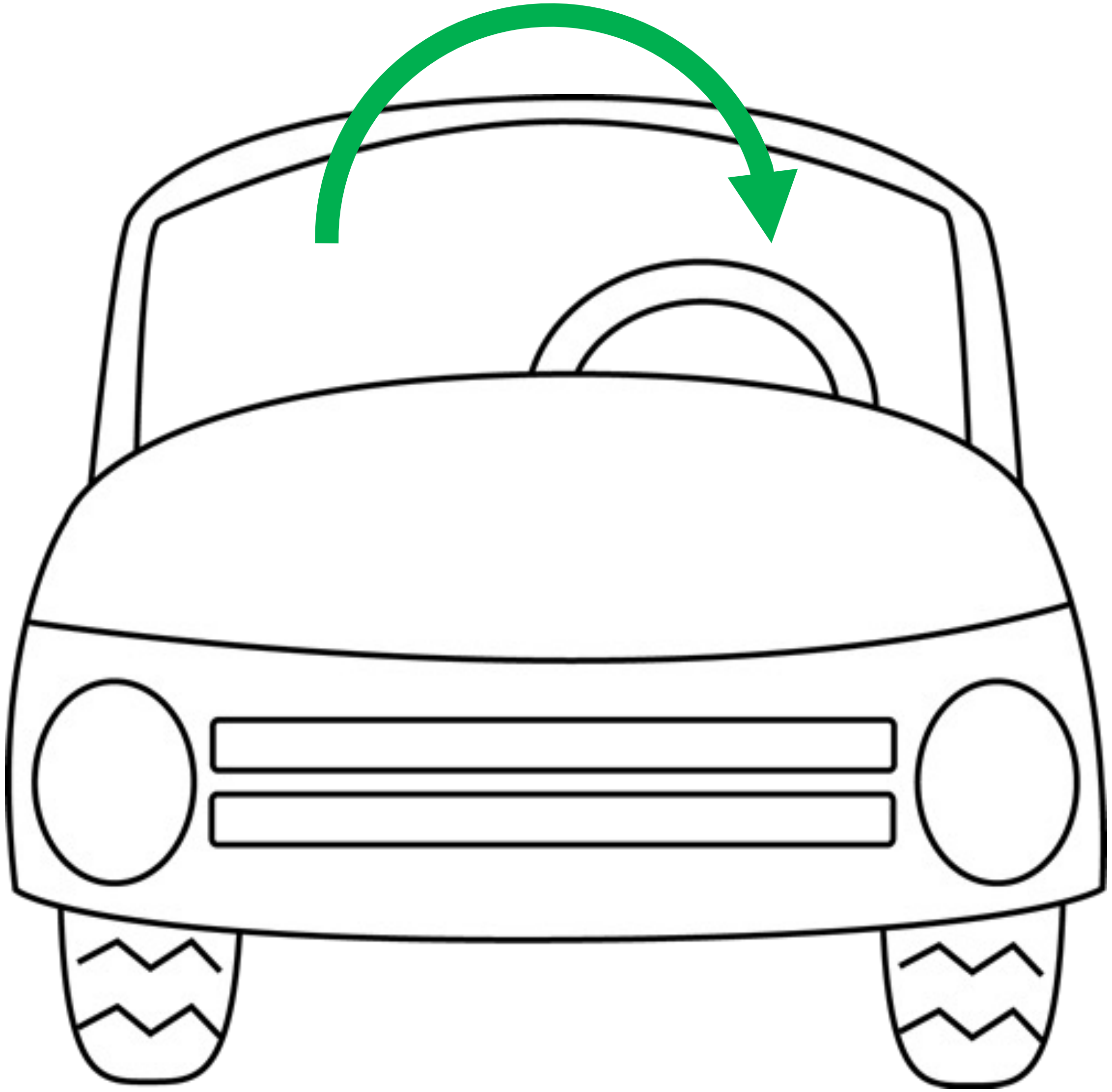
Plan sequence of motions

Assume to be solved for now

Control robot to follow plan

??

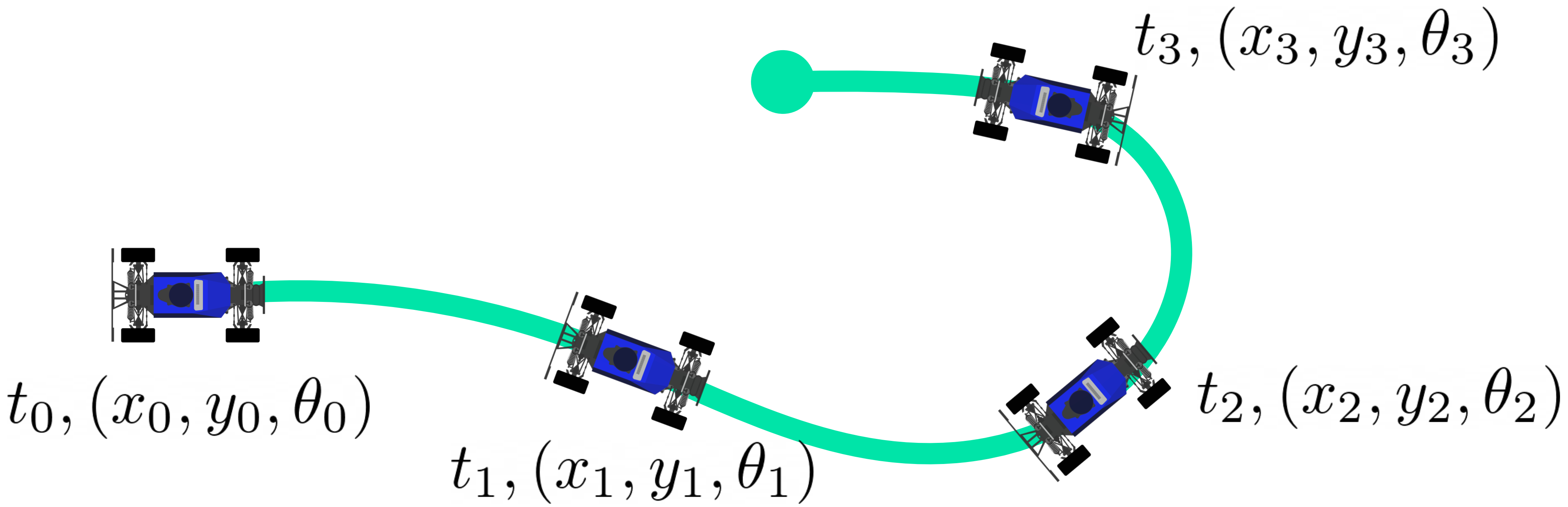
From perception to control …



When I think about control …

What is Control?

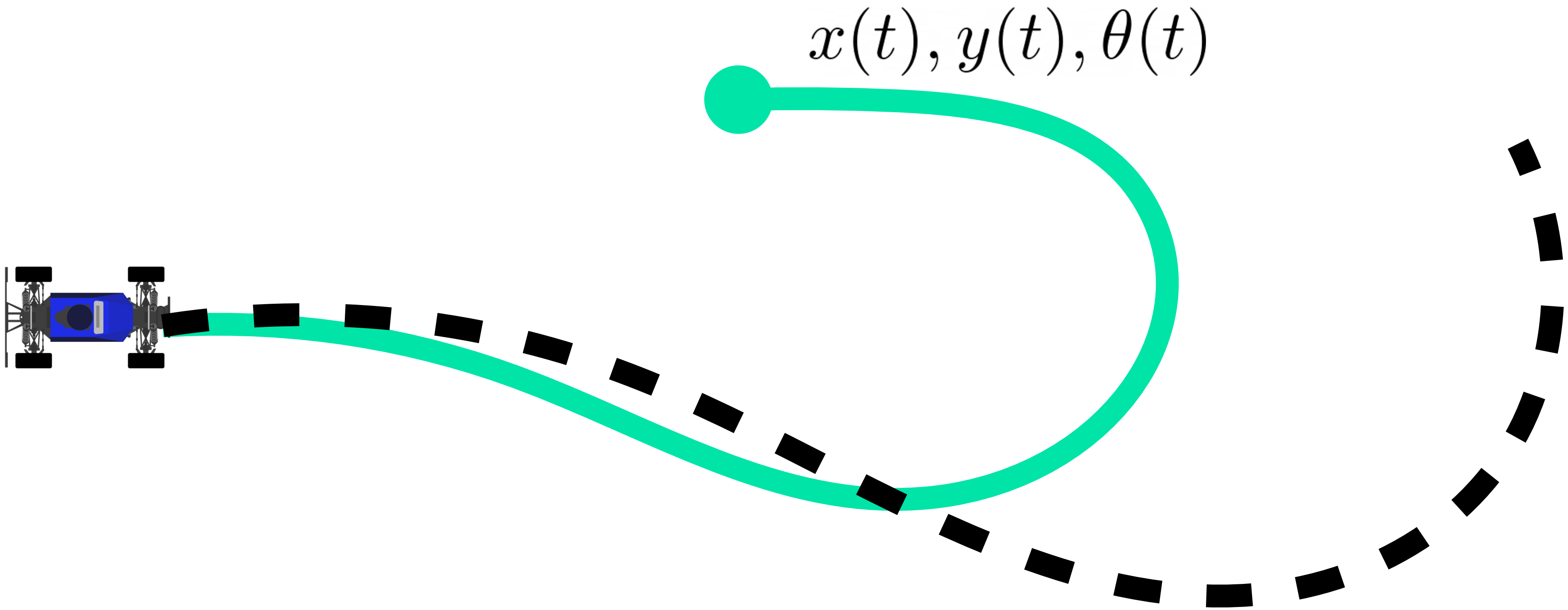
What is a Plan?



Can express this problem as **tracking a reference** trajectory

$$x(t), y(t), \theta(t)$$

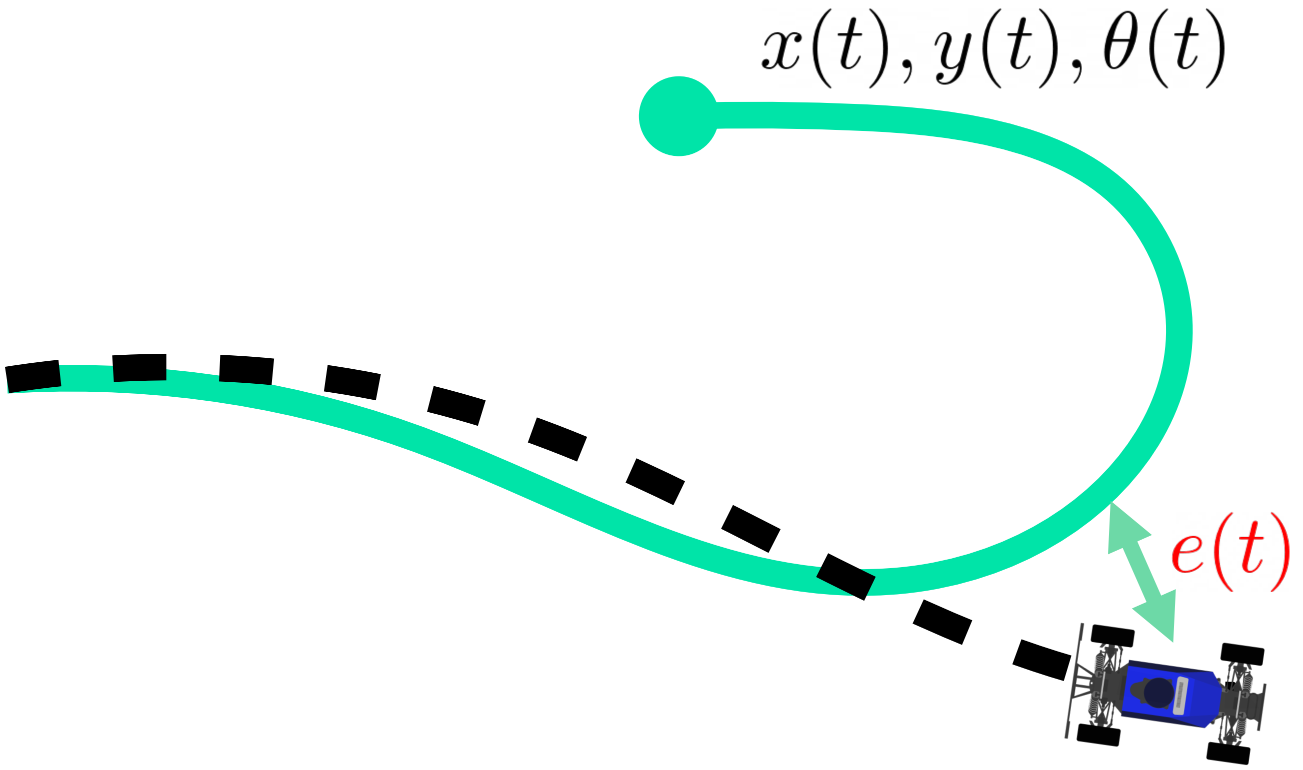
Why Feedback Control?



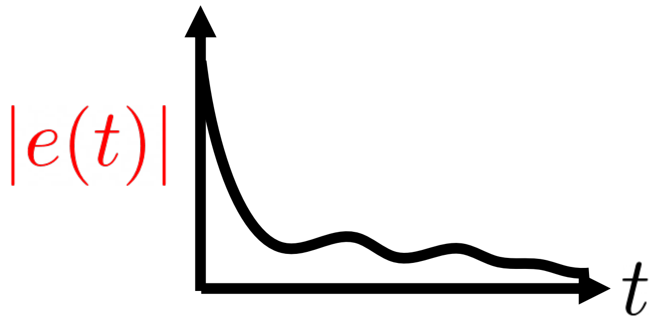
What if we send out controls $u(t)$ from kinematic car model?

Open-loop control leads to **accumulating errors**!

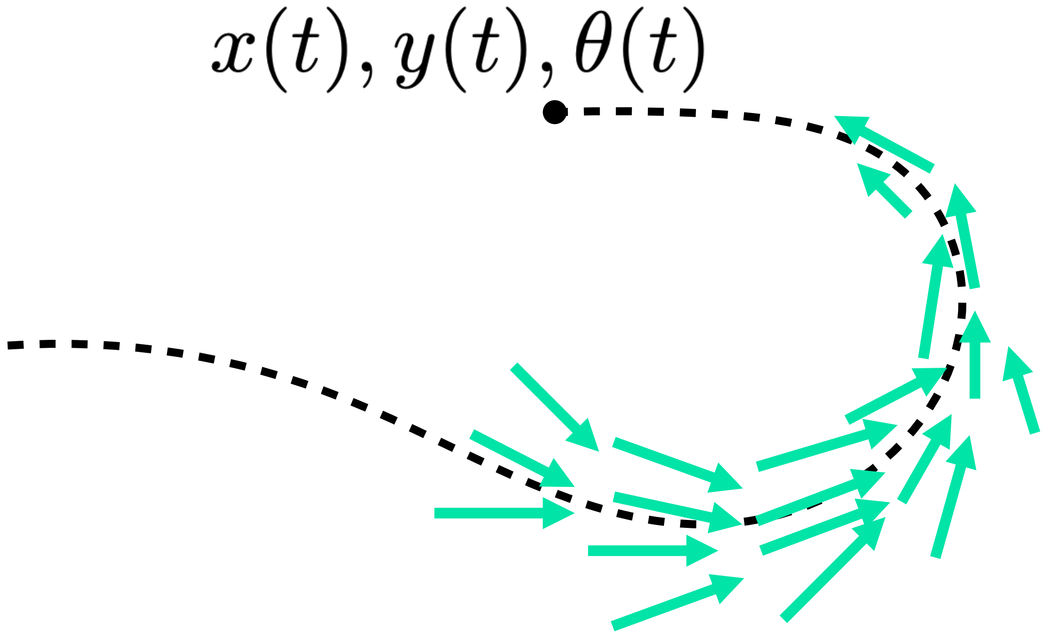
Feedback Control



1. Measure error between reference and current state.
2. Take actions to minimize this error.

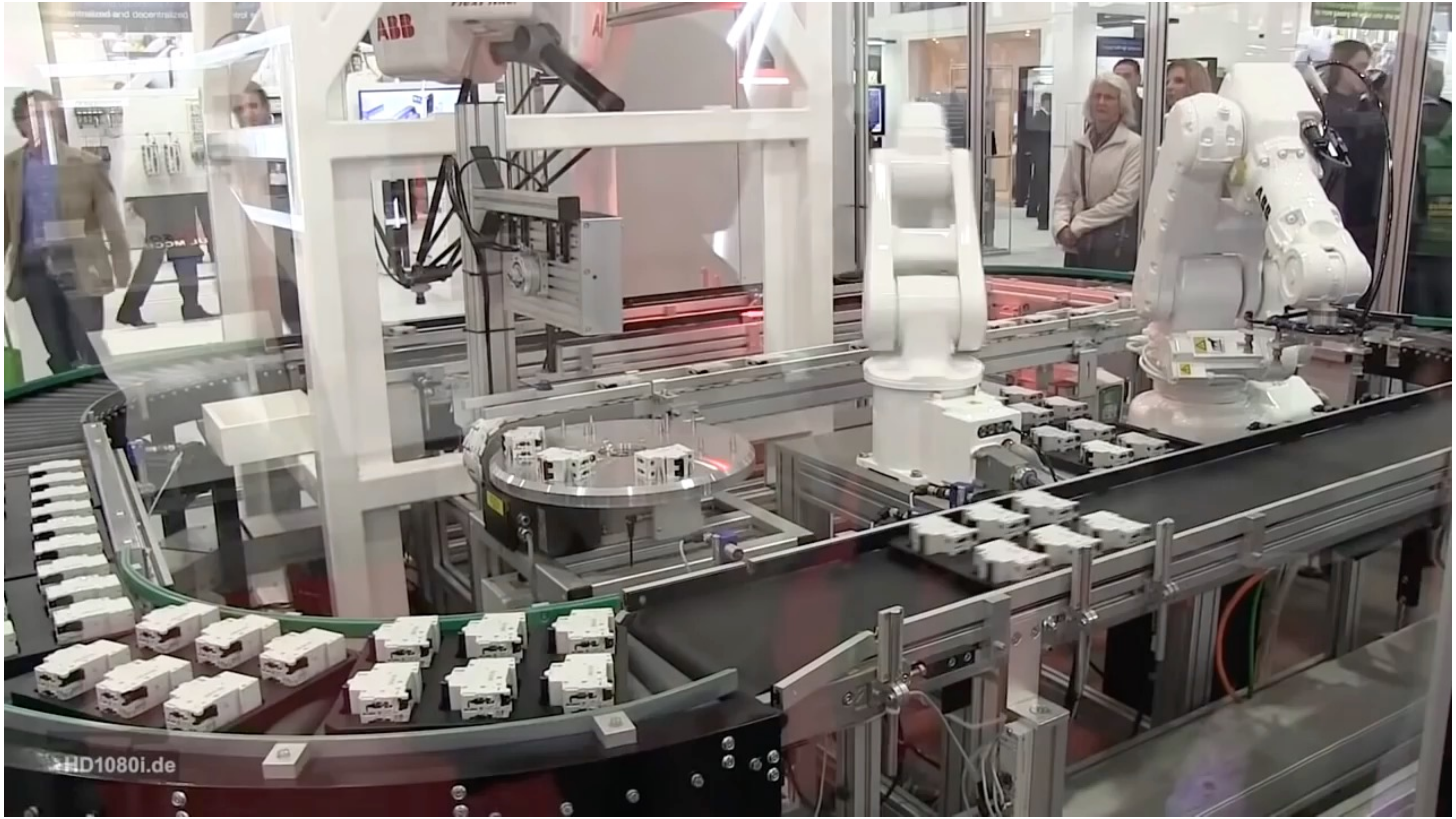


Useful to think of control laws as vector fields



Is this still a research problem?

Industrial robots hard at work



https://www.youtube.com/watch?v=J_8OnDsQVZE&t=315s

Assumptions made by such controllers

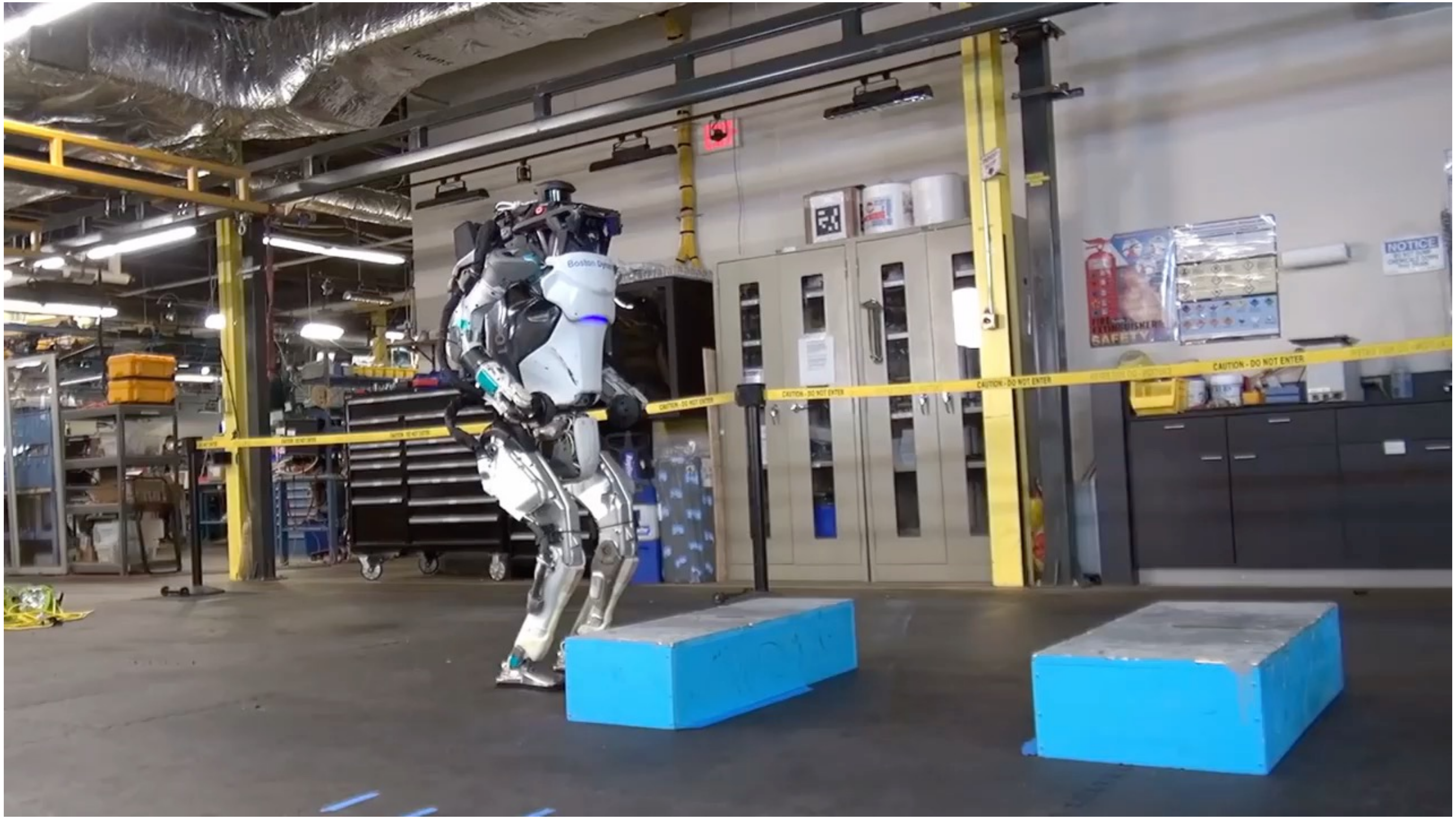
1. Fully actuated: There exists an inverse mapping from reference to control actions

$$\sigma(t) \rightarrow u(t)$$

2. Almost no execution error or state estimation error

3. Enough control authority to clamp down errors / overcome disturbances

The Atlas robot hard at … play?



https://www.youtube.com/watch?v=fRj34o4hN4I

Challenge 1: Underactuated systems

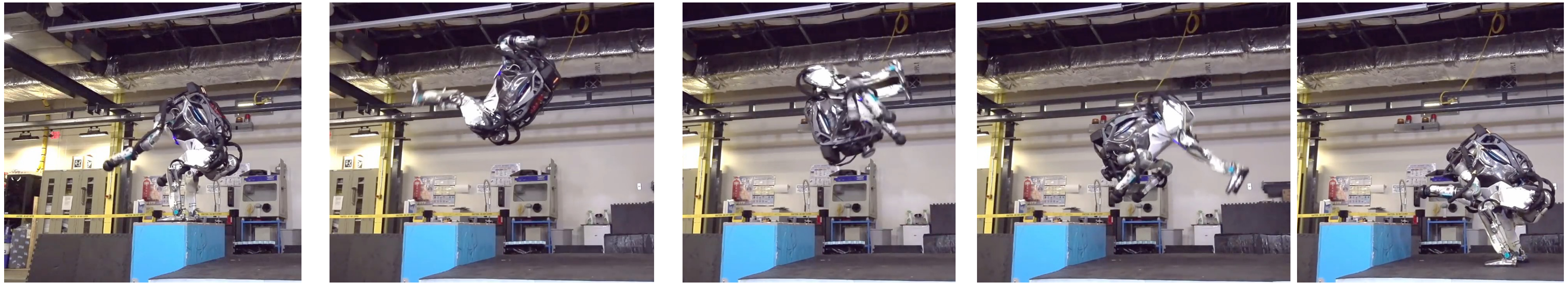
Fully actuated: There exists an inverse mapping from reference to control actions

$$\sigma(t) \quad u(t)$$

We don’t have full authority to move the system along arbitrary trajectories

Challenge 1: Underactuated systems

What affects the error between robot state and reference?



Some
initial
motor
thrust …

Whole lot of gravity!

Whole lot of momentum!

… some precise
control adjustments

Question:

If we know the model of our robot, can’t we solve a huge optimization problem to figure out control?

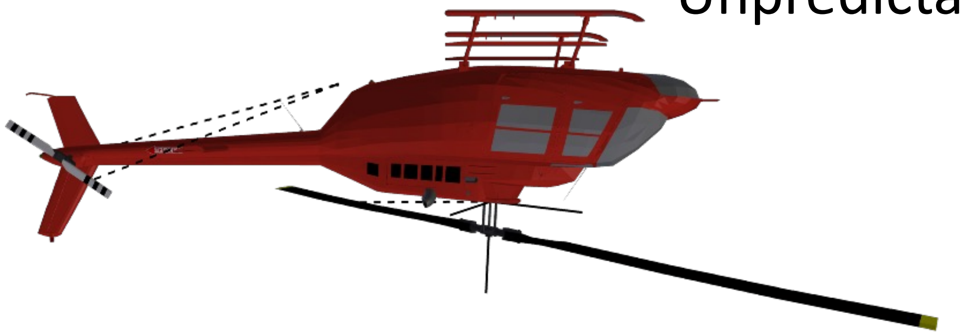
Doing backflips with a helicopter

Redbull Eurocopter BO-105

https://www.youtube.com/watch?v=RGu45s1_QPU

And what is this model ?!?

Nothing countering gravity!

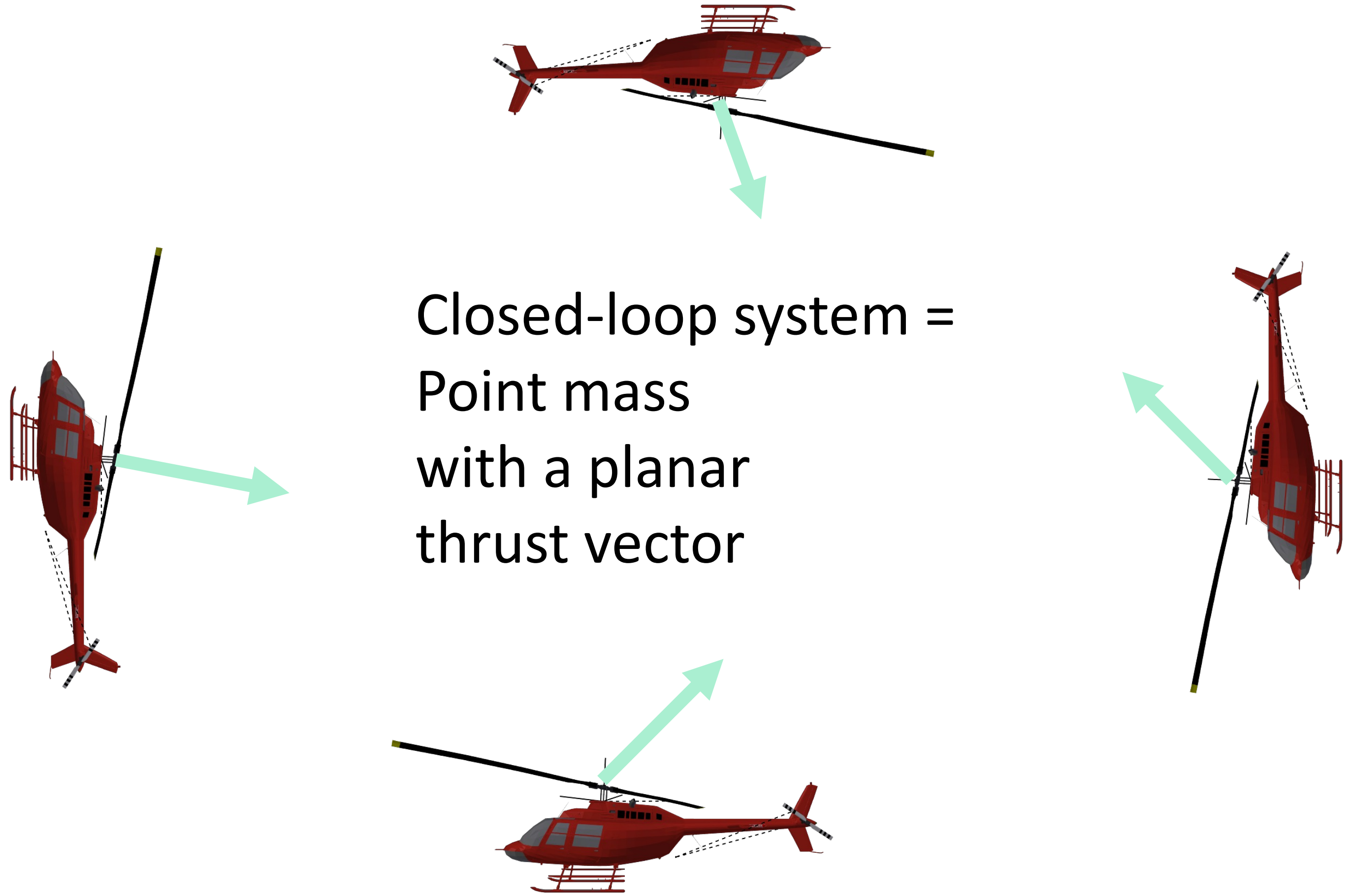


Unpredictable drag forces!

Chaotic vortex around blades!

Hopeless to assume we know exactly how the helicopter will behave upside down…

Challenge 2: Choosing good closed-loop models



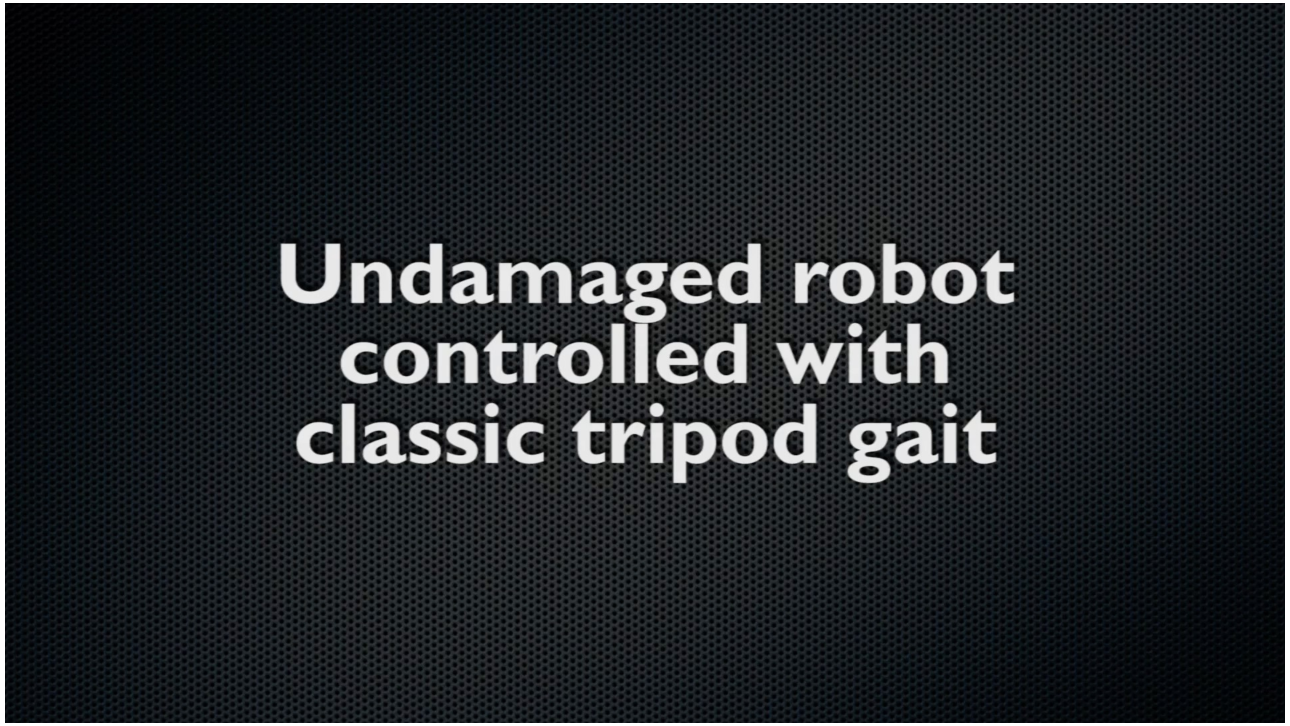
Closed-loop system =
Point mass
with a planar
thrust vector

Chaotic dynamics + Feedback control law = Well-behaved system

Challenge 3: Model changing on the fly!

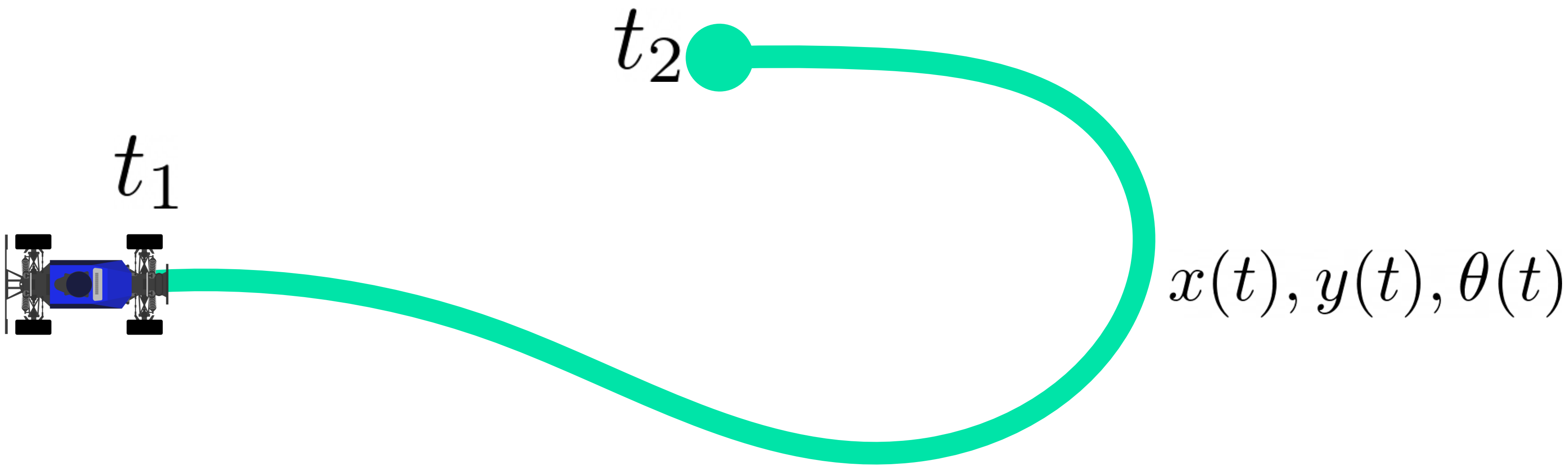
Run real-time estimators for wheel characteristics

Need control laws for all possible model parameters



Ok let’s control racecars!

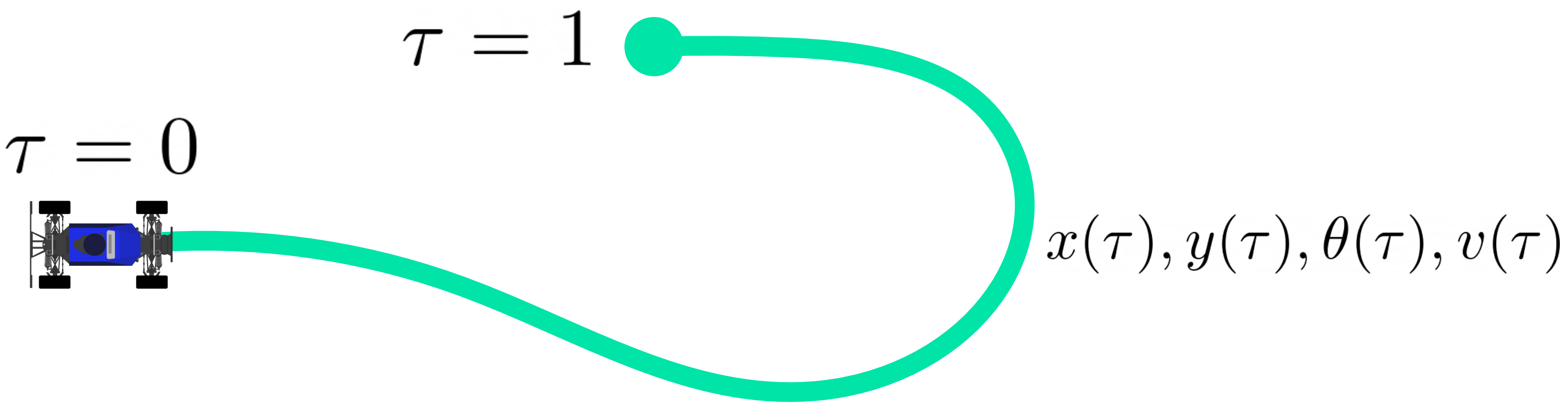
Reference Parameterizations



Option 1: **Time**-parameterized trajectory

Pro: Useful if we want the robot to respect time constraints
Con: Sometimes we only care about deviation from reference

Reference Parameterizations



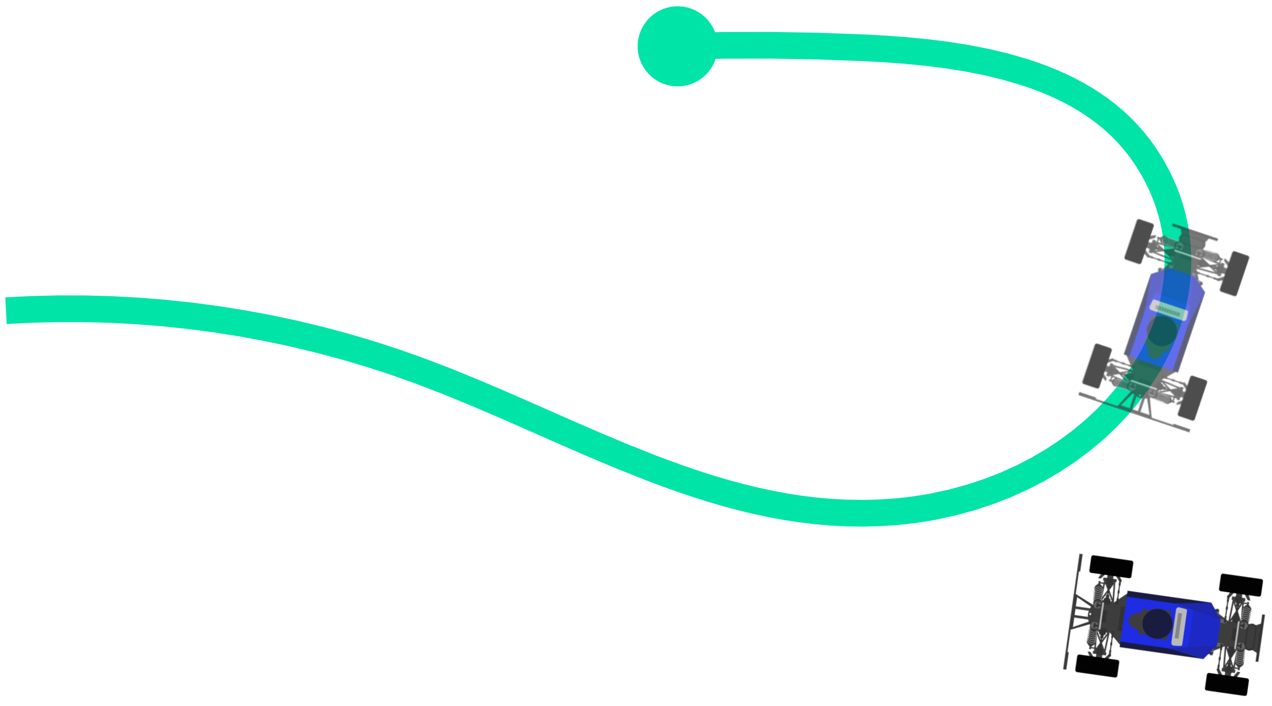
Option 2: **Index**-parameterized geometric path (untimed)

Pro: Useful for conveying shape for the robot to follow
Con: Can't control when robot will reach a point

Controller Design Decisions

1. Get a reference path/trajectory to track
2. Pick a reference state from the reference path/trajectory
3. Compute error to reference state
4. Compute control law to minimize error

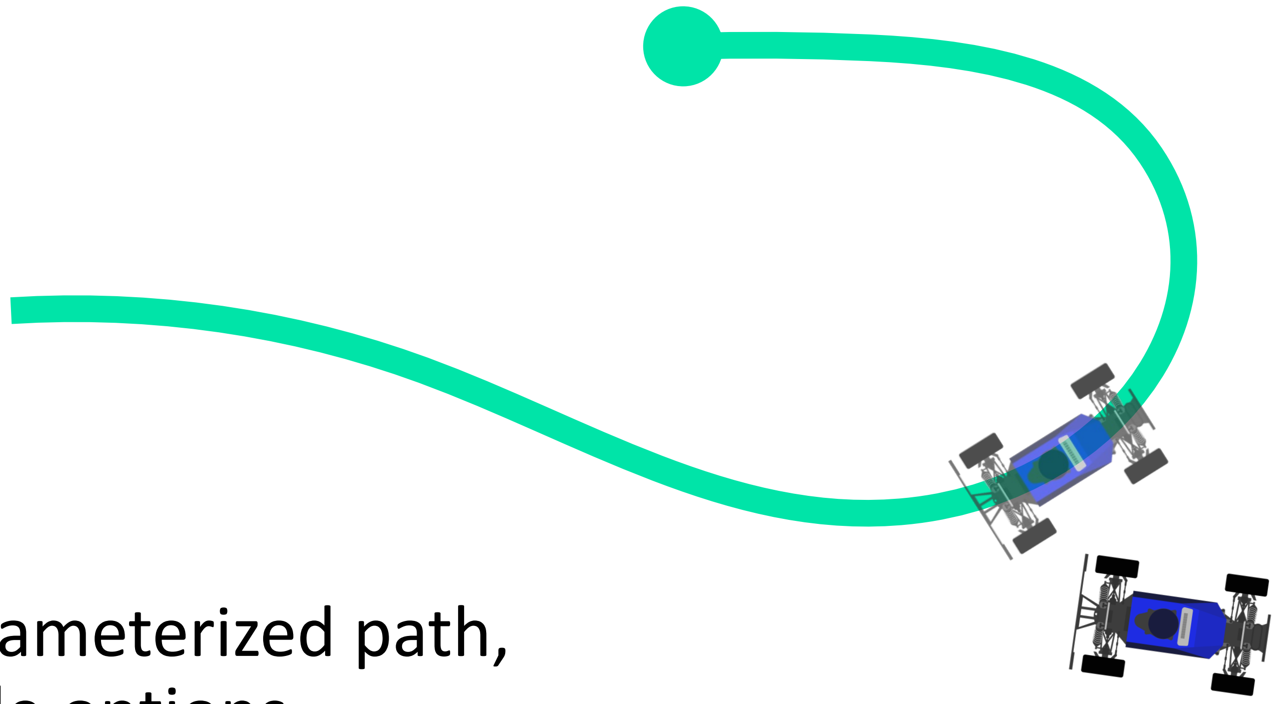
Step 2: Pick a reference (desired) state



$$\begin{bmatrix} x_{\text{ref}} \\ y_{\text{ref}} \\ \theta_{\text{ref}} \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

How do we choose a reference state?



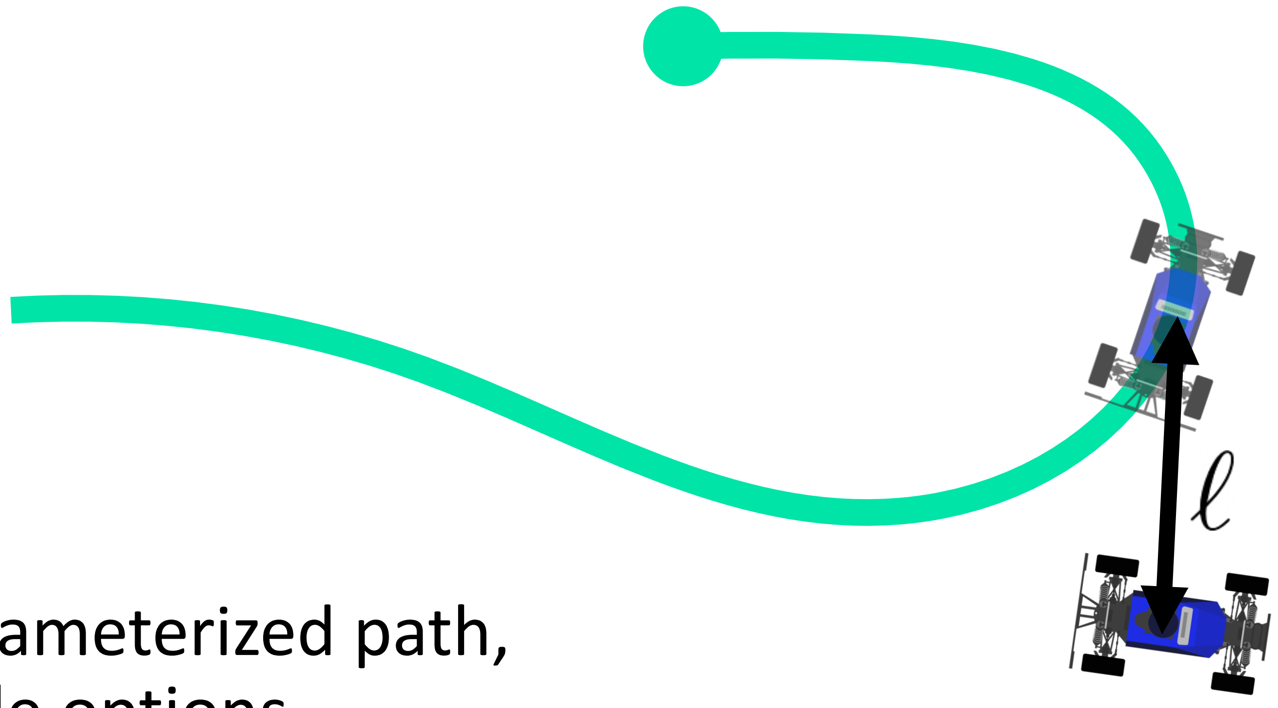
$$\begin{bmatrix} x(\tau_{\text{ref}}) \\ y(\tau_{\text{ref}}) \\ \theta(\tau_{\text{ref}}) \\ v(\tau_{\text{ref}}) \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

For an **index**-parameterized path,
there are multiple options.

Closest point $$\tau_{\text{ref}} = \arg\min_{\tau} \| \begin{bmatrix} x & y \end{bmatrix}^\top - \begin{bmatrix} x(\tau) & y(\tau) \end{bmatrix}^\top \|$$

How do we choose a reference state?

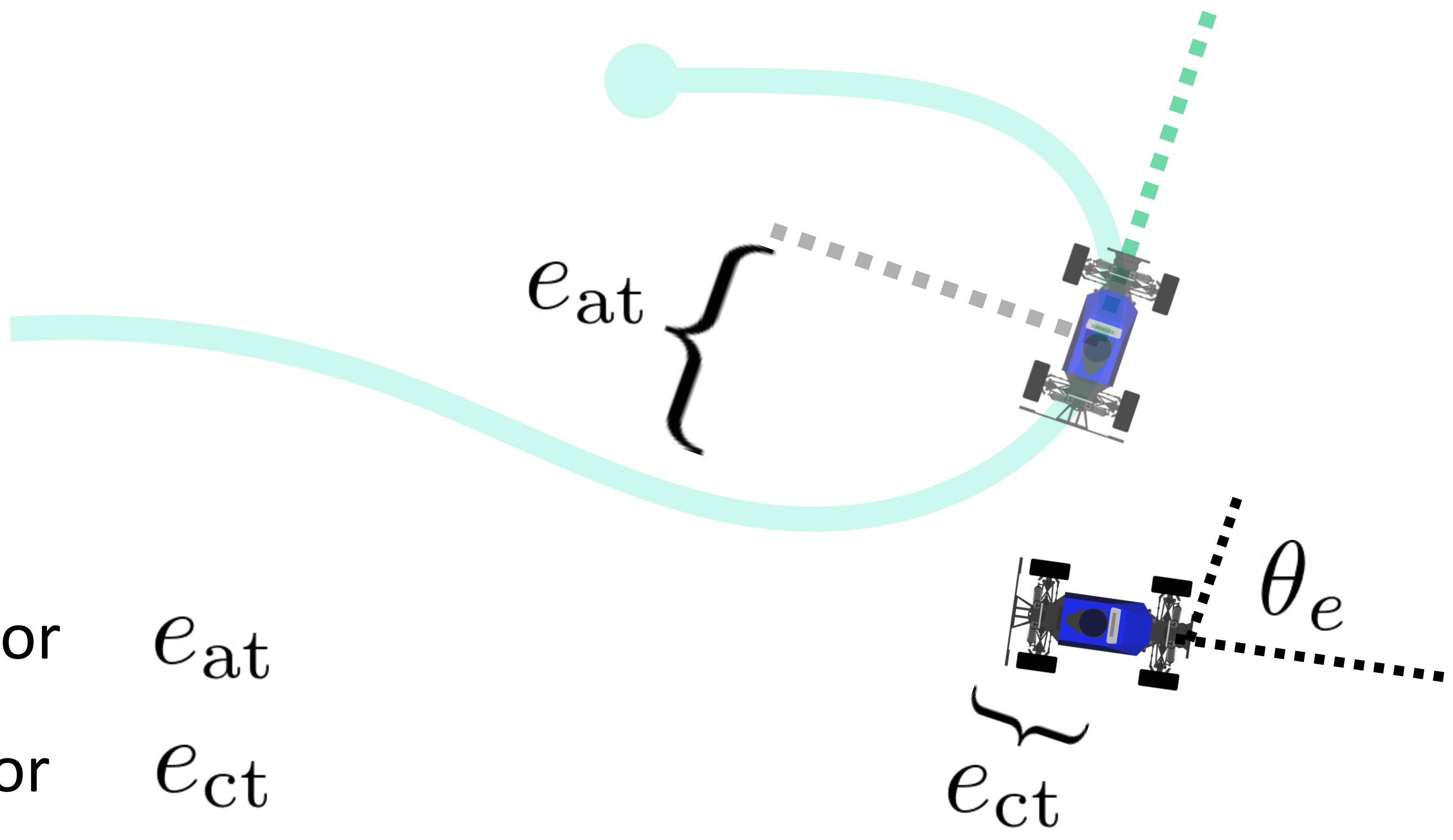


For an **index**-parameterized path, there are multiple options.

Lookahead

$$\tau_{\text{ref}} = \arg\min_{\tau} \left(\| \begin{bmatrix} x & y \end{bmatrix}^\top - \begin{bmatrix} x(\tau) & y(\tau) \end{bmatrix}^\top \| - \ell \right)^2$$

Step 3: Compute error to reference state

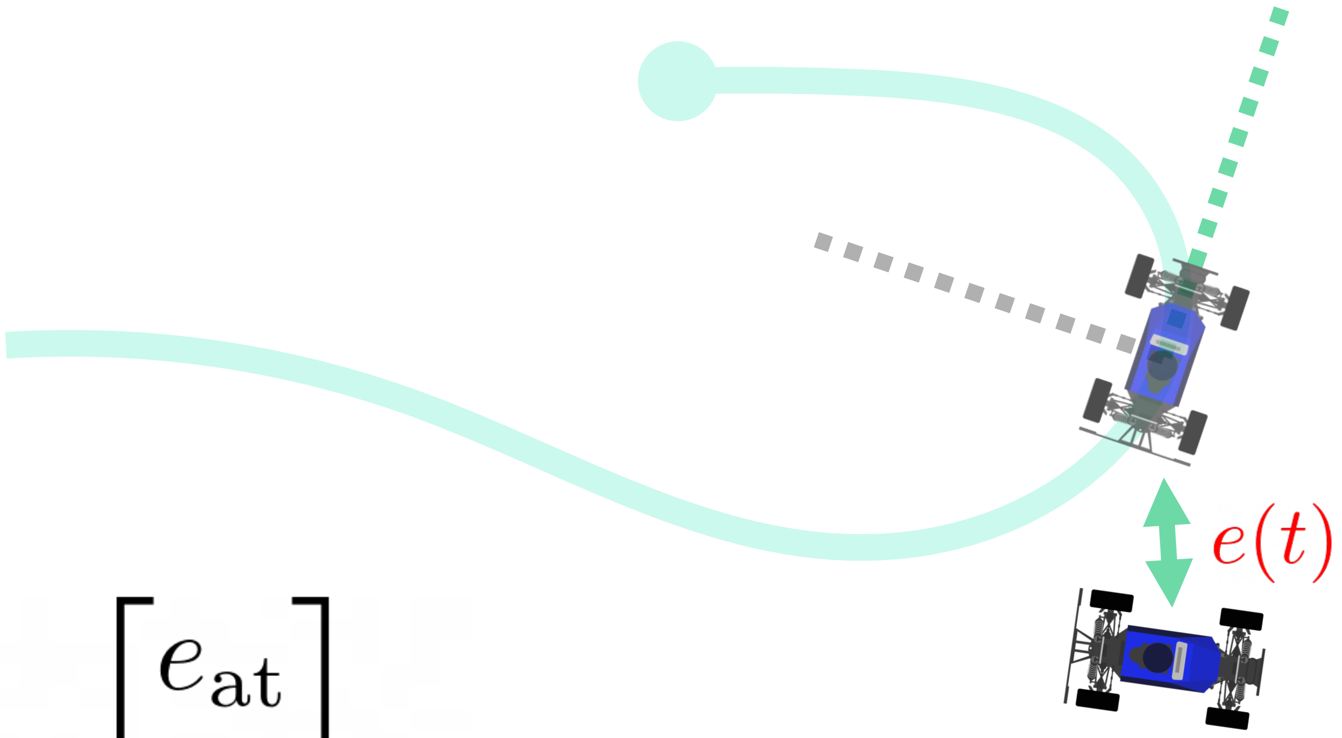


Along-track error e_{at}

Cross-track error e_{ct}

Heading error θ_e

Step 3: Compute error to reference state

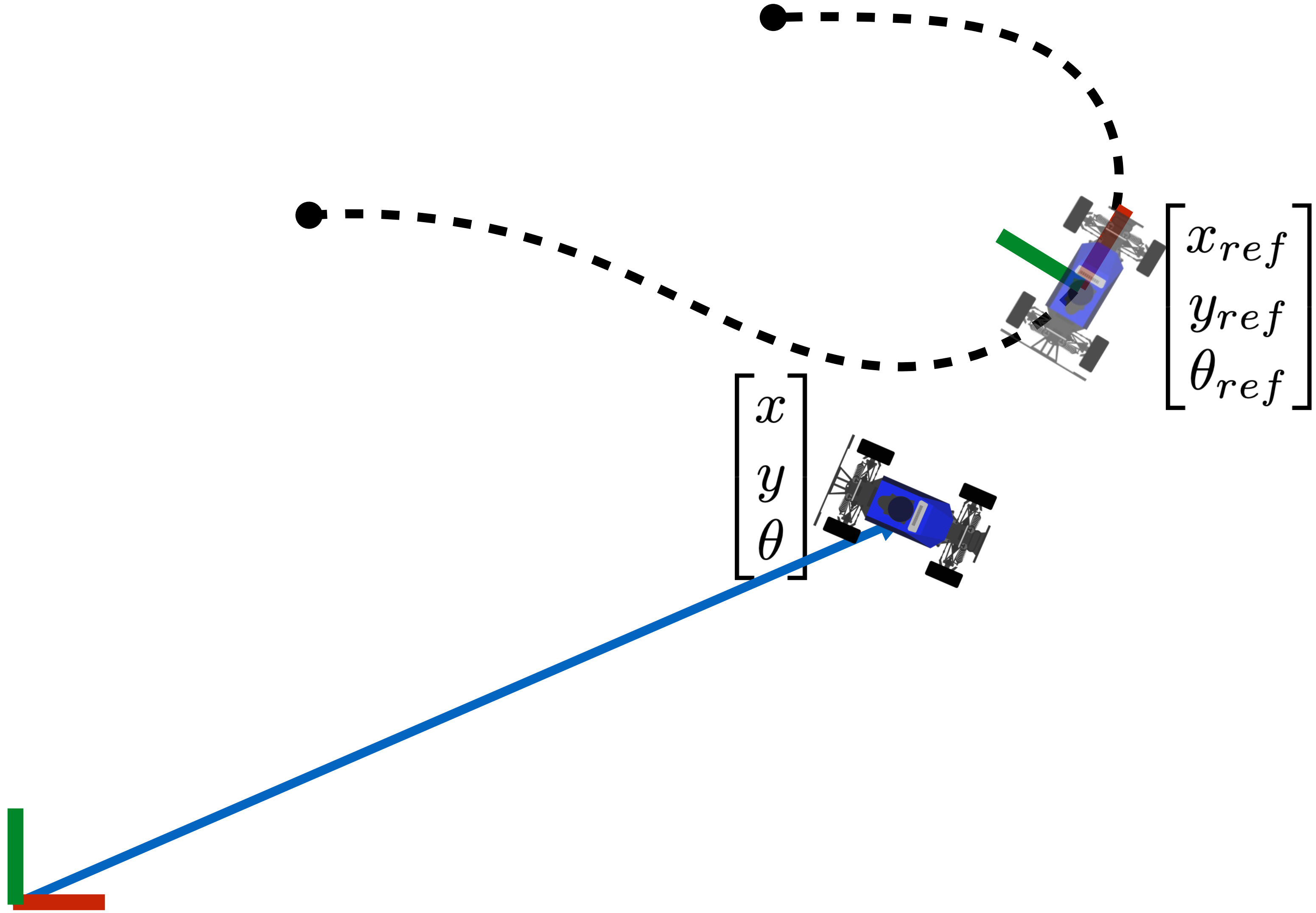


$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_G \rightarrow \begin{bmatrix} e_{\mathrm{at}} \\ e_{\mathrm{ct}} \\ \theta_e \end{bmatrix}_{\mathrm{ref}}$$

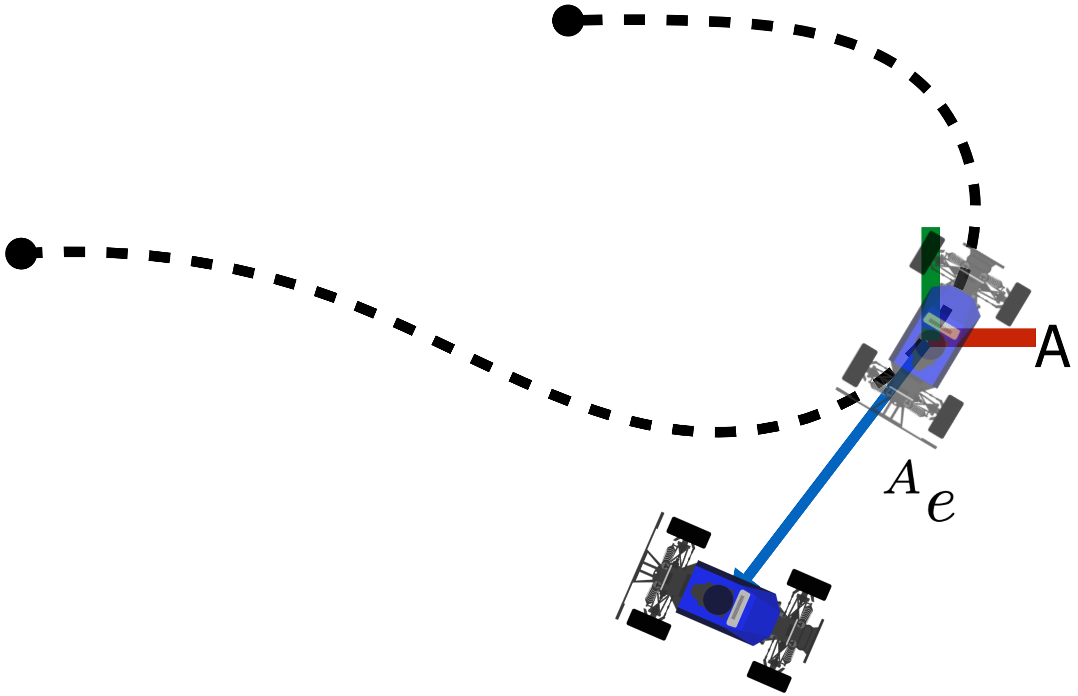
Aside: Rotation Matrices (Plane)

$$R = R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Step 3: Compute error to reference state



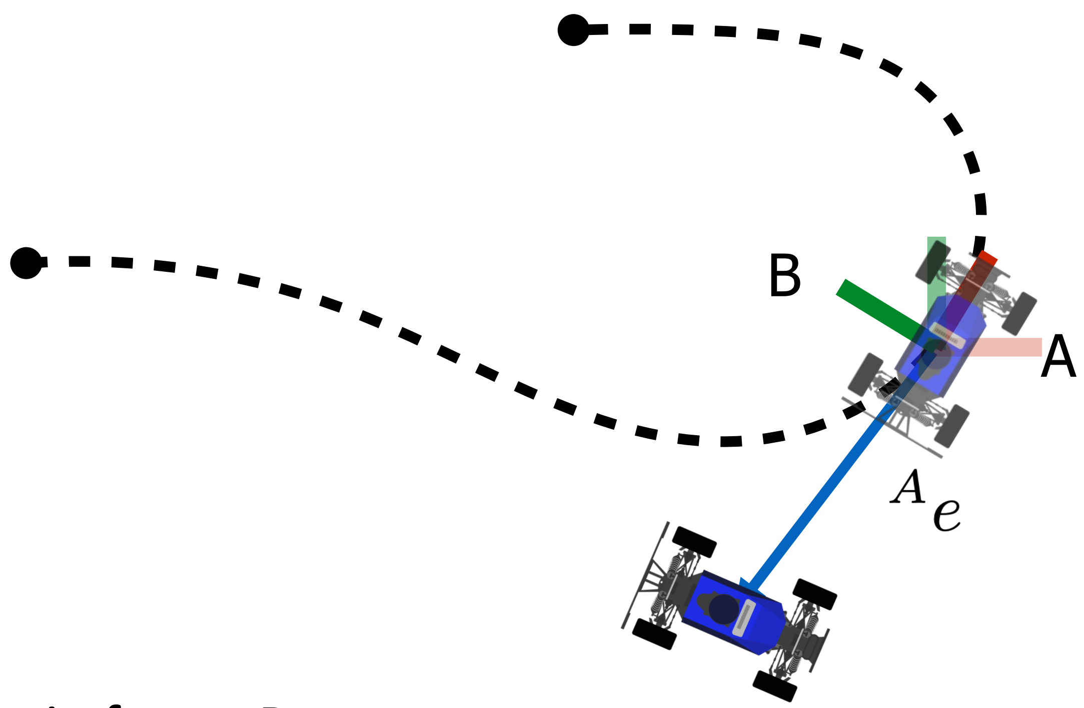
Step 3: Compute error to reference state



Position in frame A

$$ {}^{A}e = \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_{ref} \\ y_{ref} \end{bmatrix} $$

Step 3: Compute error to reference state

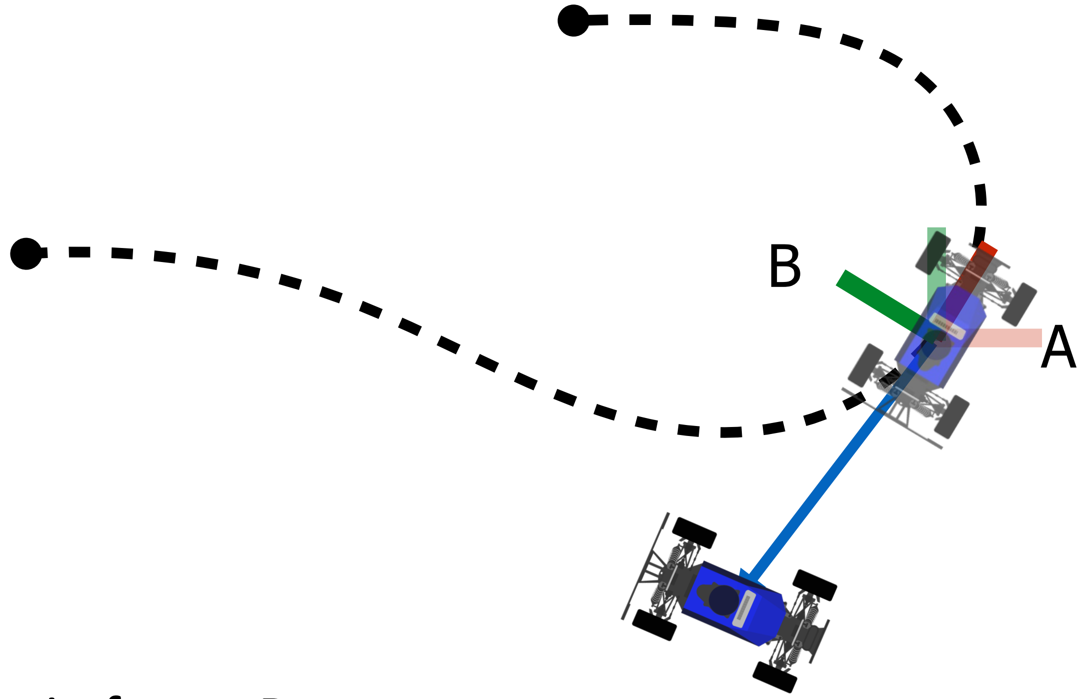


We want position in frame B

$$^{B}e = {}^{B}_{A}R \; {}^{A}e = R(-\theta_{ref}) \left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_{ref} \\ y_{ref} \end{bmatrix} \right)$$

(rotation of A w.r.t B) (rotation of A w.r.t B)

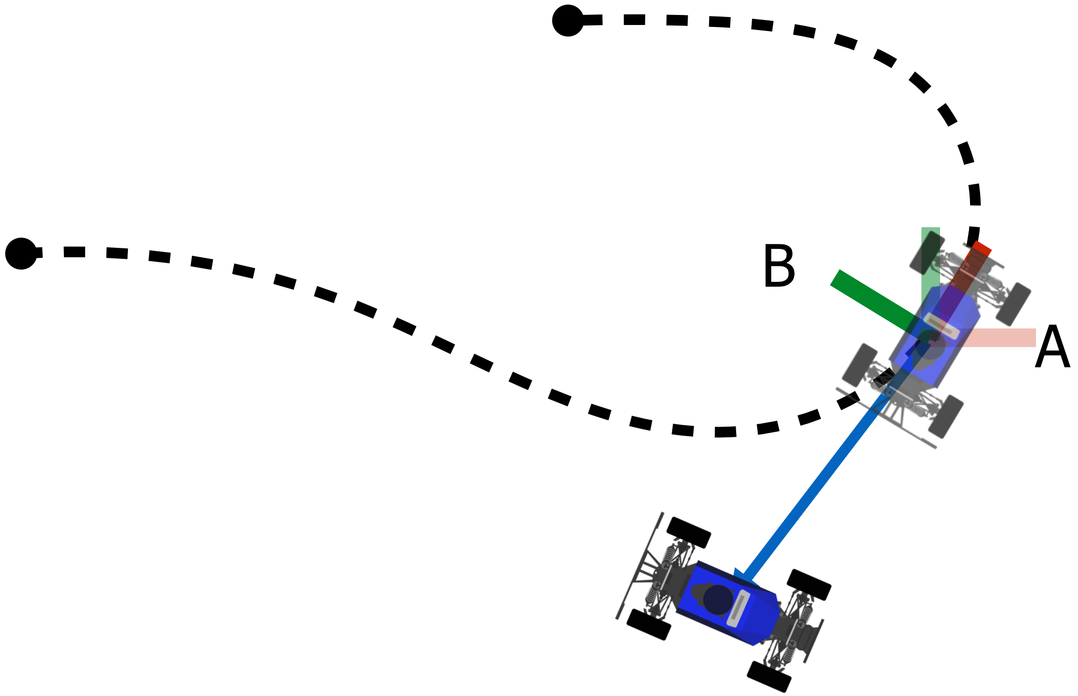
Step 3: Compute error to reference state



We want position in frame B

$$ {}^{B}e = \begin{bmatrix} e_{at} \\ e_{ct} \end{bmatrix} = \begin{bmatrix} \cos(\theta_{ref}) & \sin(\theta_{ref}) \\ -\sin(\theta_{ref}) & \cos(\theta_{ref}) \end{bmatrix} \left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_{ref} \\ y_{ref} \end{bmatrix} \right) $$

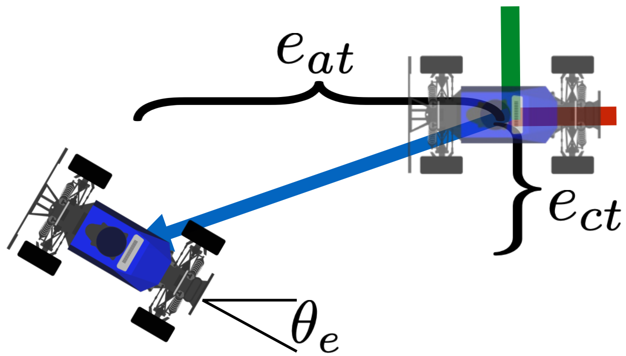
Step 3: Compute error to reference state



Heading error

$$\theta_e = \theta - \theta_{ref}$$

Step 3: Compute error to reference state



(Along-track) $e_{at} = \cos(\theta_{ref})(x - x_{ref}) + \sin(\theta_{ref})(y - y_{ref})$

(Cross-track) $e_{ct} = -\sin(\theta_{ref})(x - x_{ref}) + \cos(\theta_{ref})(y - y_{ref})$

(Heading) $\theta_e = \theta - \theta_{ref}$

Step 4: Compute control law

We will only control steering angle;
fixed constant speed
As a result, no real control for along-track error
Some control laws will only minimize cross-track error, others will also minimize heading

$$u = K(e)$$

Different Control Laws

Proportional-integral-derivative (PID) control

Pure-pursuit control

Model-predictive control (MPC)

Linear-quadratic regulator (LQR)

And many many more!

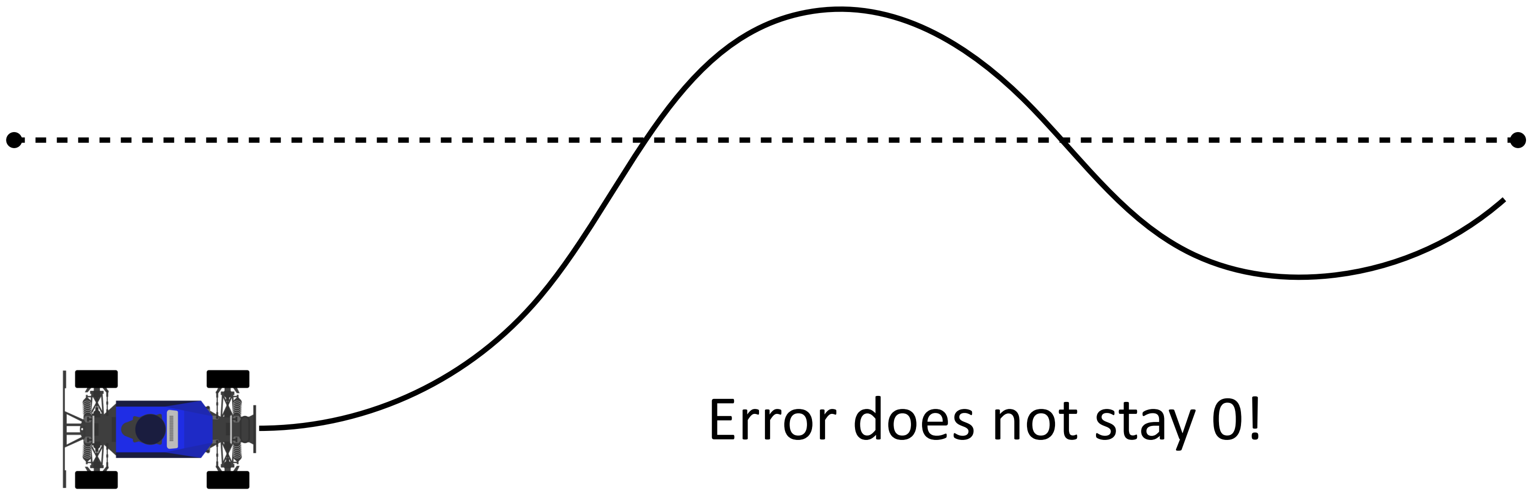
Bang-bang control

Simple control law - choose between hard left and hard right

$$u = \begin{cases} u_{max} & \text{if } e_{ct} < 0 \\ -u_{max} & \text{otherwise} \end{cases}$$

Bang-bang control

What happens when we run this control?



Error does not stay 0!

Need to adapt the magnitude of control proportional to the error …

This clearly sucks! Come back on Monday to find out more

Class Outline

State Estimation
- Robotic System Design
- Filtering
- Localization
- SLAM

Control
- Feedback Control
- PID Control
- MPC
- LQR

Planning
- Search
- Heuristic Search
- Motion Planning
- Lazy Search

Learning
- Imitation Learning
- Policy Gradient
- Actor-Critic
- Model-Based RL