Autonomous Robotics
Winter 2025

Tyler Westenbroek

TAs: Carolina Higuera, Entong Su, Bernie Zhu

Slides borrowed from many sources – Sidd Srinivasa, Sanjiban Choudhury, Russ Tedrake

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

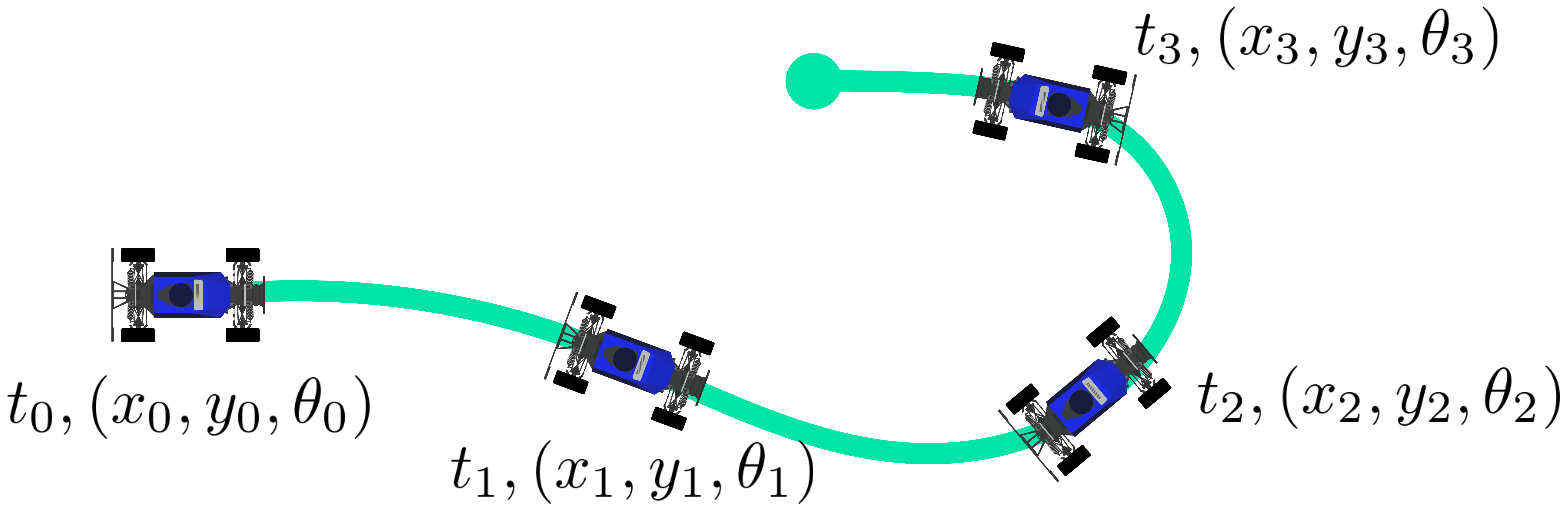
Learning

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

Recap

What is Control?

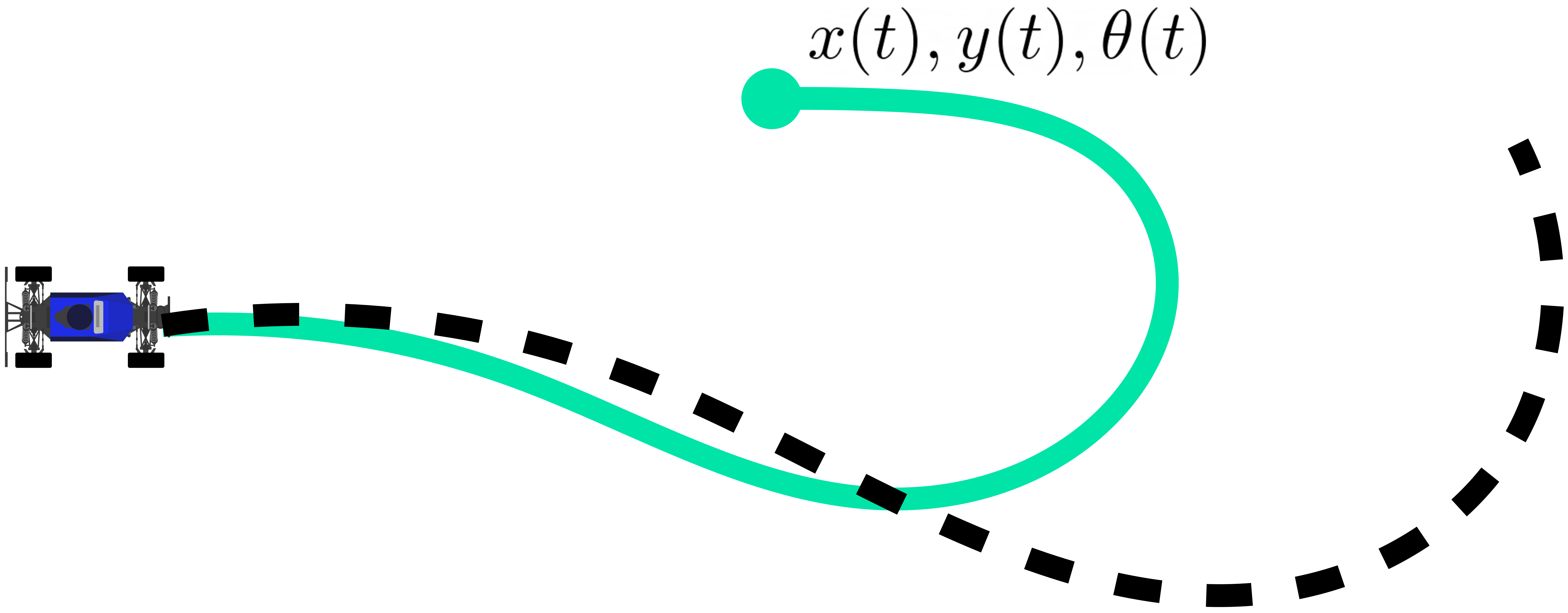
What is a Plan?



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

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

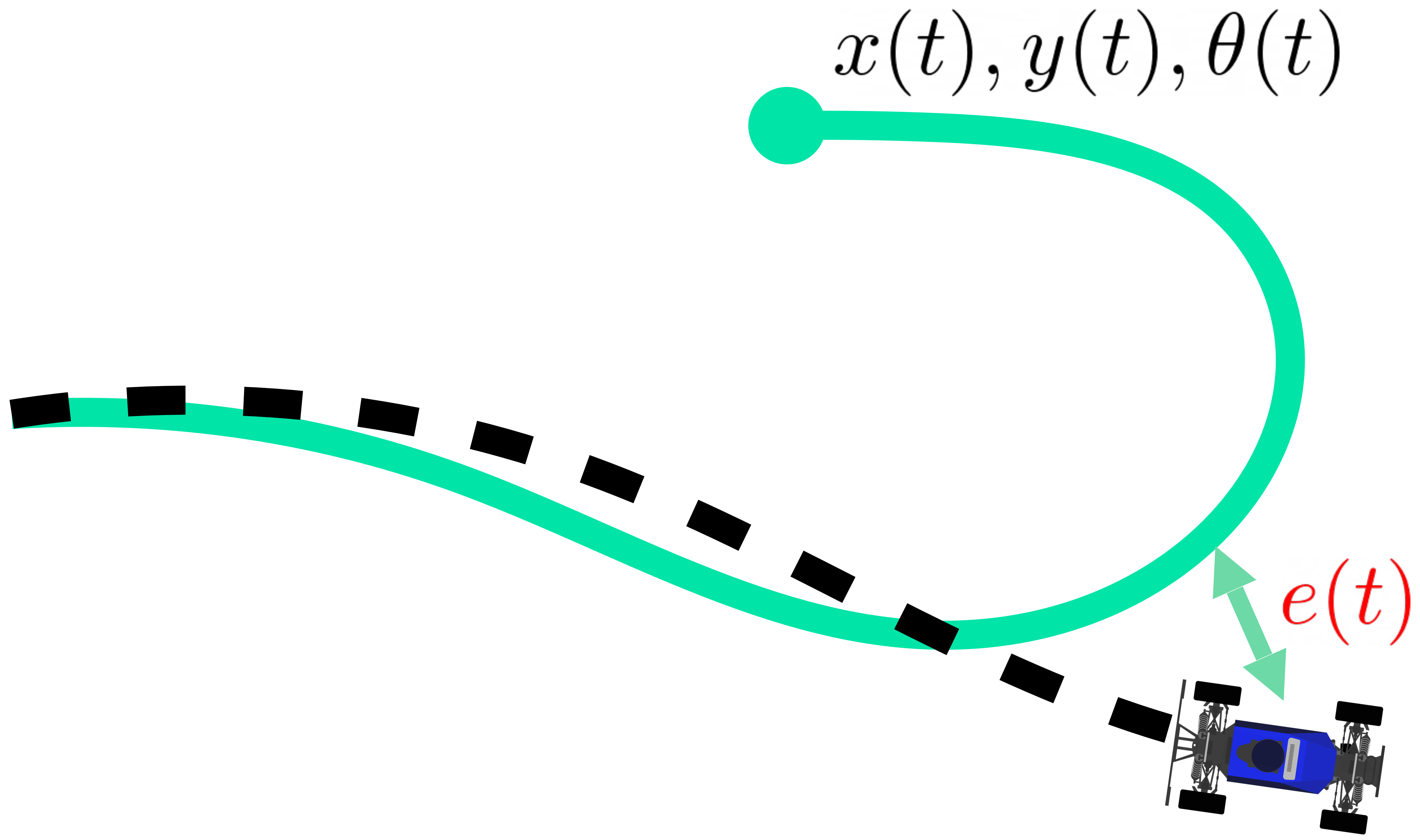
Why Feedback Control?



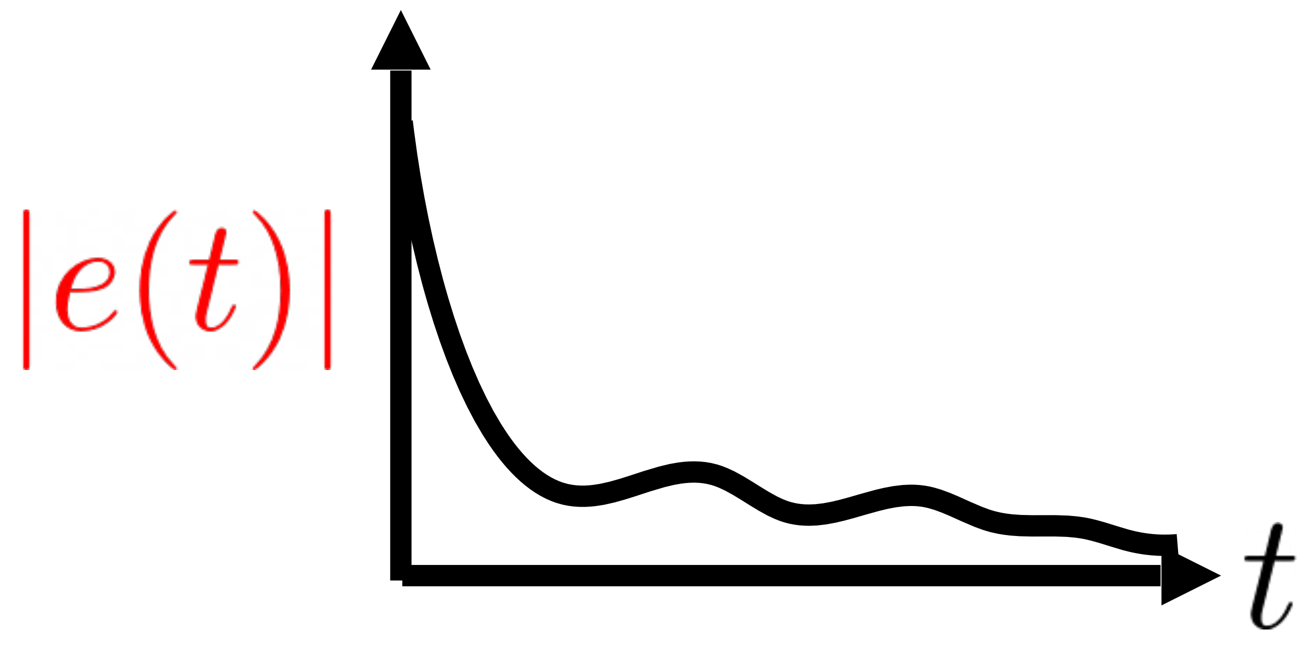
What if we send out controls $u(t)$ from kinematic car model?

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

Feedback Control



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

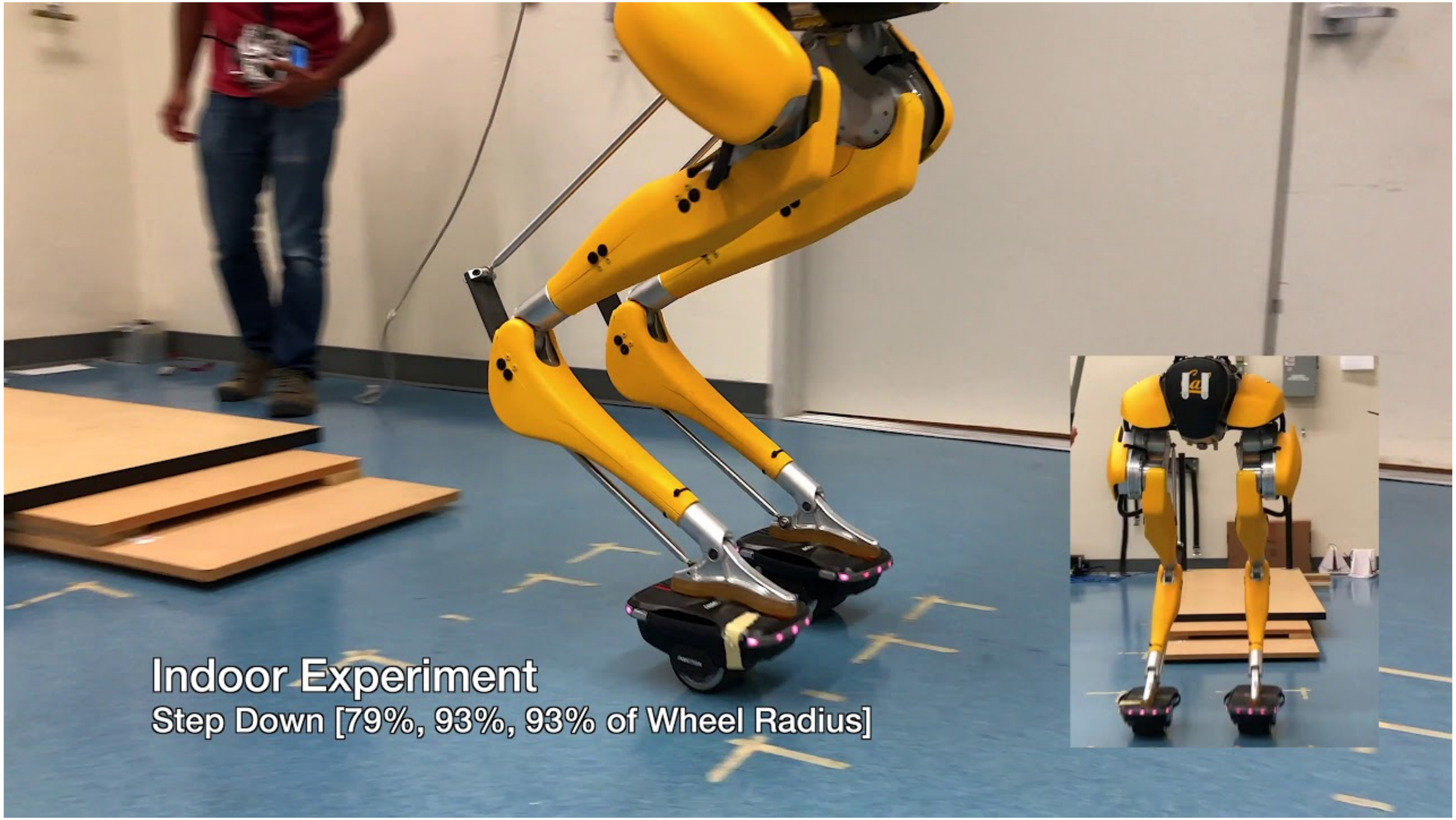


Controller Design Decisions

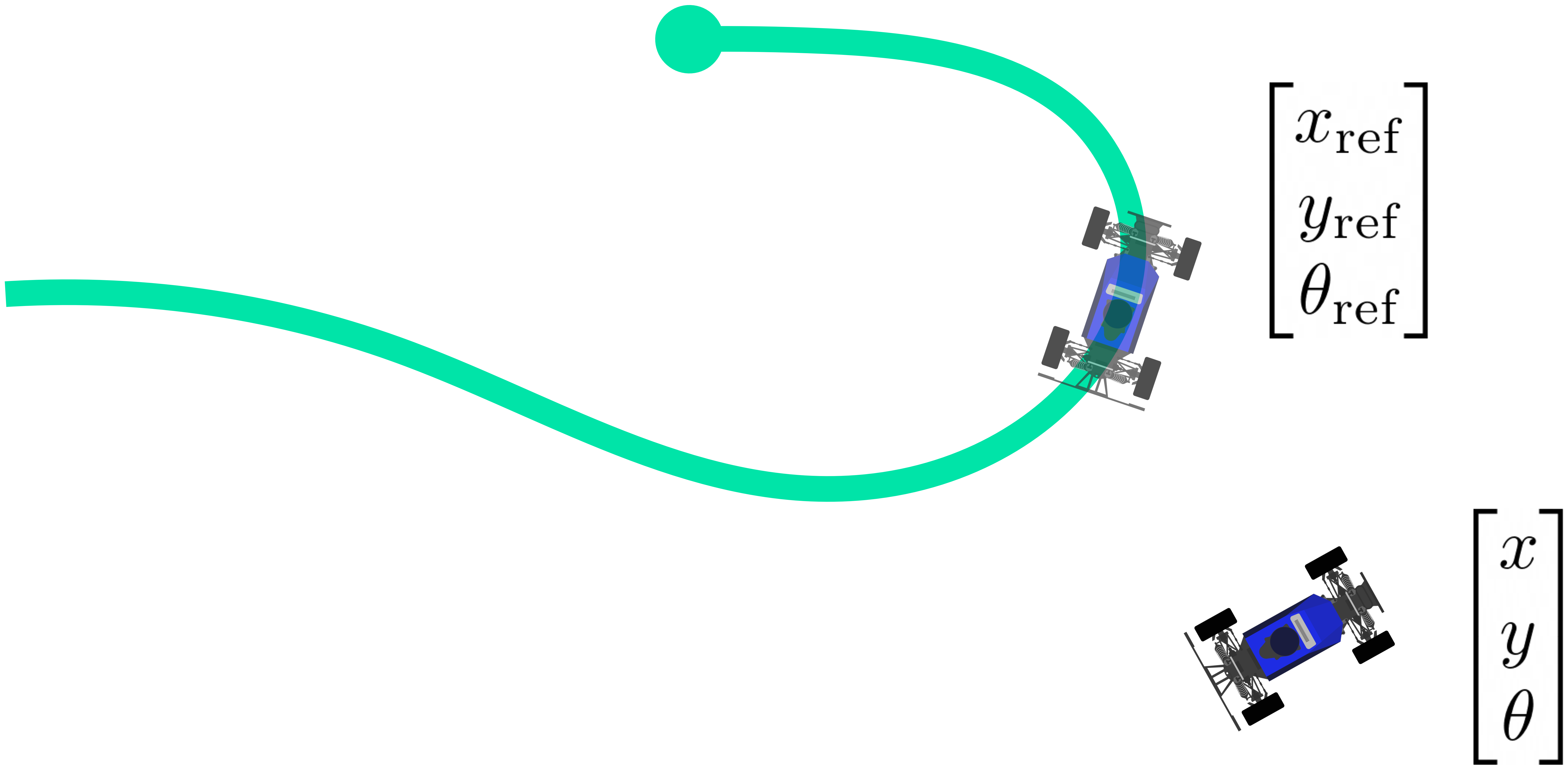
1. Get a reference path/trajectory to track
2. Pick a reference state from the reference path/trajectory
3. Compute error to reference state
4. Compute control law to minimize error

These design decisions are <u>extremely</u> coupled —
there's never one right answer!

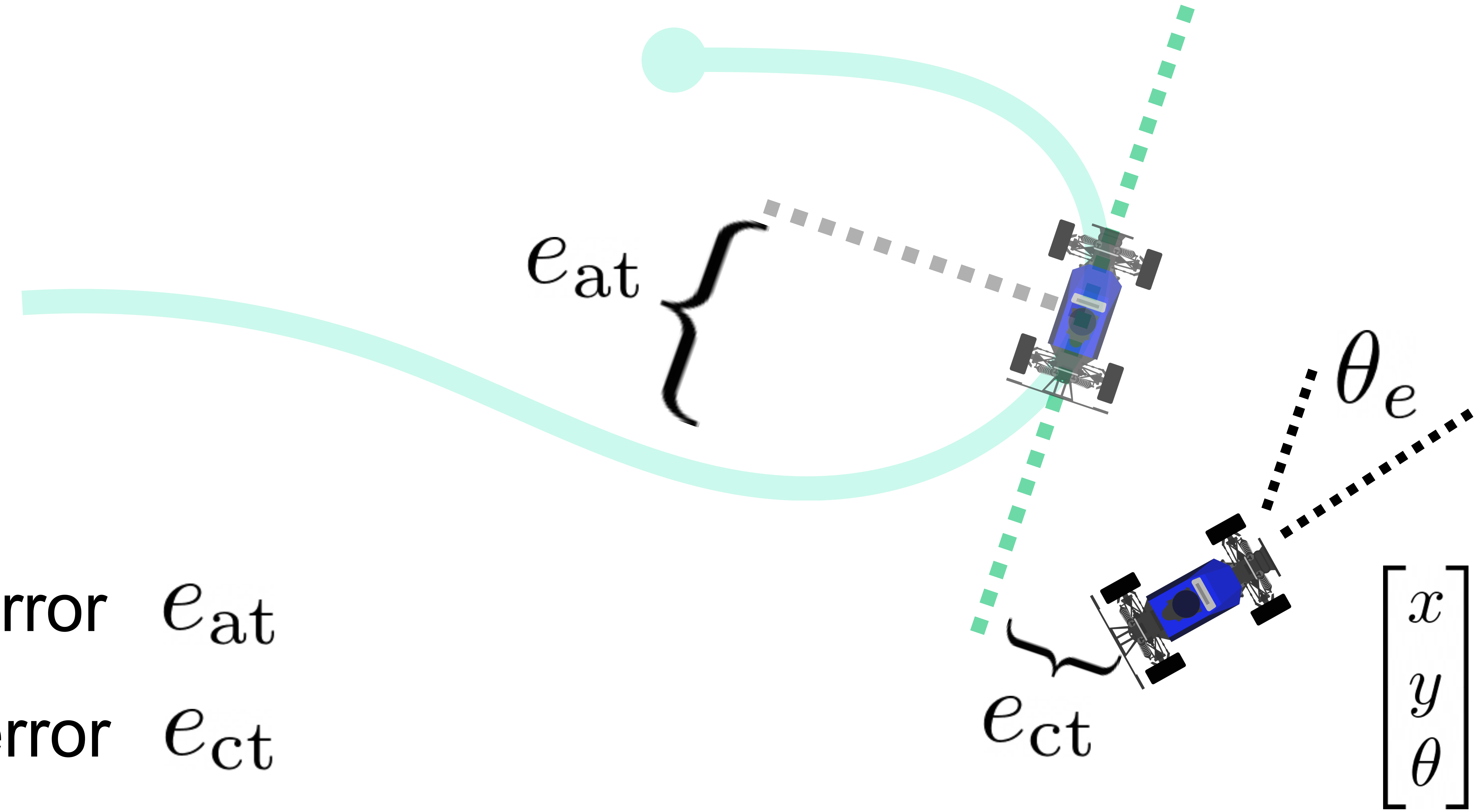
Basic idea scale to complicated systems!



Step 2: Pick a reference (desired) state



Step 3: Compute error to reference state



Along-track error e_{at}

Cross-track error e_{ct}

Heading error θ_e

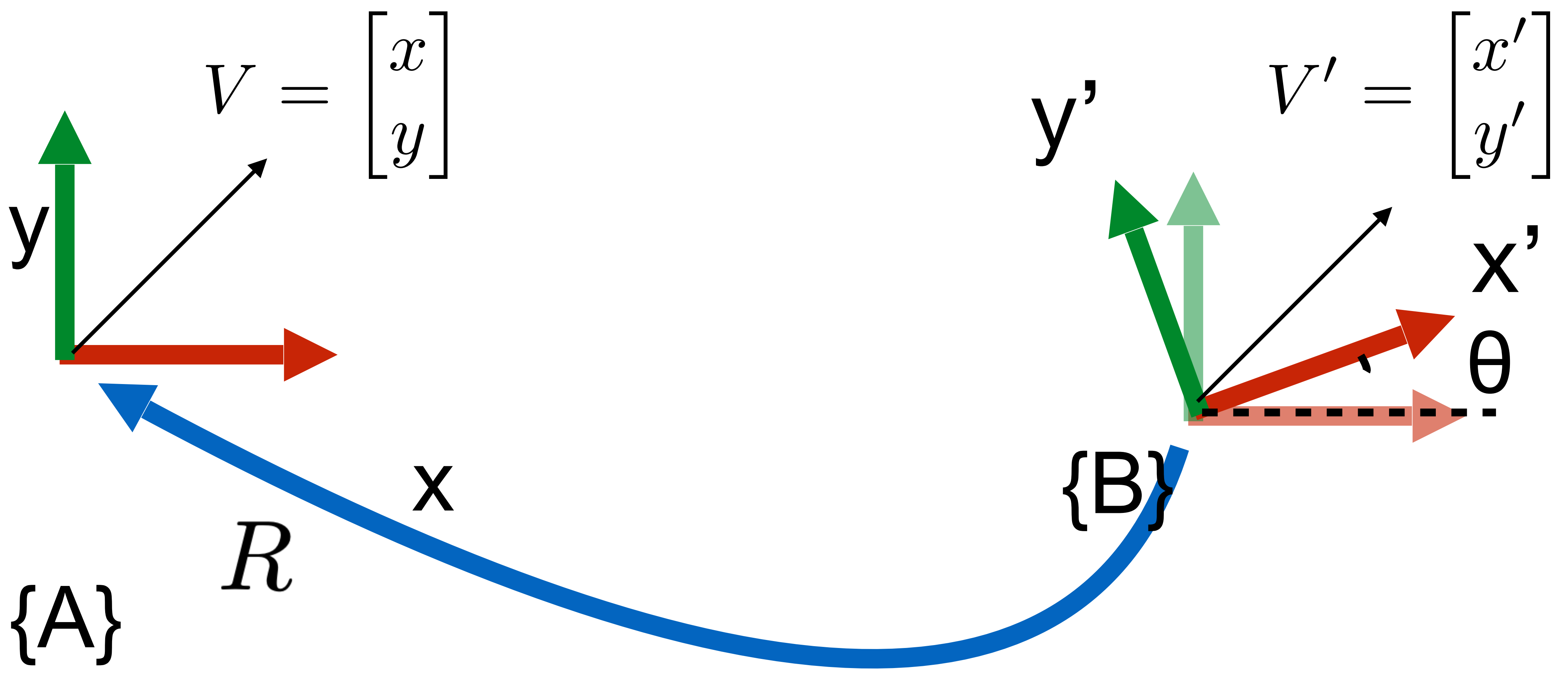
Step 3: Compute error to reference state

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

Think of tracking error as a change in coordinates!

How to compute this error systematically?

Aside: Rotation Matrices (Plane)

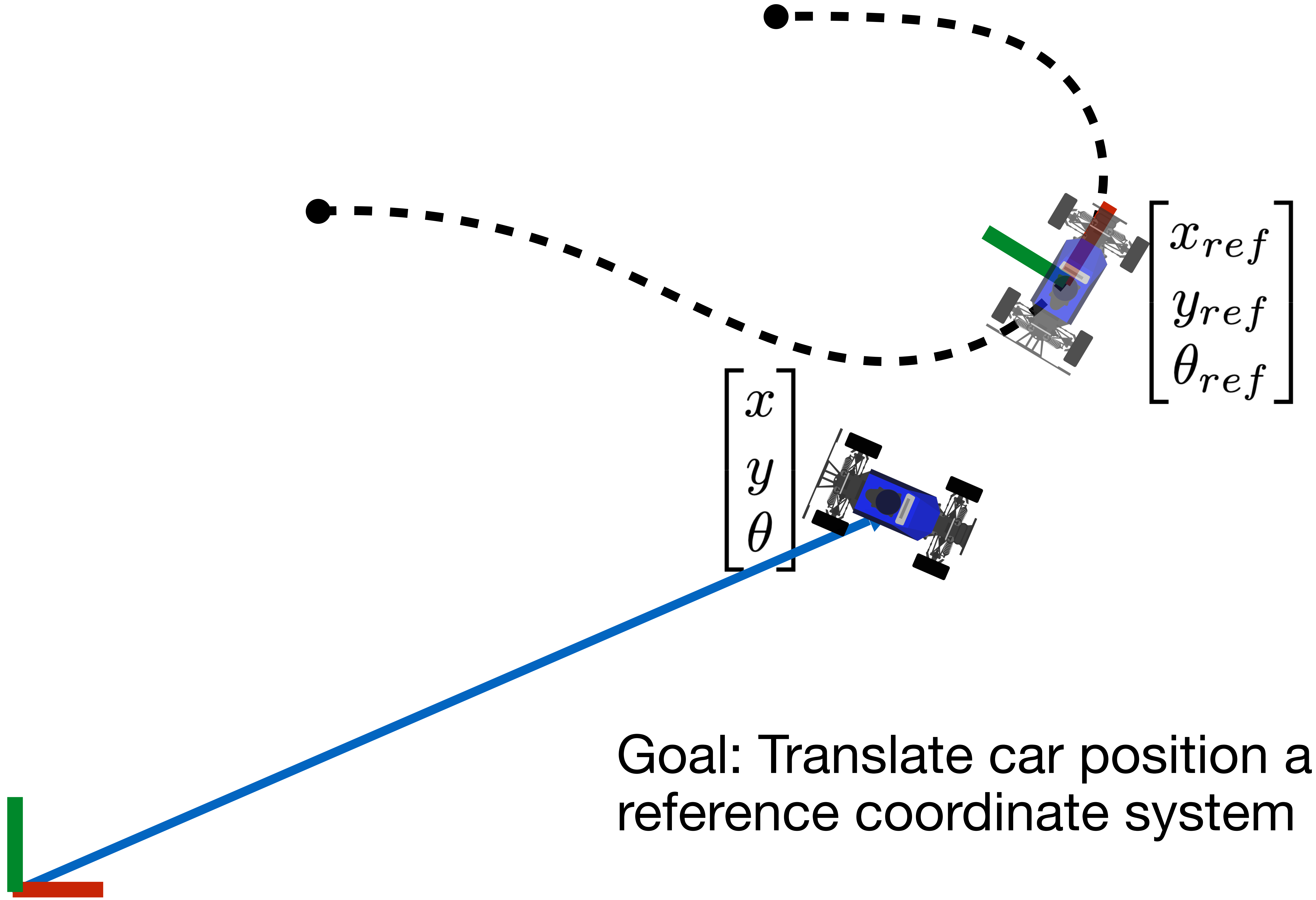


$$R = R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \underbrace{R(\theta)}_{{}^A_B R} \begin{bmatrix} x' \\ y' \end{bmatrix}$$

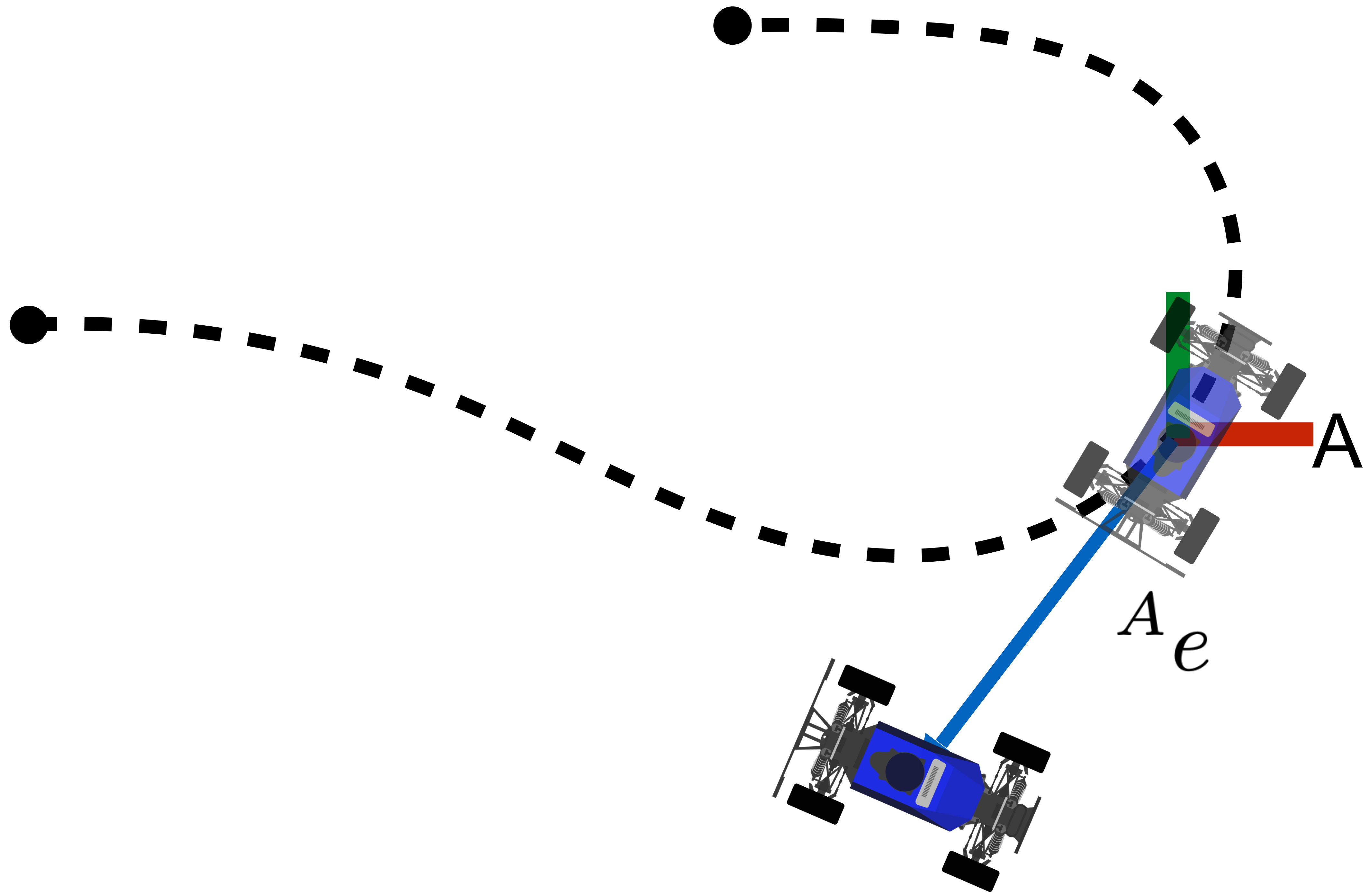
(rotation of B w.r.t A)

Step 3: Compute error to reference state



Goal: Translate car position and heading into reference coordinate system

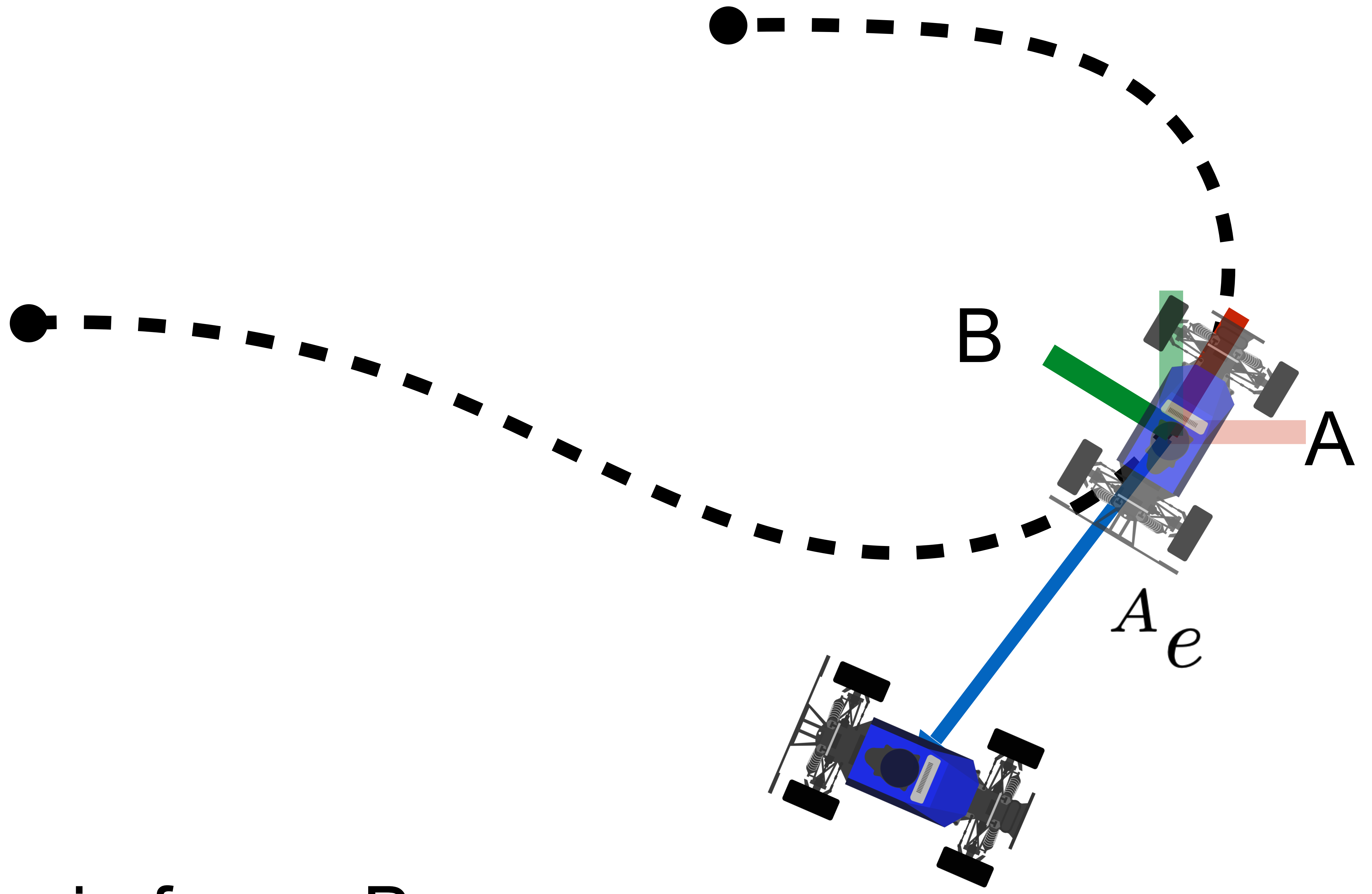
Step 3: Compute error to reference state



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



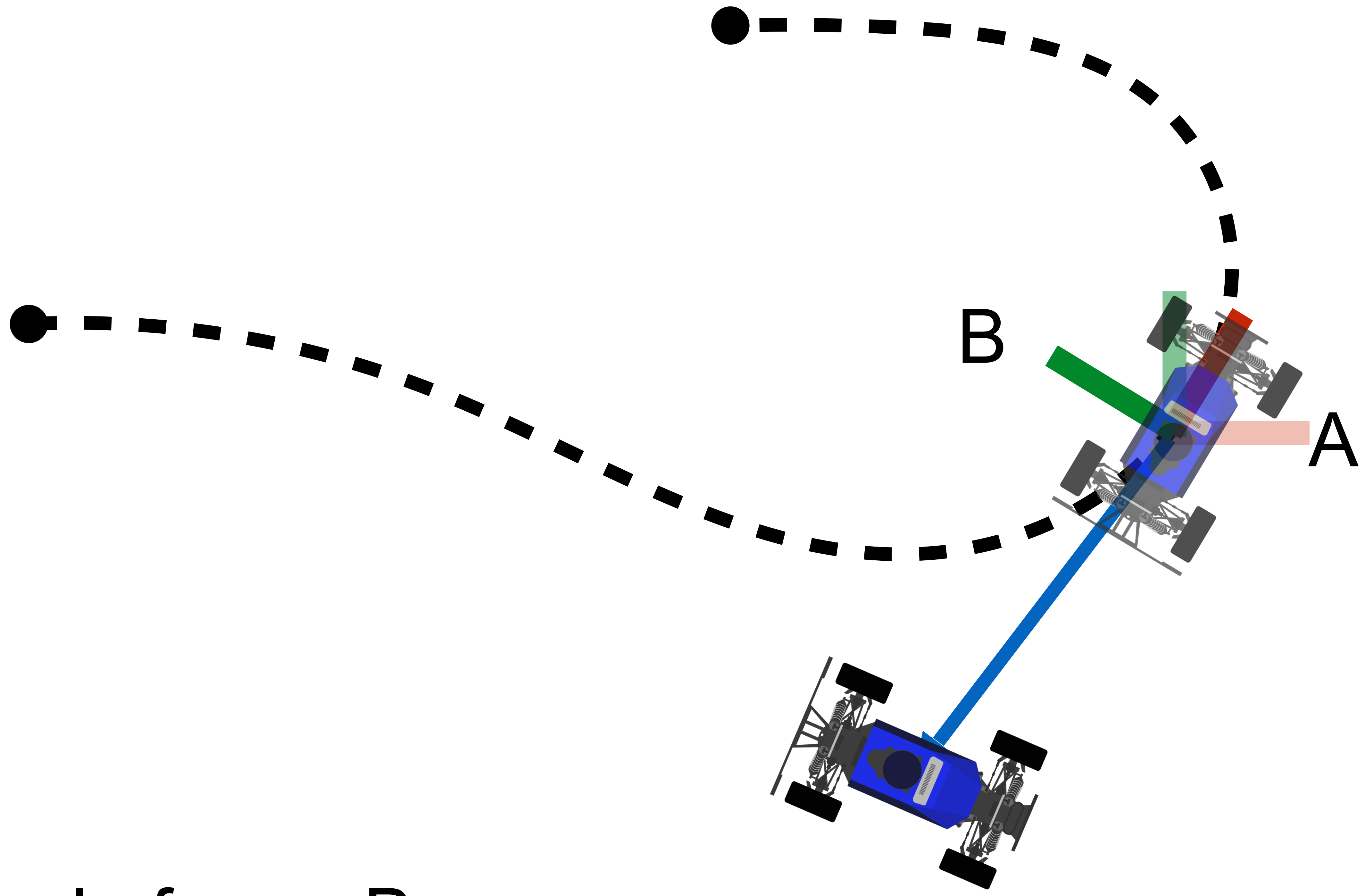
We want position in frame B

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

(rotation of A w.r.t B)

(rotation of A w.r.t B)

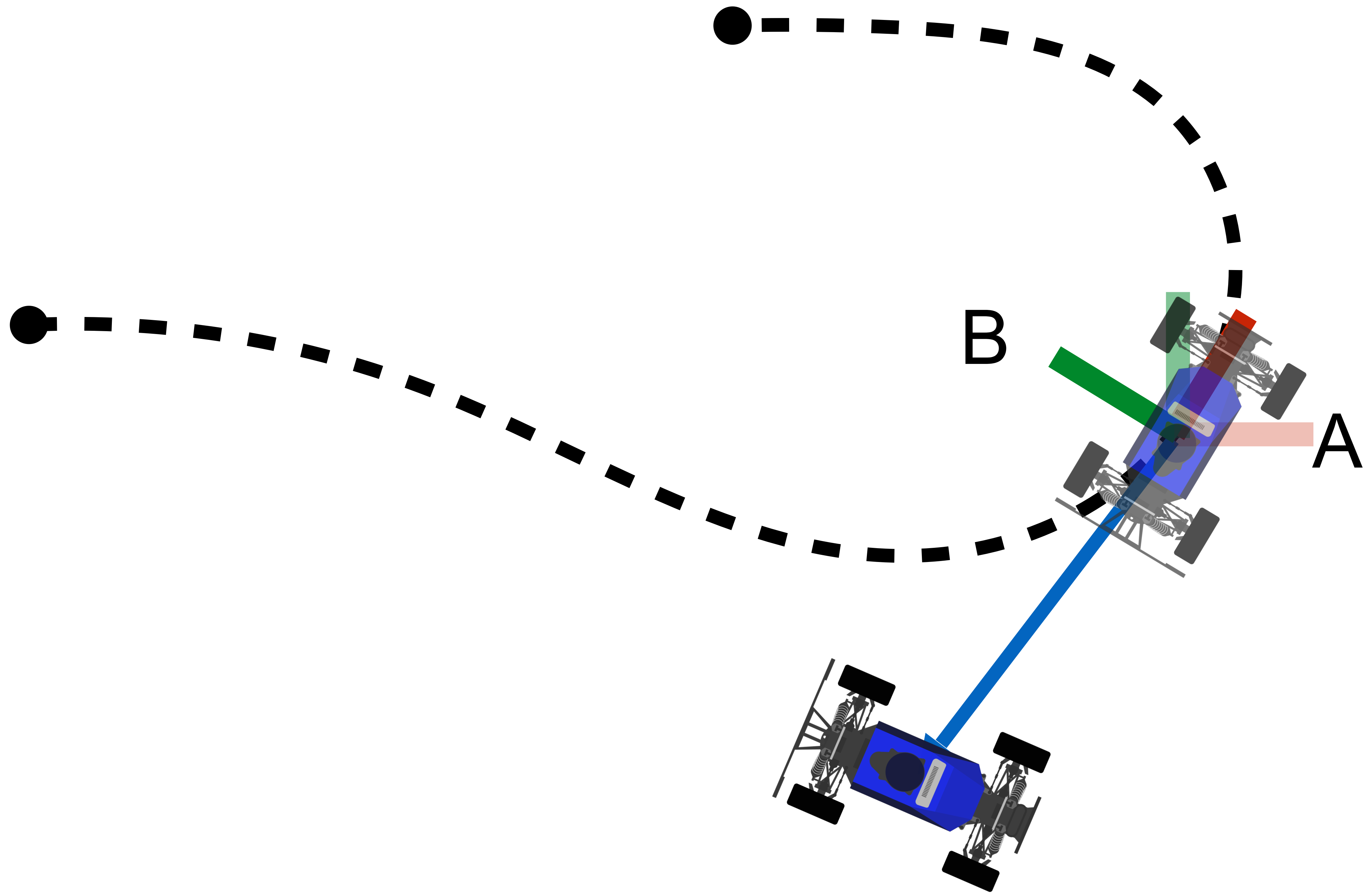
Step 3: Compute error to reference state



We want position in frame B

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

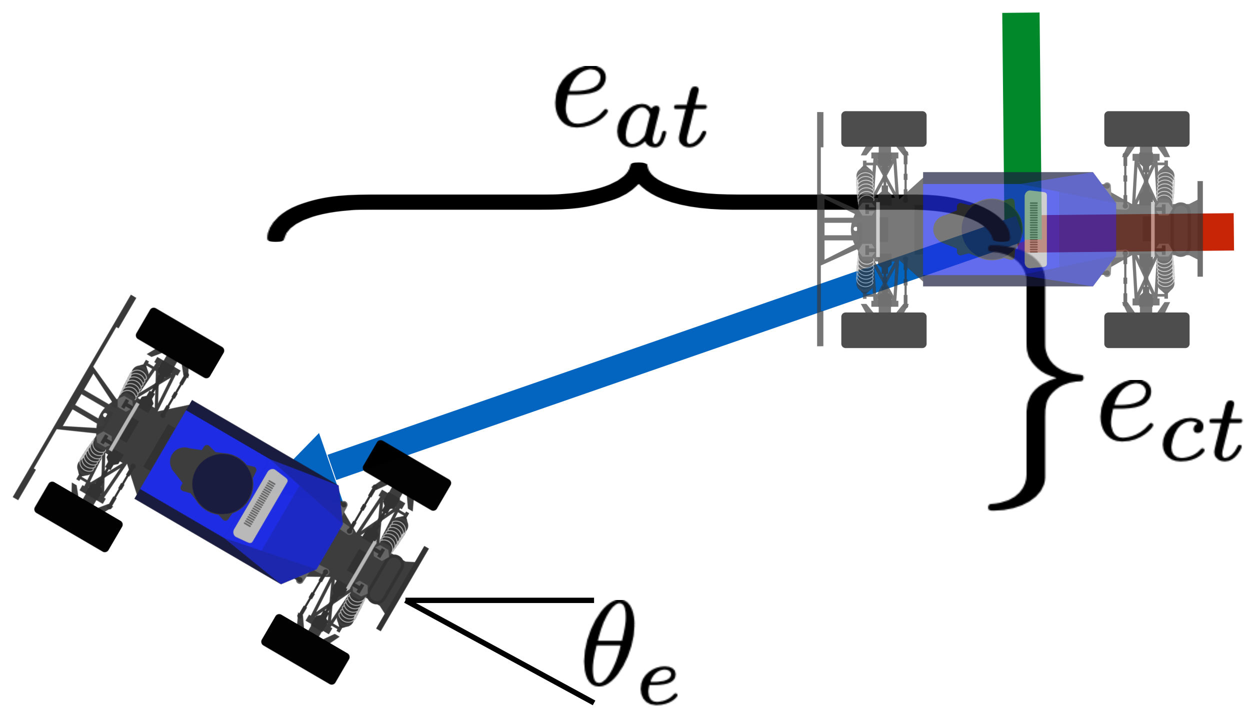
Step 3: Compute error to reference state



Heading error

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

Step 3: Compute error to reference state

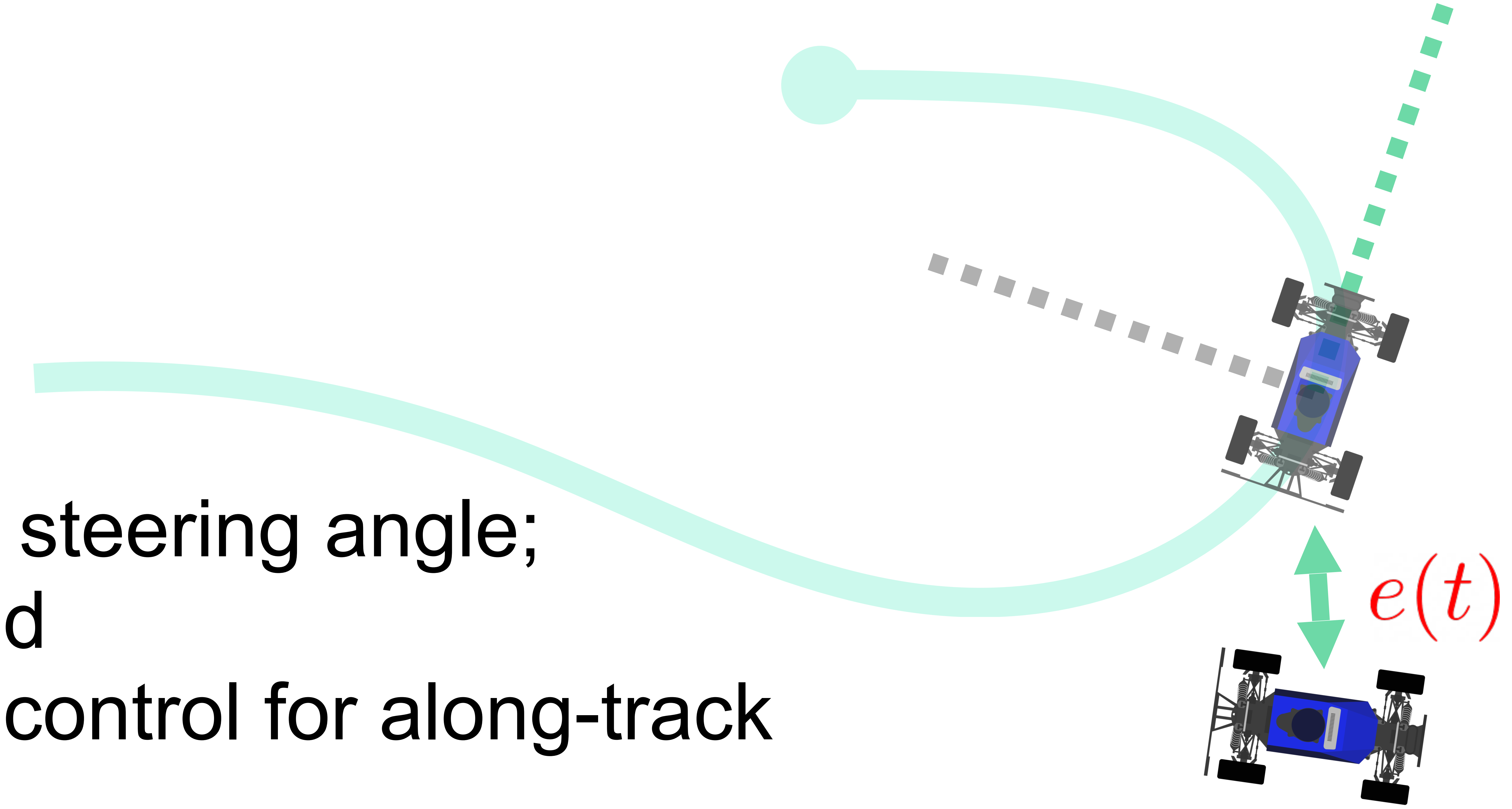


(Along-track) $e_{at} = \cos(\theta_{ref})(x - x_{ref}) + \sin(\theta_{ref})(y - y_{ref})$

(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

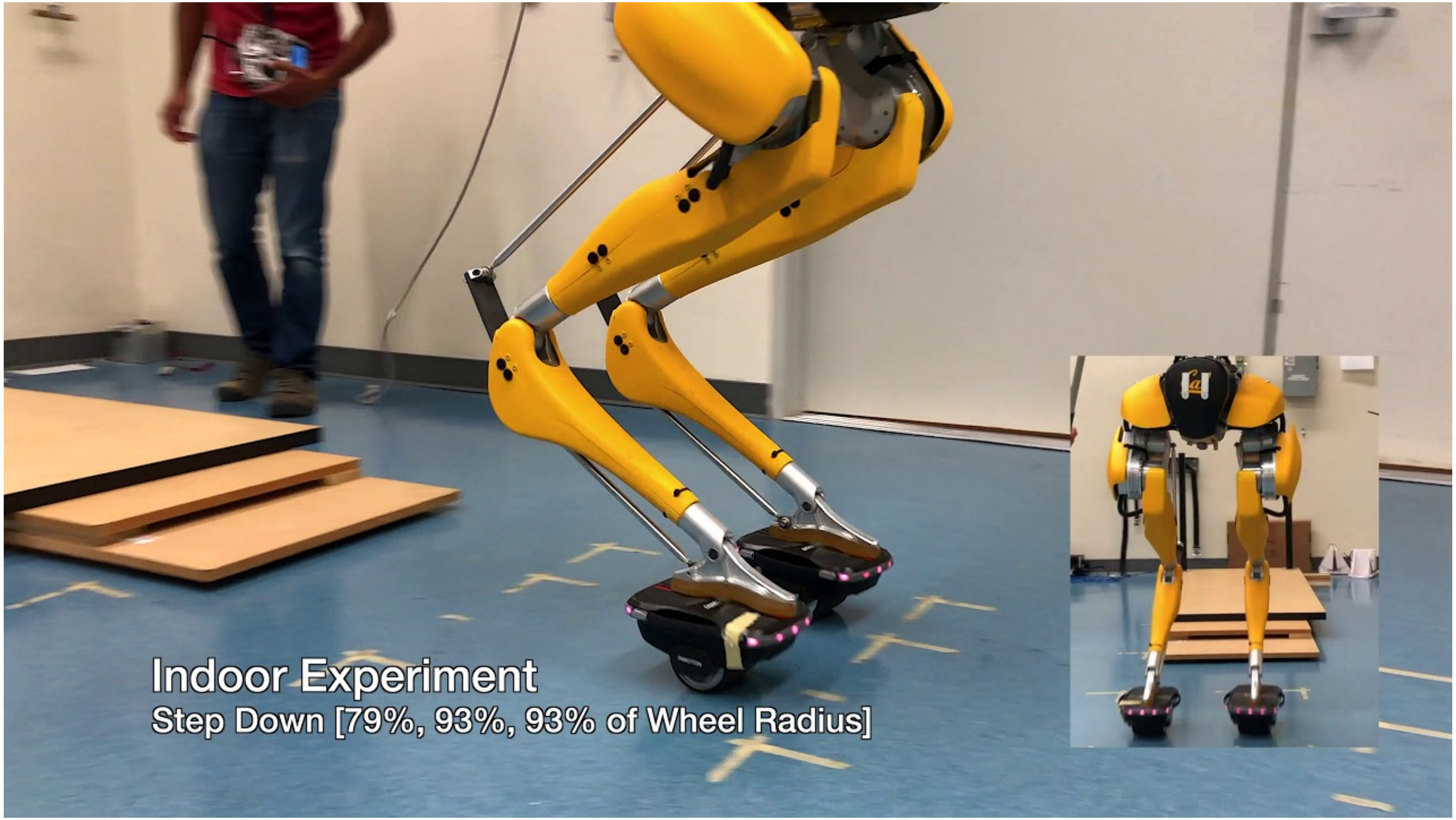


We will only control steering angle;
fixed constant speed
As a result, no real control for along-track error
Some control laws will only minimize cross-track error, others will also minimize heading

$$u = K(e)$$

The more things you want to control the more complicated it gets!

Basic idea scale to complicated systems!



Step 4: Compute control law

Compute control action based on instantaneous error

$$u = K(\mathbf{x}, e)$$

control

state error

(steering angle, speed)

Apply control action, robot moves a bit, compute new error, repeat

Different laws have different trade-offs

Different Control Laws

Proportional-integral-derivative (PID) control

Pure-pursuit control

Model-predictive control (MPC)

Linear-quadratic regulator (LQR)

And many many more!

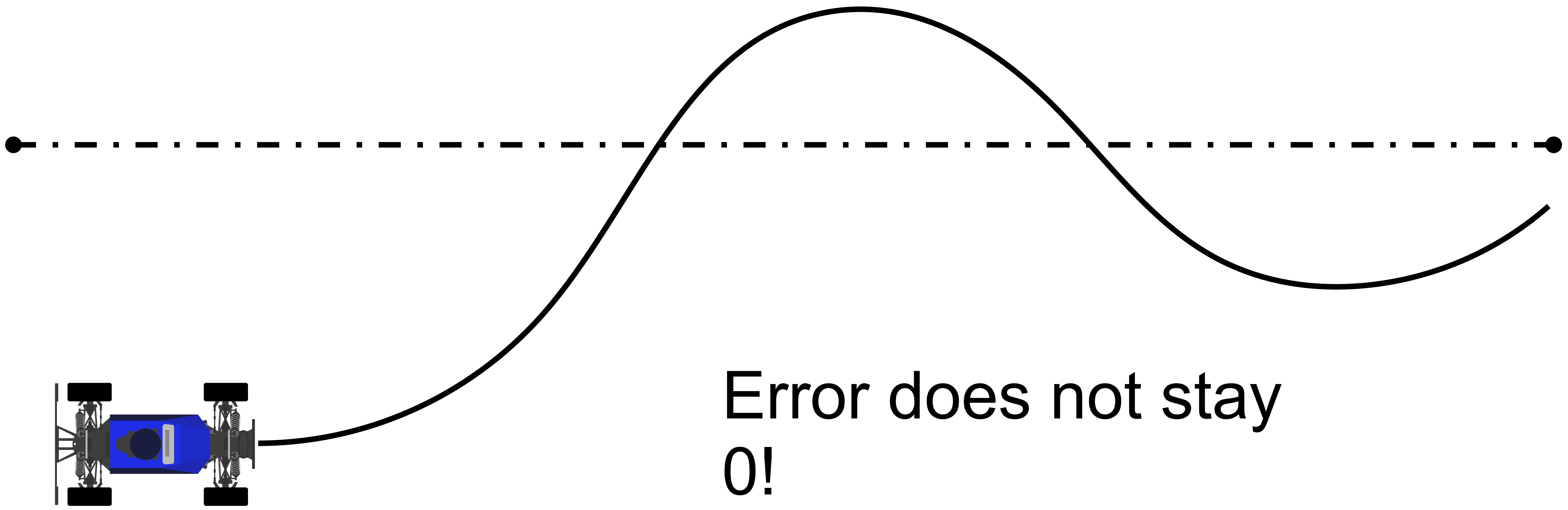
Bang-bang control

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?

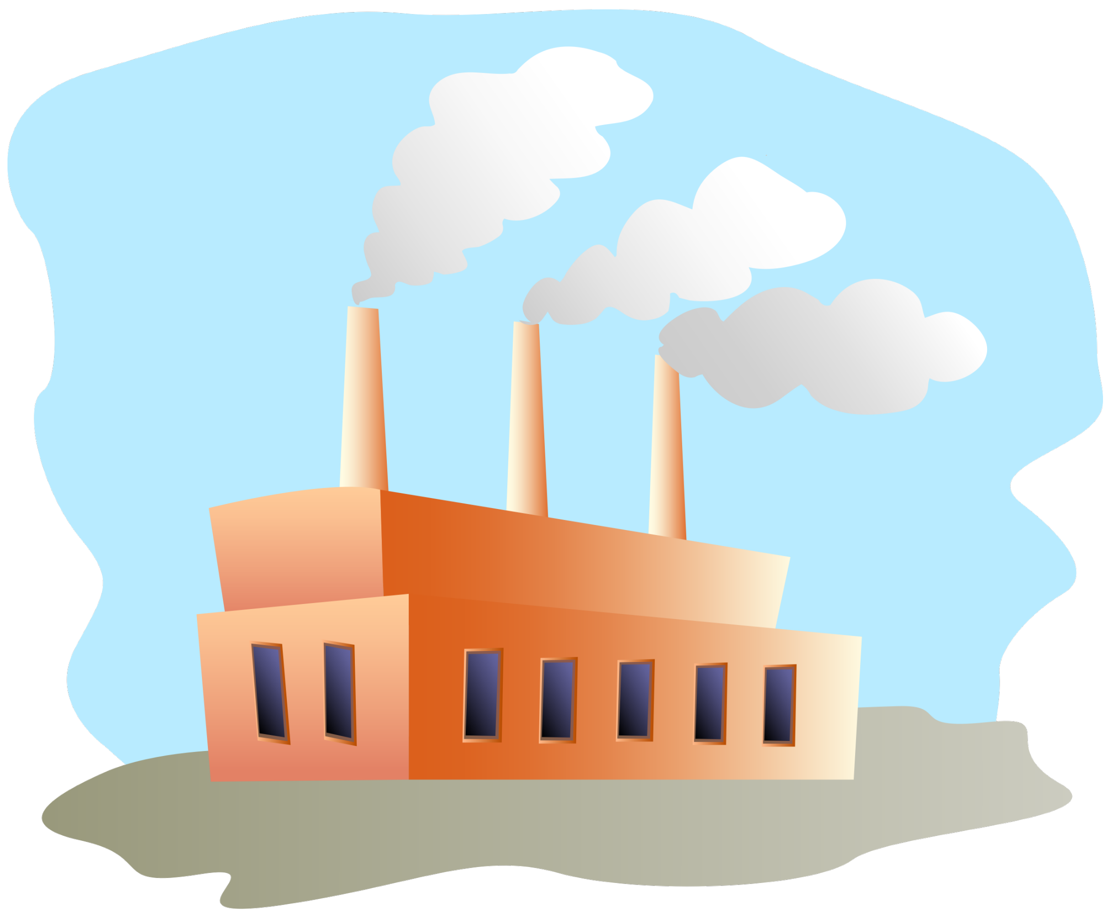


Error does not stay 0!

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

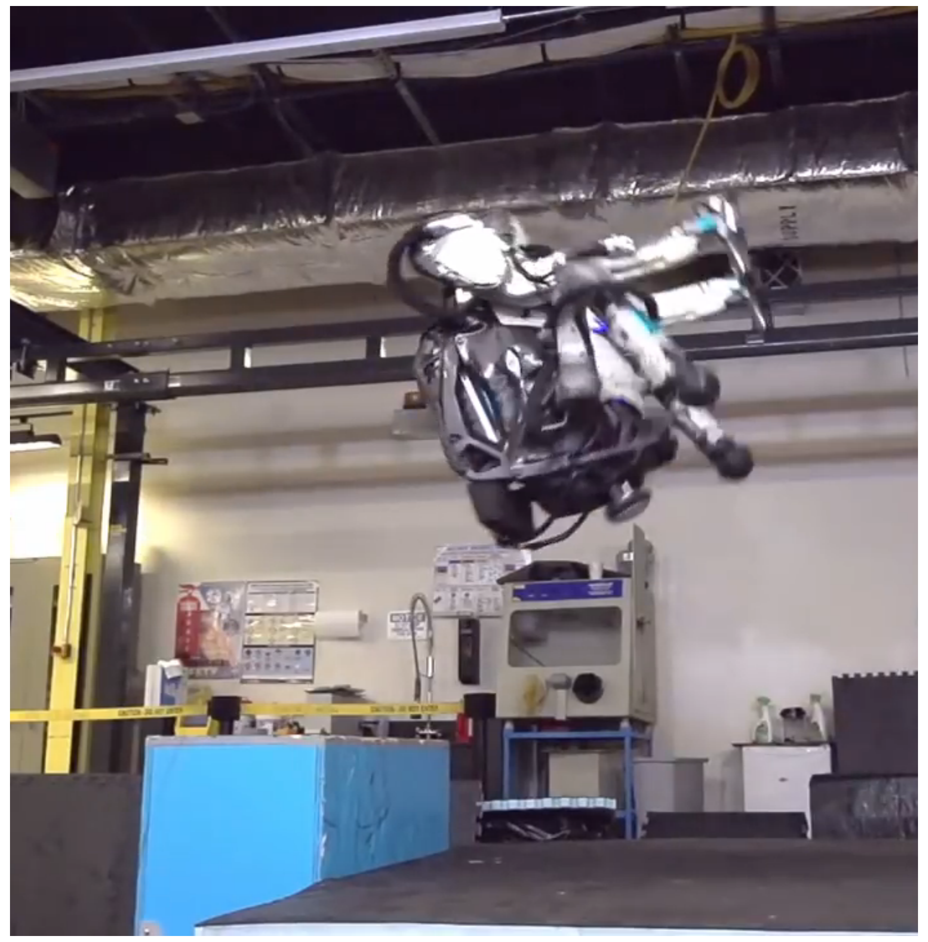
This clearly sucks! How can we do better?

PID controllers



Used widely in
industrial
control from 1900s

Regulate temp, press, speed
etc



Do not try
this
with PID!!!

PID control overview

Select a control law that tries to drive error to zero (and keep it there)

$$u = -\left(K_p e_{ct} + K_i \int e_{ct}\, dt + K_d \dot{e}_{ct}\right)$$

PROPORTIONAL (PRESENT)

INTEGRAL (PAST)

DERIVATIVE (FUTURE)

PID Intuition

$$u = -\left(K_p e_{\mathrm{ct}} + K_i \int e_{\mathrm{ct}} dt + K_d \dot{e}_{\mathrm{ct}} \right)$$

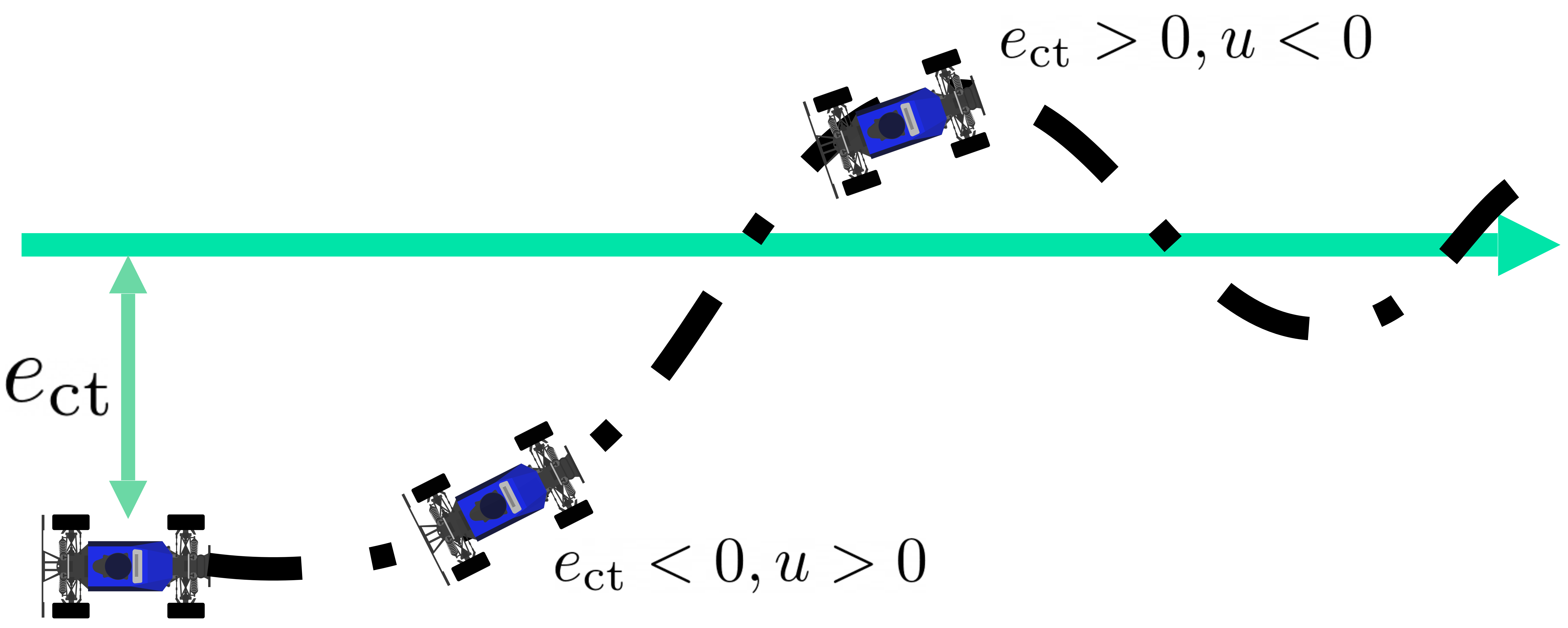
PROPORTIONAL	**INTEGRAL**	**DERIVATIVE**
(PRESENT)	(PAST)	(FUTURE)

Proportional: minimize the current error!

Integral: if I’m accumulating error, try harder!

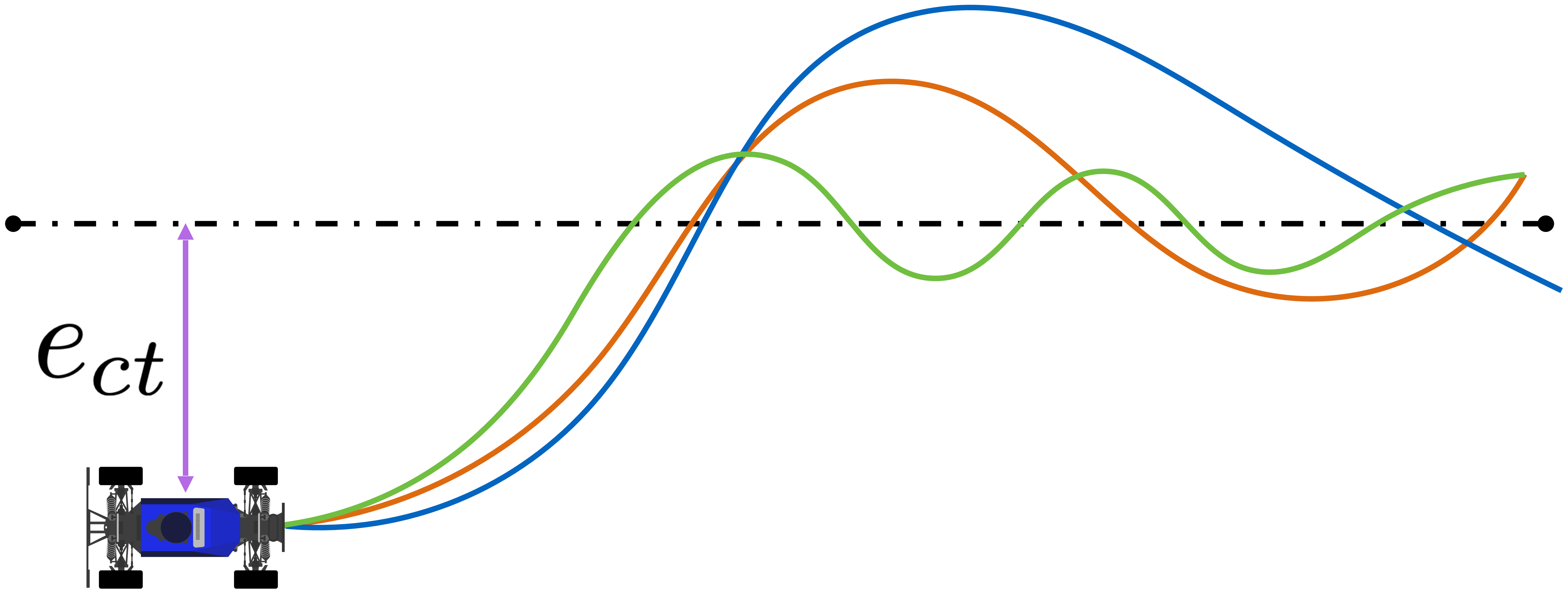
Derivative: if I’m going to overshoot, slow down!

Proportional Control



$$u = -K_p e_{\text{ct}}$$

The proportional gain matters!



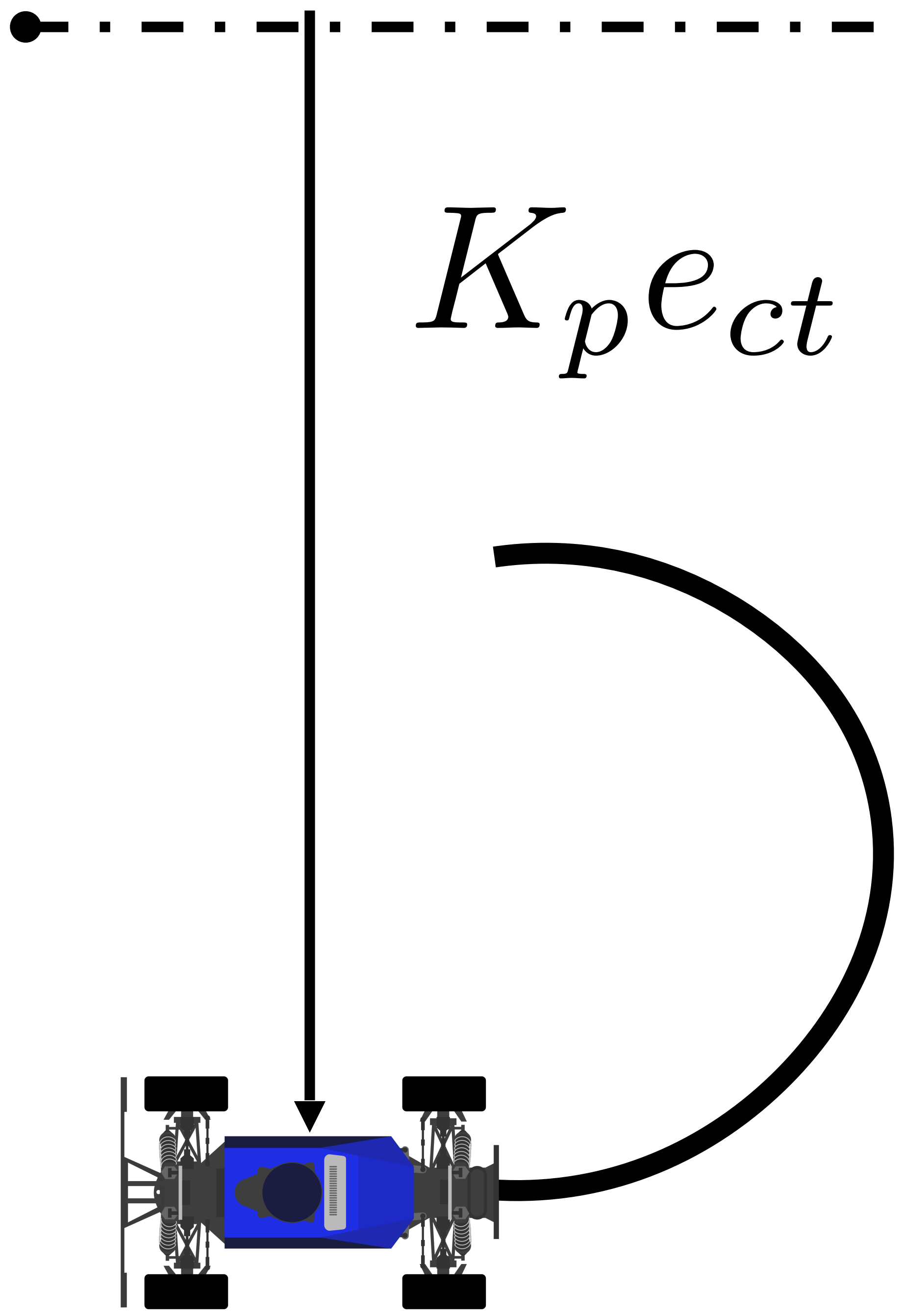
What happens when gain is low?

What happens when gain is high?

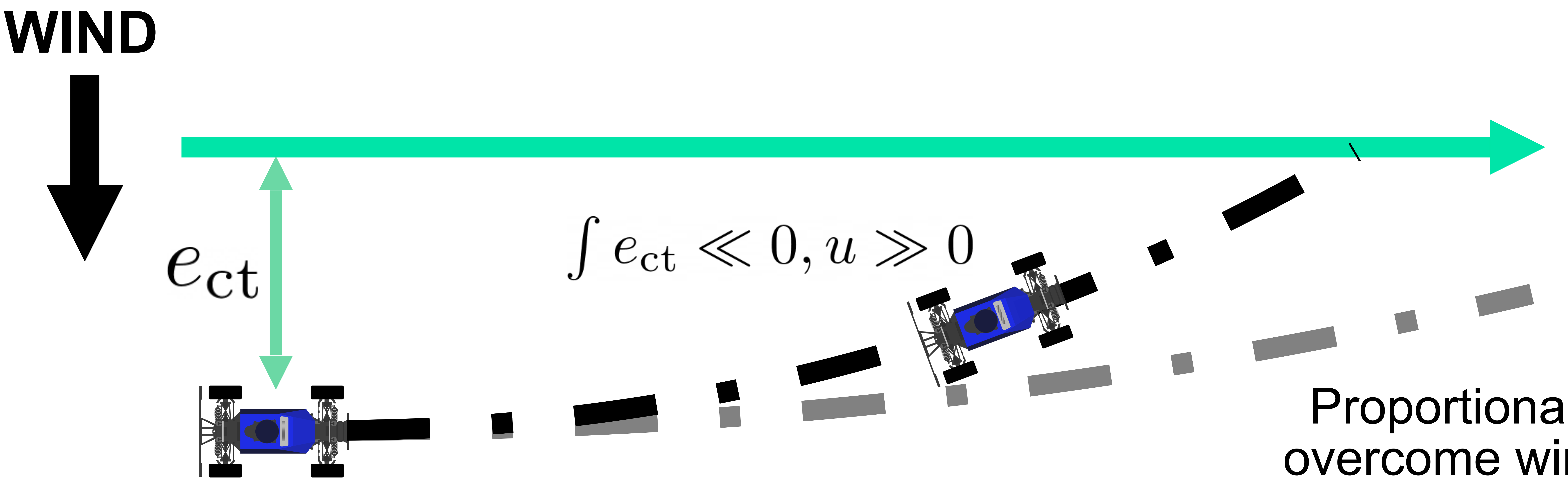
Proportional term

What happens when gain is too high?

$$K_p e_{ct} \gg u_{max}$$



Proportional Integral (PI) Control



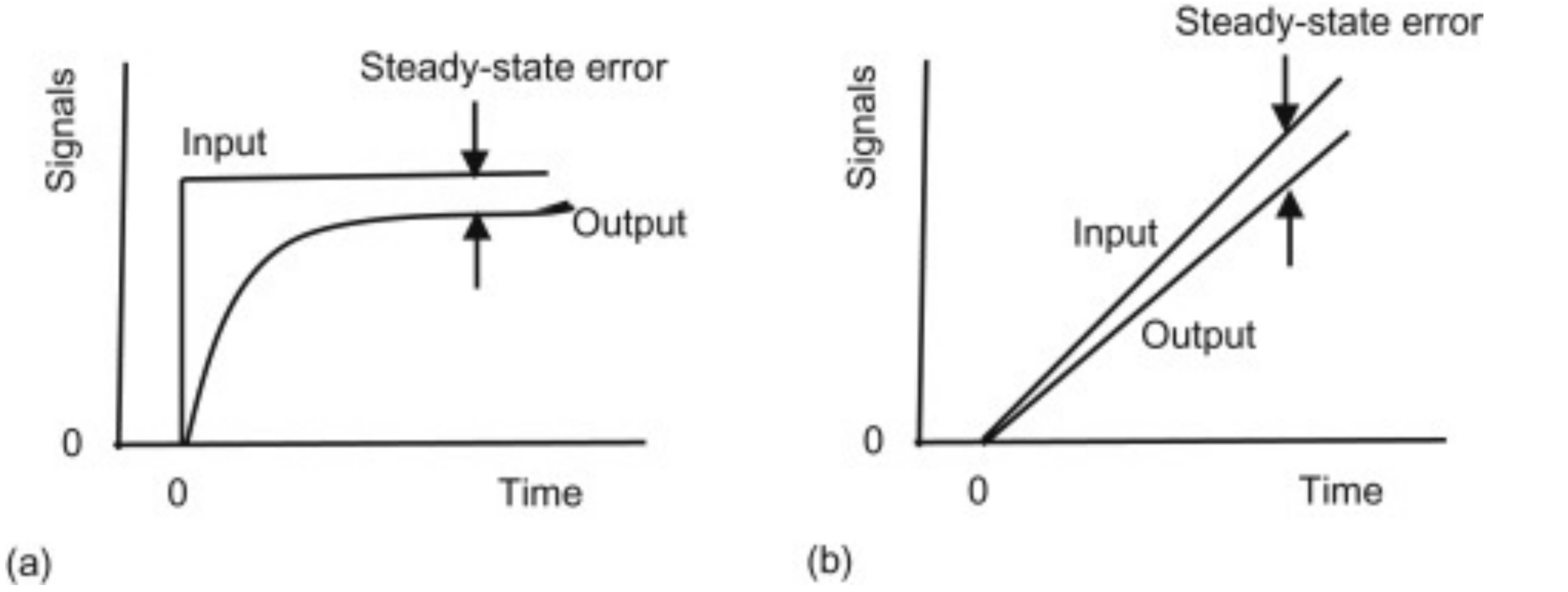
Proportional cannot
overcome wind alone!

$$u = -\left(K_p e_{ct} + K_i \int e_{ct} dt\right)$$

Proportional Integral (PI) Control

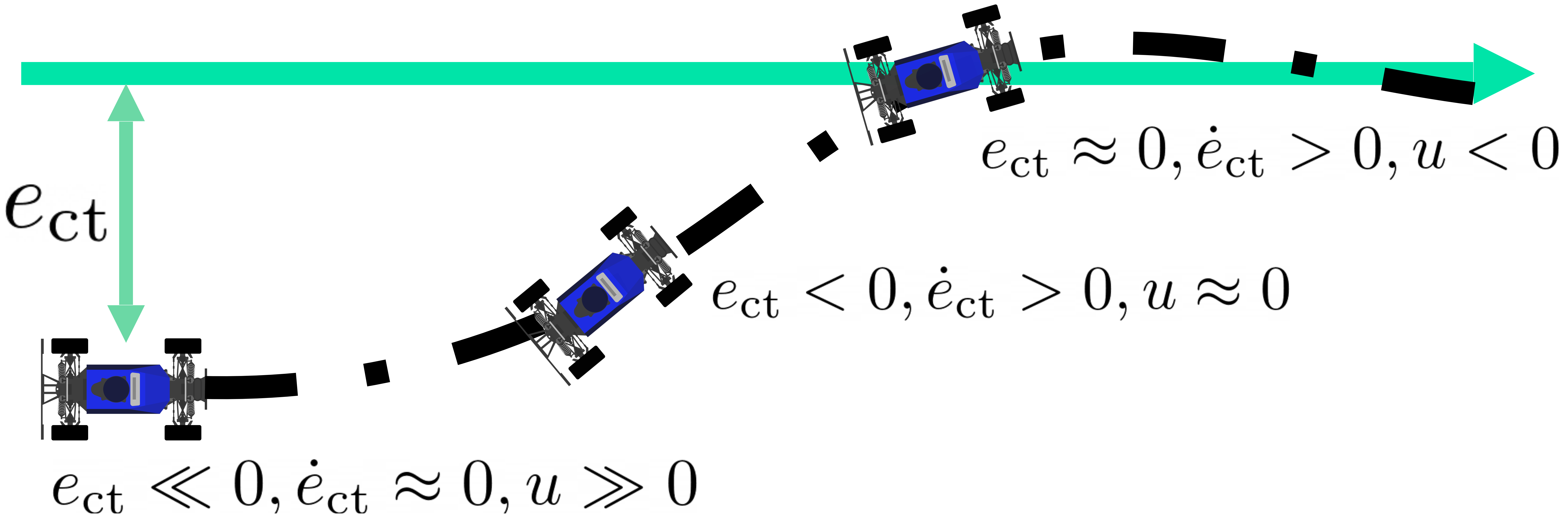
$$u = -\left(K_p e_{\mathrm{ct}} + K_i \int e_{\mathrm{ct}}\,dt\right)$$

Integral control gets rid of this term since the integral keeps growing



Proportional Derivative (PD) Control

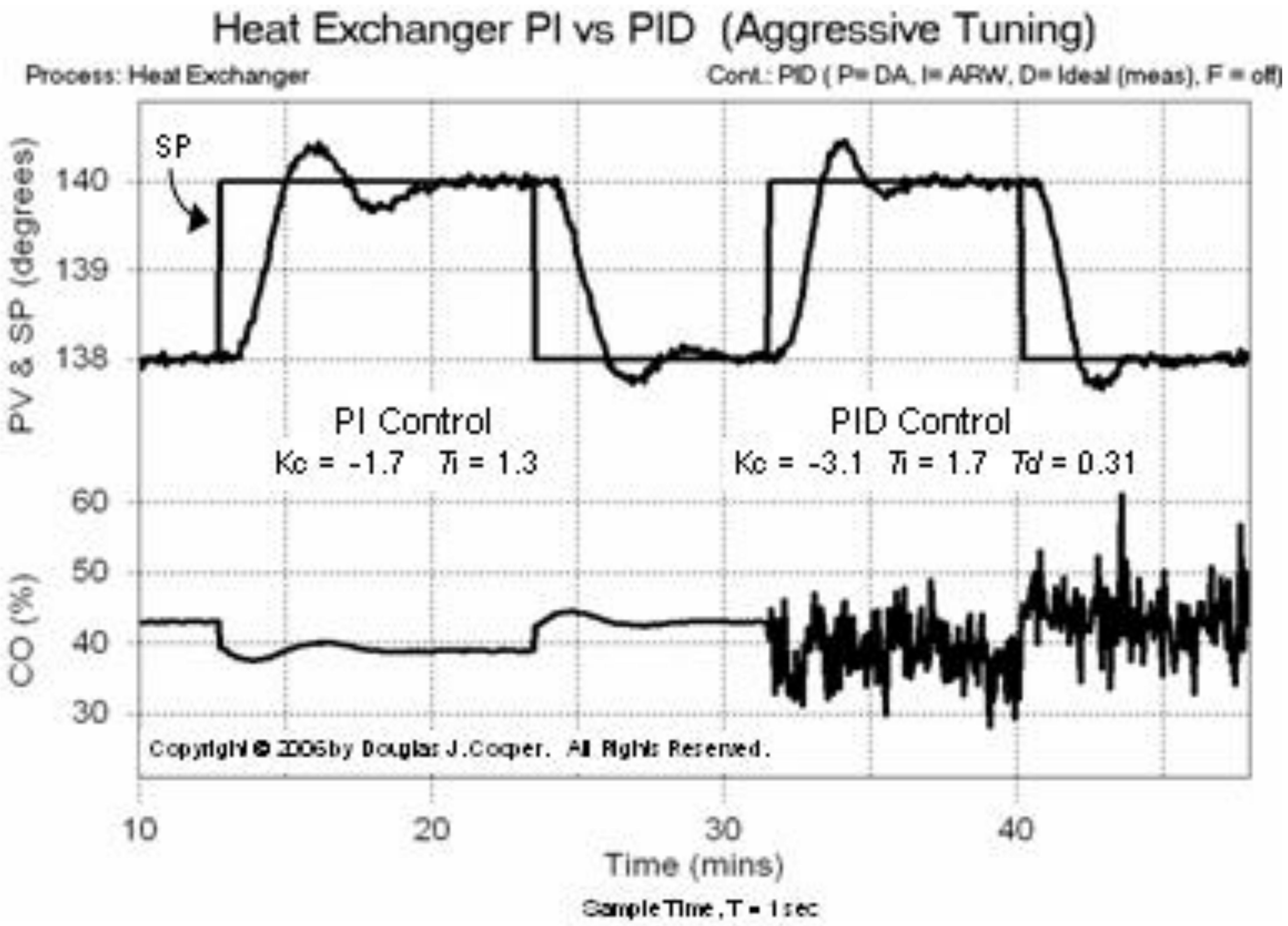
Apply the brakes when moving too fast! [?] converge to the steady state

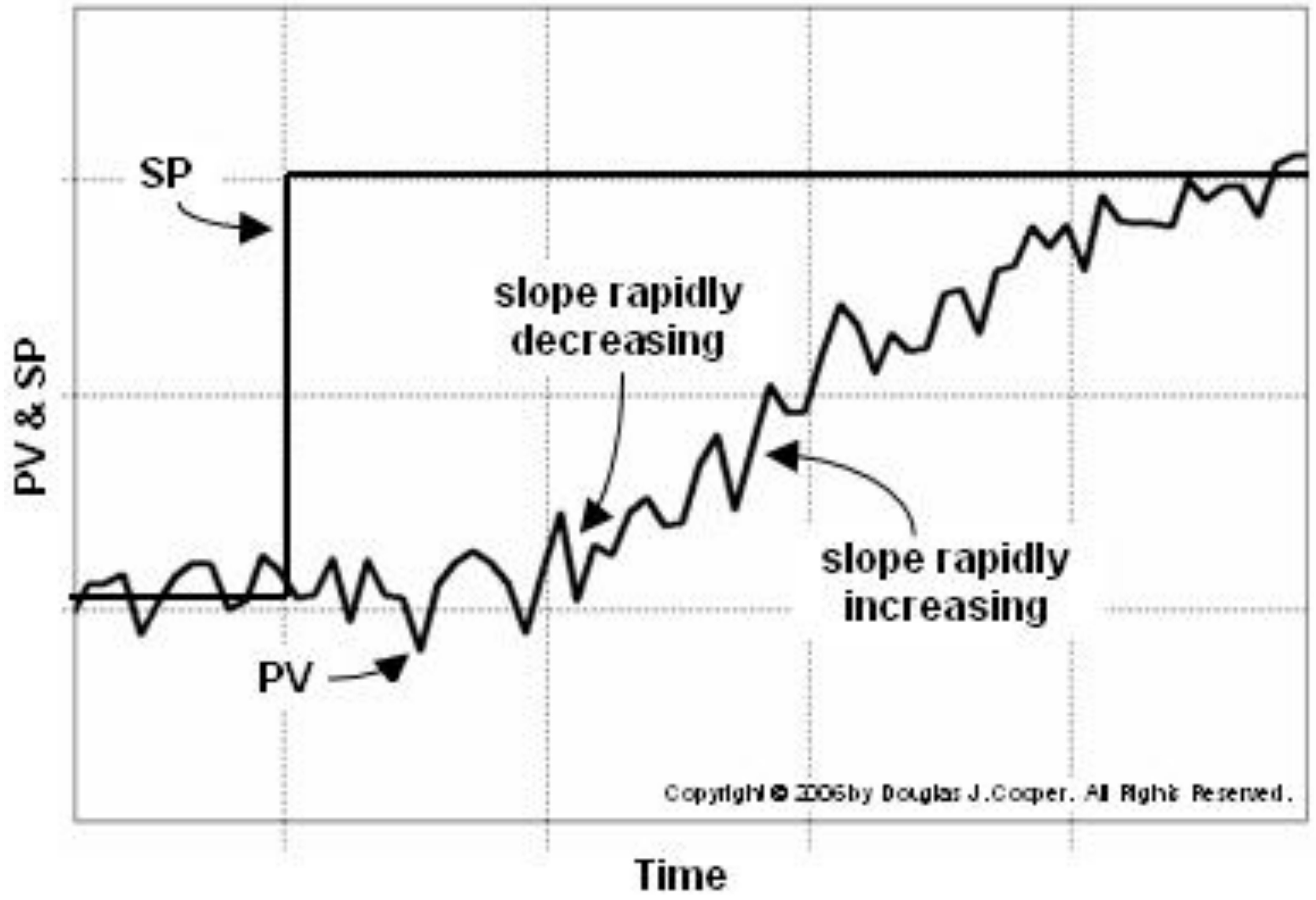


$$u = -(K_p e_{\text{ct}} + K_d \dot{e}_{\text{ct}})$$

Challenges with using the derivative term

Noise can lead to wildly changing derivatives – leading to huge control variations





$$y = f(t) + \sin(\omega t)$$

$$y' = f'(t) + \omega \cos(\omega t)$$

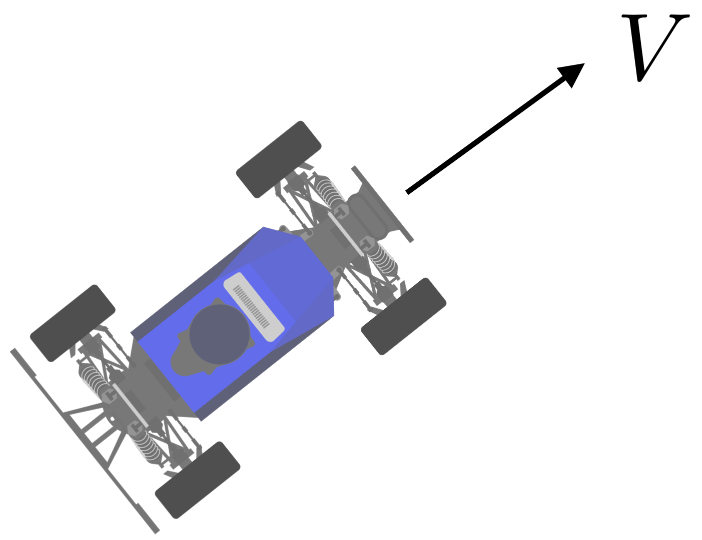
How do you evaluate the derivative term?

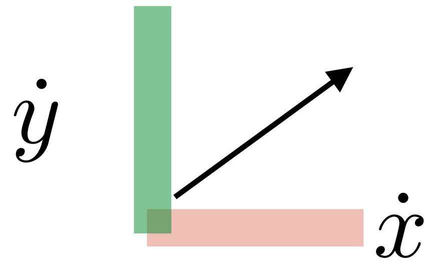
Terrible way: Calculate $\dot{e}_{ct}$ by measuring x, y and numerically differentiating to estimate $\dot{x}, \dot{y}$

Smart way: Analytically compute the derivative of the cross track error

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

$$\begin{aligned}\dot{e}_{ct} &= -\sin(\theta_{ref})\dot{x} + \cos(\theta_{ref})\dot{y} \\ &= -\sin(\theta_{ref})V\cos(\theta) + \cos(\theta_{ref})V\sin(\theta) \\ &= V\sin(\theta - \theta_{ref}) = V\sin(\theta_e)\end{aligned}$$





$$\dot{x} = V\cos(\theta)$$
$$\dot{y} = V\sin(\theta)$$

$$u = -\left(K_p e_{ct} + K_d V \sin\theta_e\right)$$

PID Intuition

$$u = -\left(K_p e_{\text{ct}} + K_i \int e_{\text{ct}} dt + K_d \dot{e}_{\text{ct}}\right)$$

PROPORTIONAL (PRESENT)

INTEGRAL (PAST)

DERIVATIVE (FUTURE)

Proportional: minimize the current error!

Integral: if I’m accumulating error, try harder!

Derivative: if I’m going to overshoot, slow down!

Tuning PID controllers

$$u = -\left(K_p e_{ct} + K_i \int e_{ct}\,dt + K_d \dot{e}_{ct}\right)$$

PROPORTIONAL (PRESENT)

INTEGRAL (PAST)

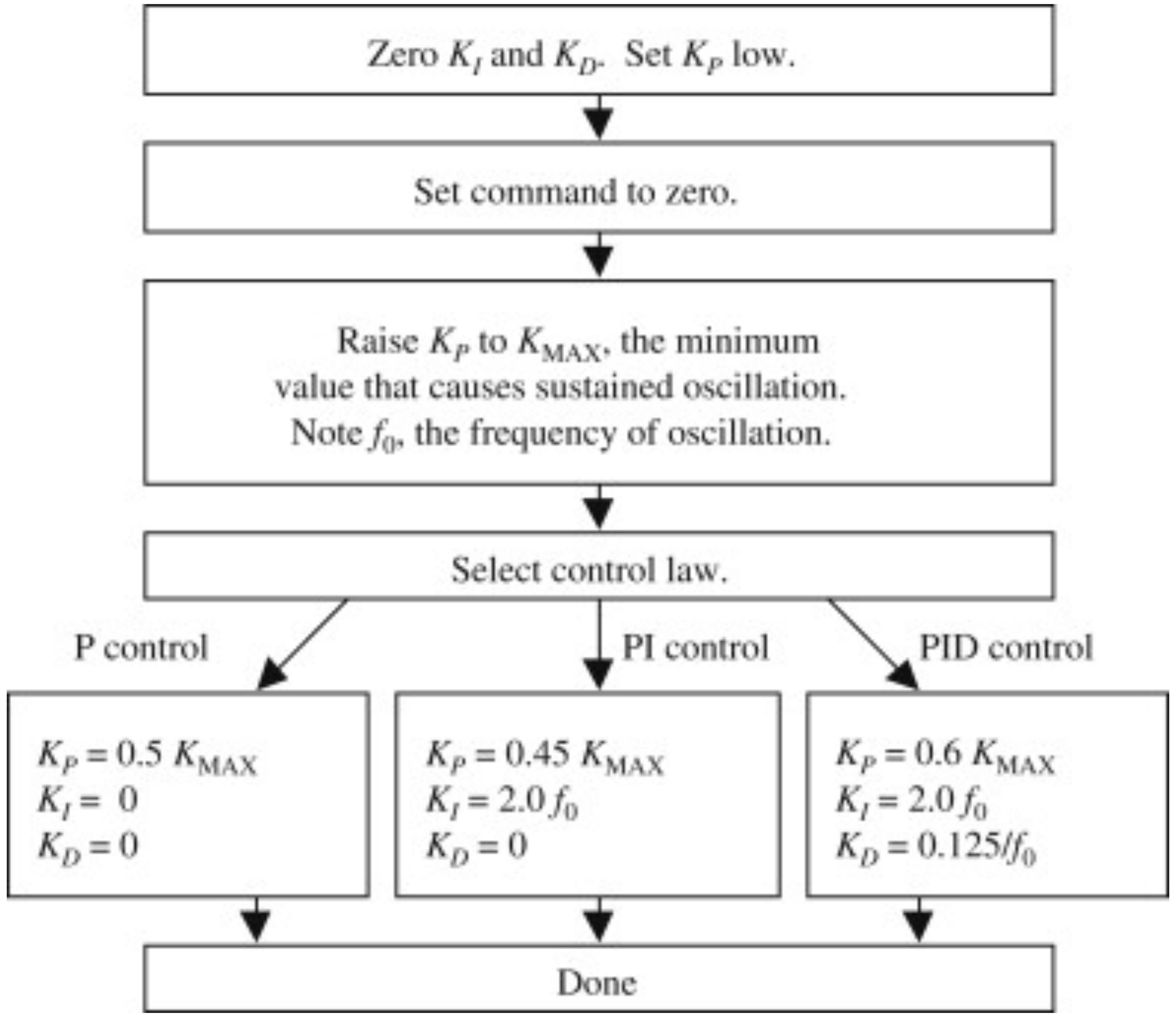
DERIVATIVE (FUTURE)

How do you set the K_p, K_i, K_d constants for a particular system?

Tuning PID controllers: Ziegler-Nichols

Heuristic/empirical method for computing K_p, K_i, K_d

$$u = -\left(K_p e_{\text{ct}} + K_i \int e_{\text{ct}} dt + K_d \dot{e}_{\text{ct}}\right)$$



See how the system responds to proportional gain

Adjust integral and proportional accordingly