

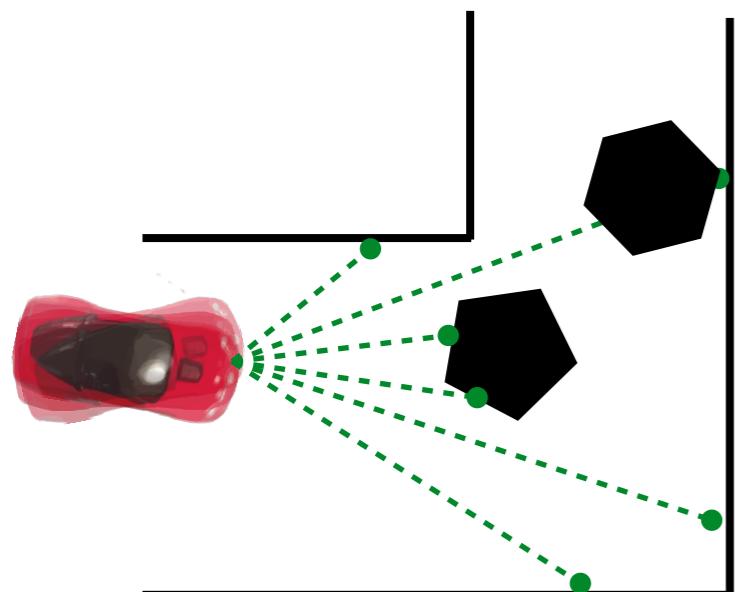
# Feedback Control

Instructor: Chris Mavrogiannis

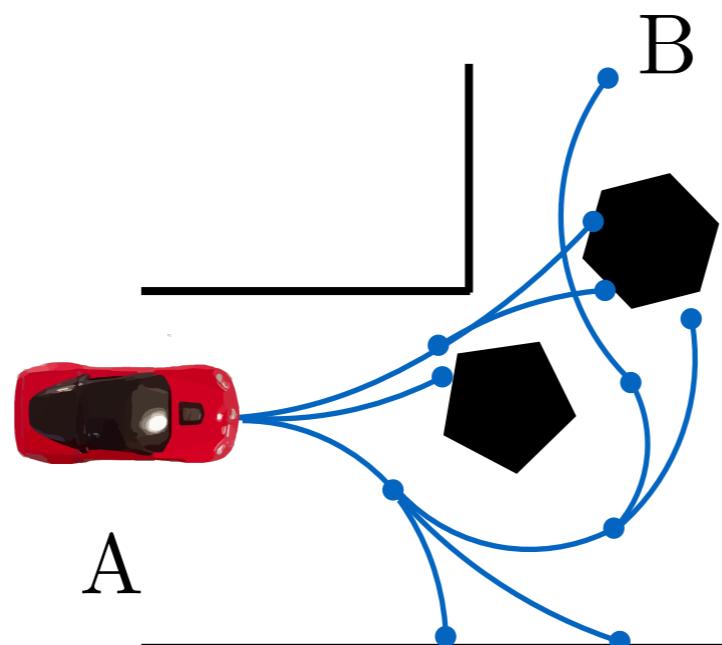
TAs: Kay Ke, Gilwoo Lee, Matt Schmittle

\*Slides based on or adapted from Sanjiban Choudhury, Russ Tedrake

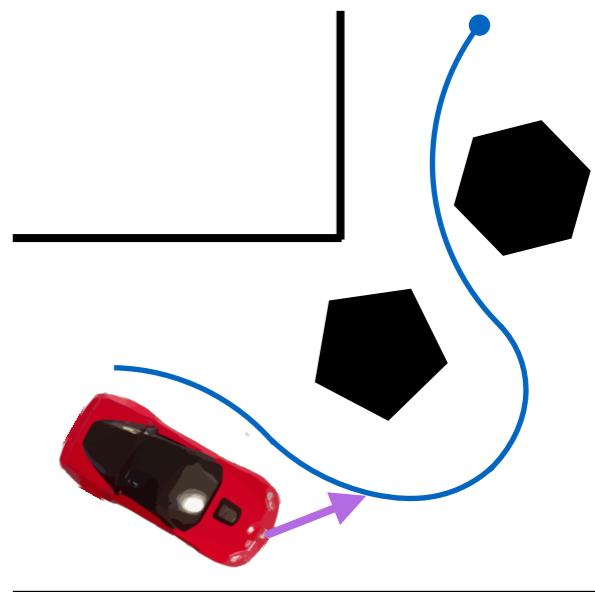
# Estimate state



# Plan a sequence of motions

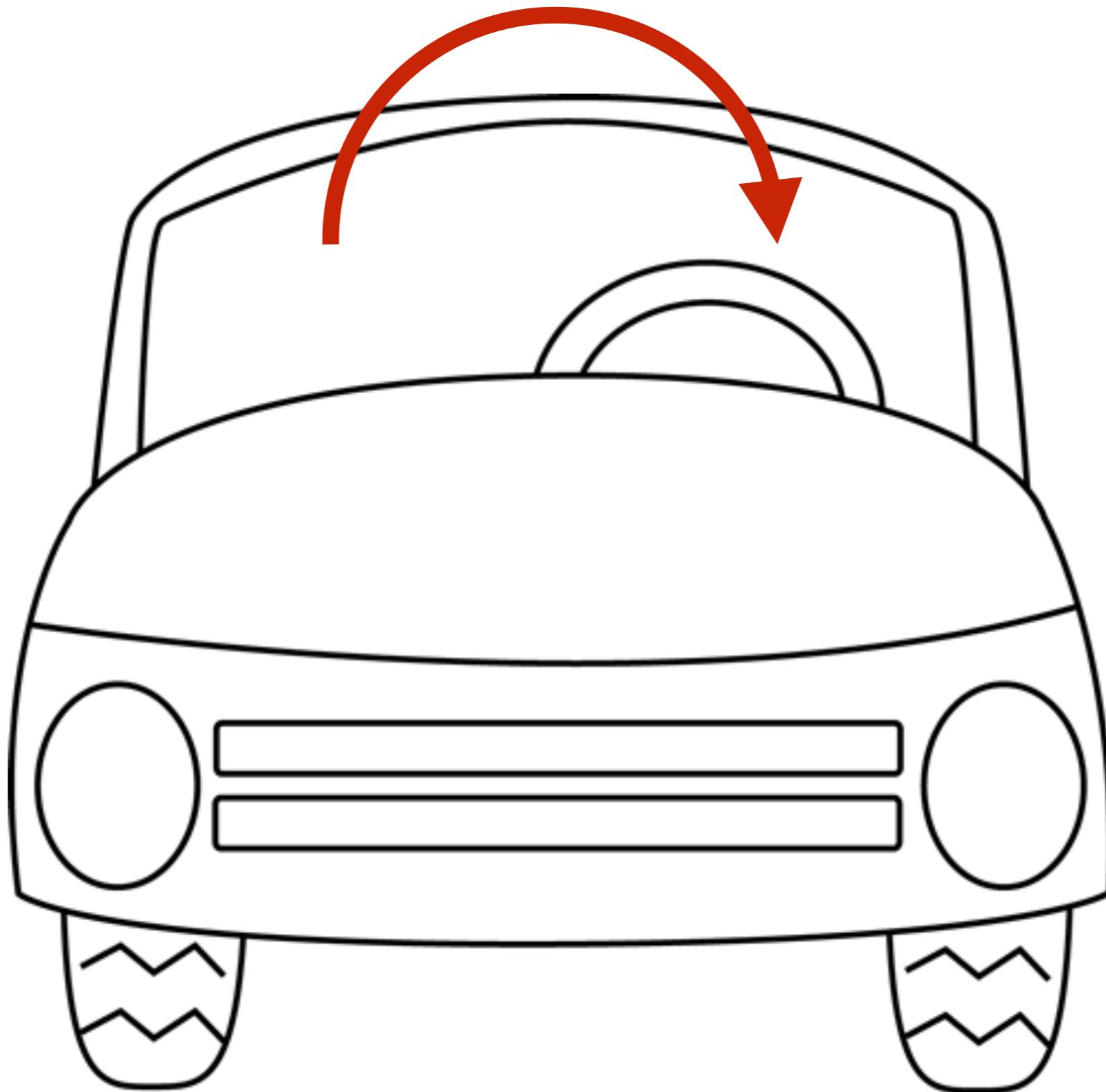


# Control robot to follow path



- Robot pose known
- Path is given

# From perception to control ...



# When I think about control ...

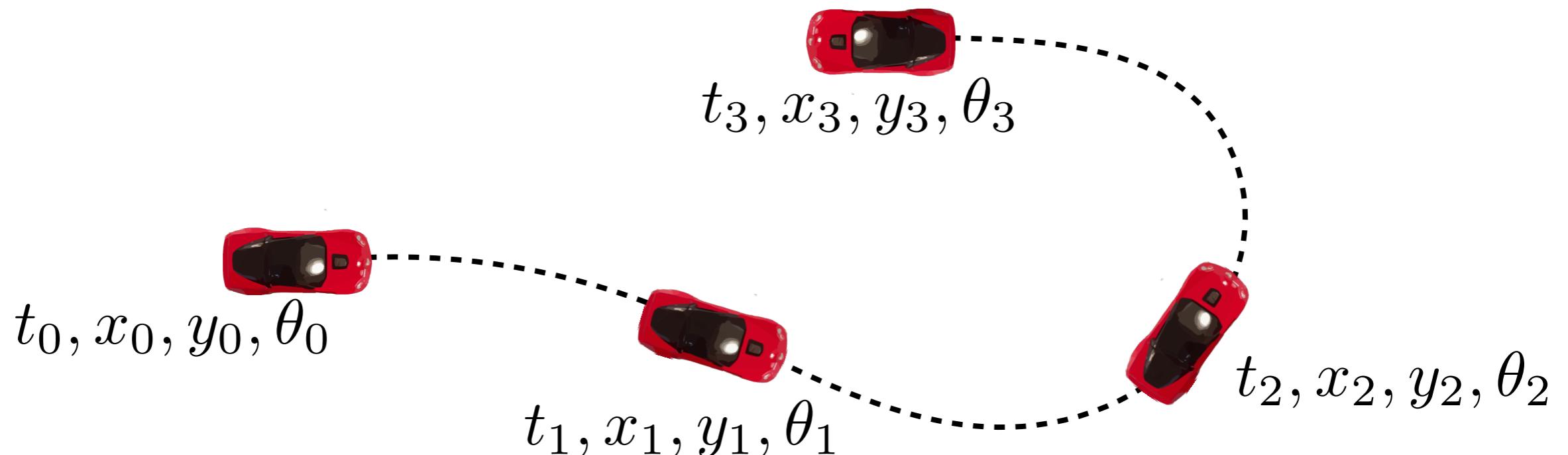


# Today's objective

1. Introduce terms and definitions in feedback control
2. Go through challenges in current control research

# The control framework

Say we want to get the car to hit a sequence of poses



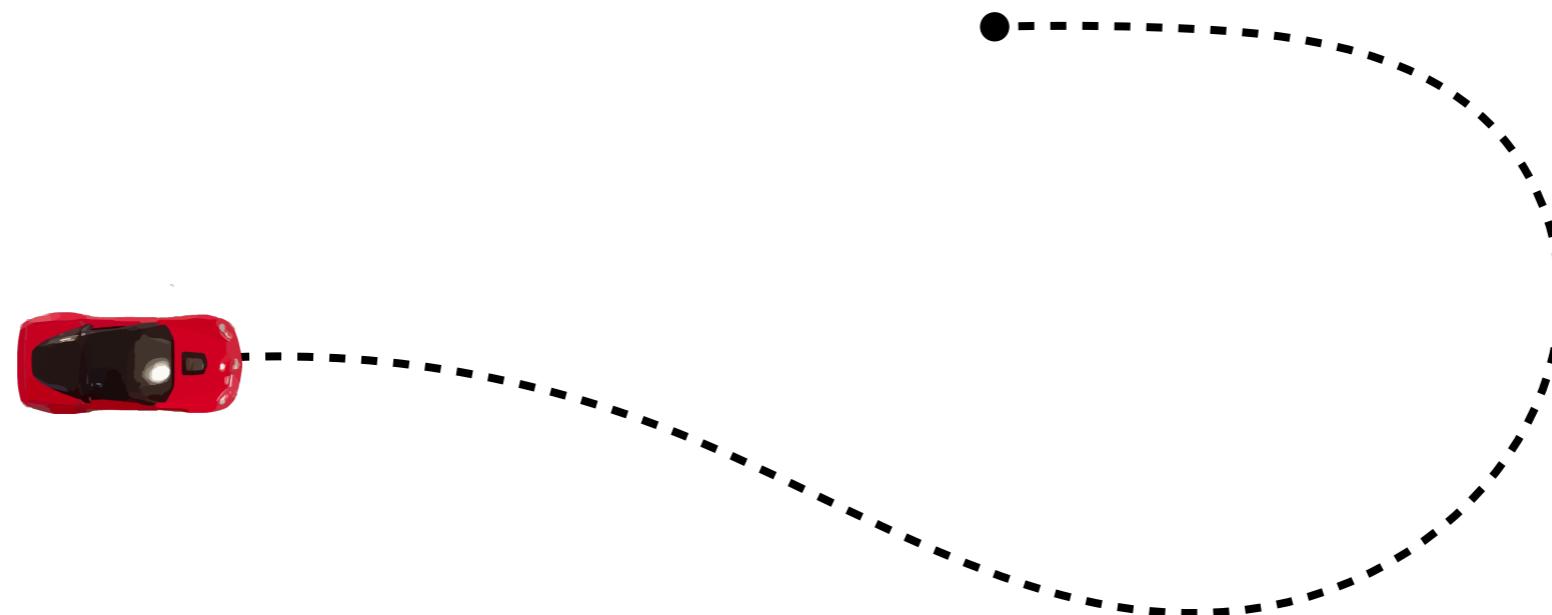
Can express this as wanting to track a **reference trajectory**

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

# The control framework

Let's say we want to track a reference trajectory

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



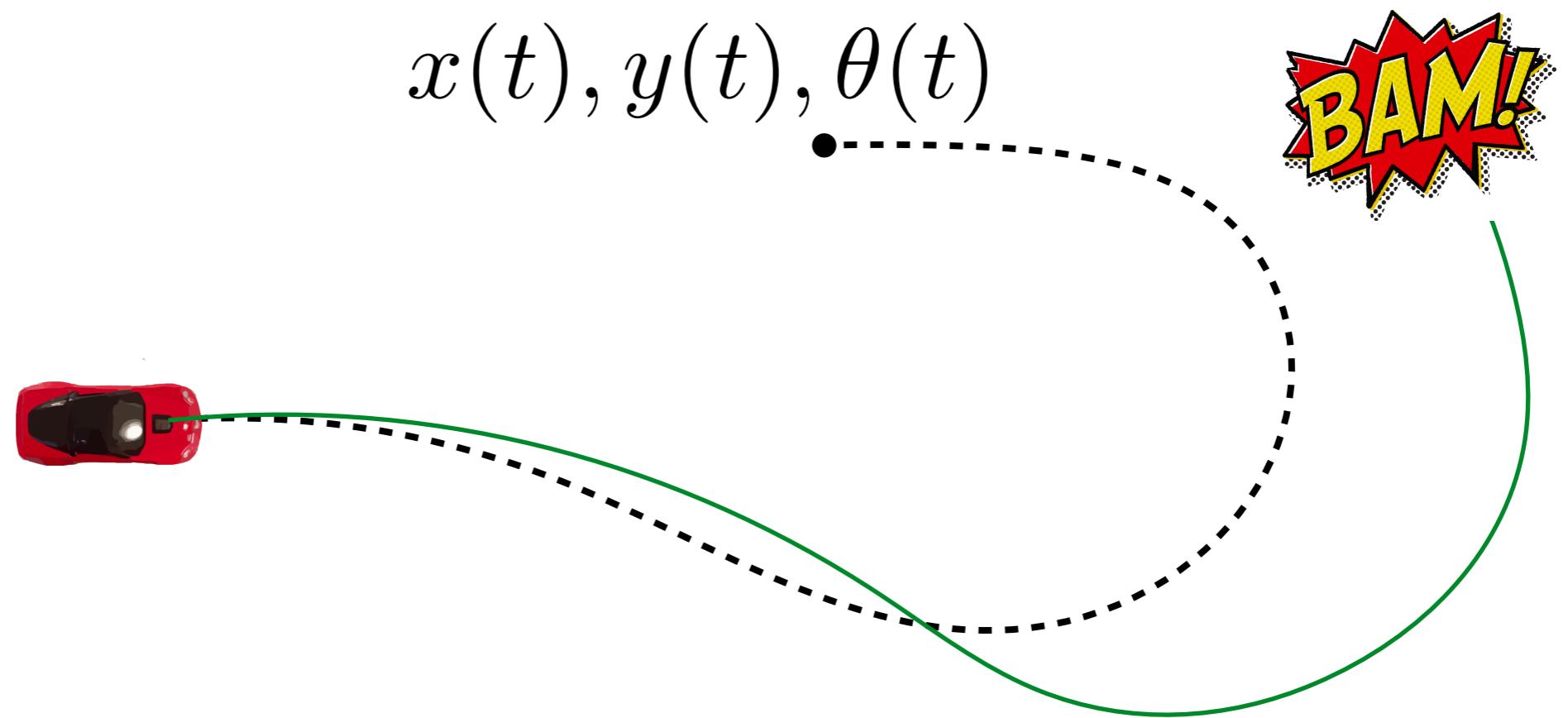
**Objective:** Figure out a **control** trajectory  $u(t)$  to achieve this

In our case, we will focus on steering angle  $\delta(t)$  as control input

Why do we need feedback?

# Why do we need feedback?

Let's say we want to track a reference trajectory



What if we send out steering angles  $\delta(t)$  obtained from kinematic car model?

Open loop control leads to accumulating errors!

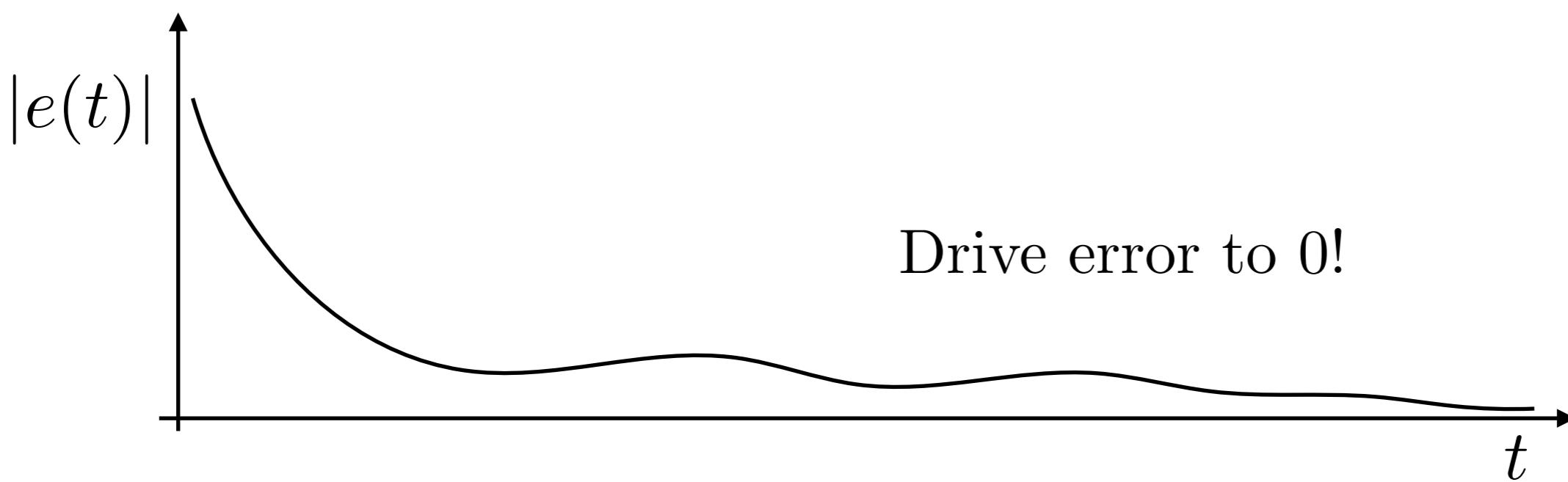
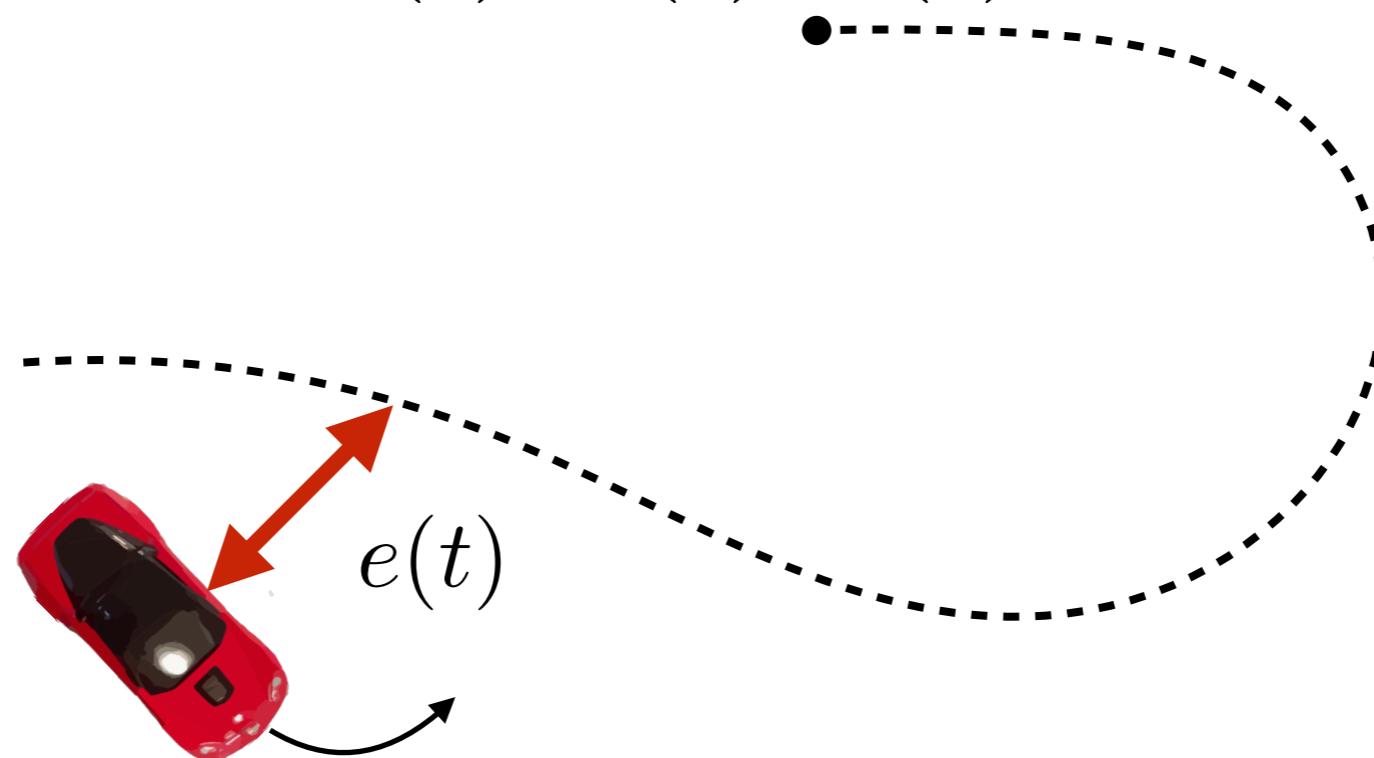
Overarching principle of  
feedback control:

Measure error between  
reference and state.  
Minimize this error.

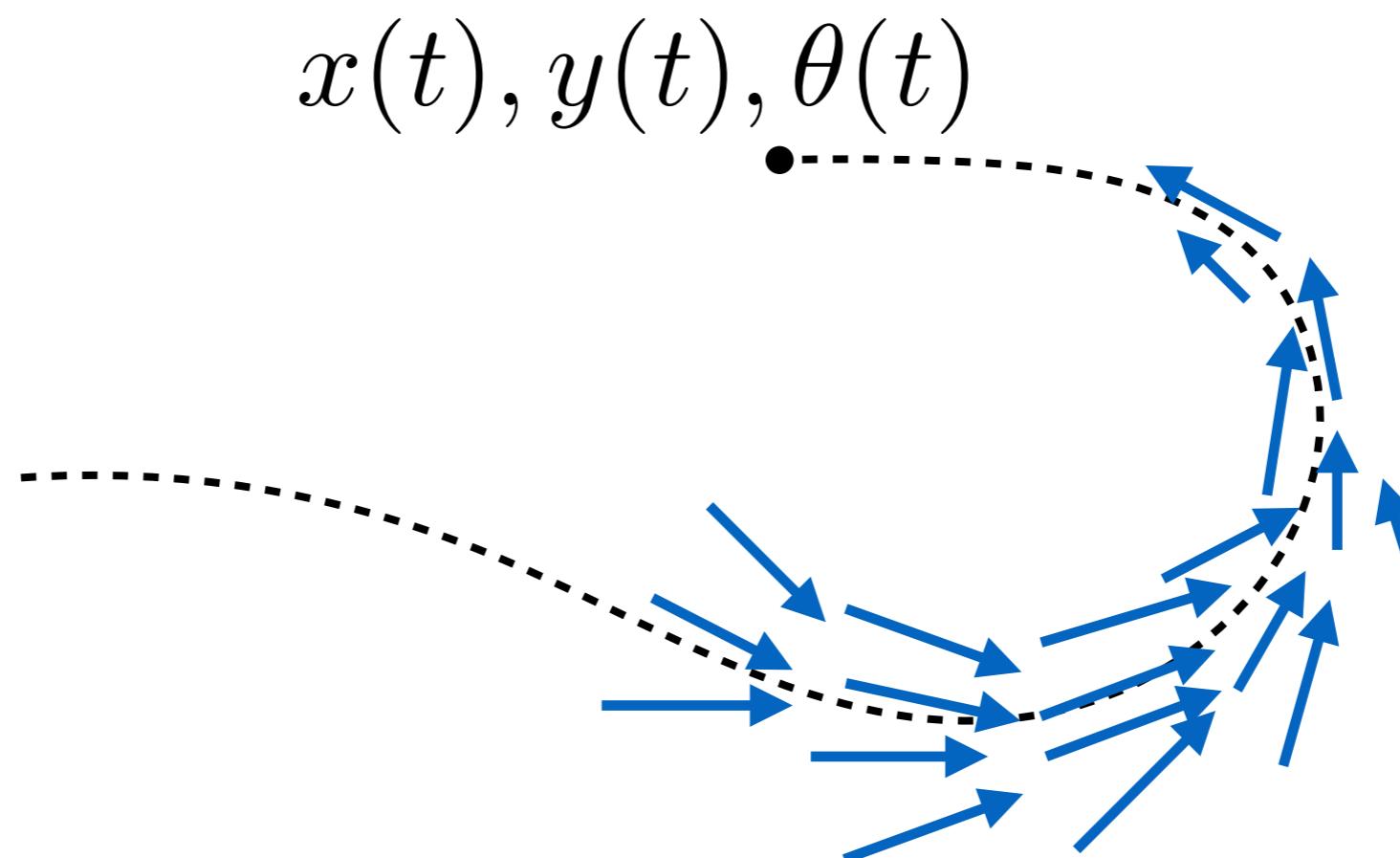
# Measure error and minimize it

Error  
between  
state and  
reference

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

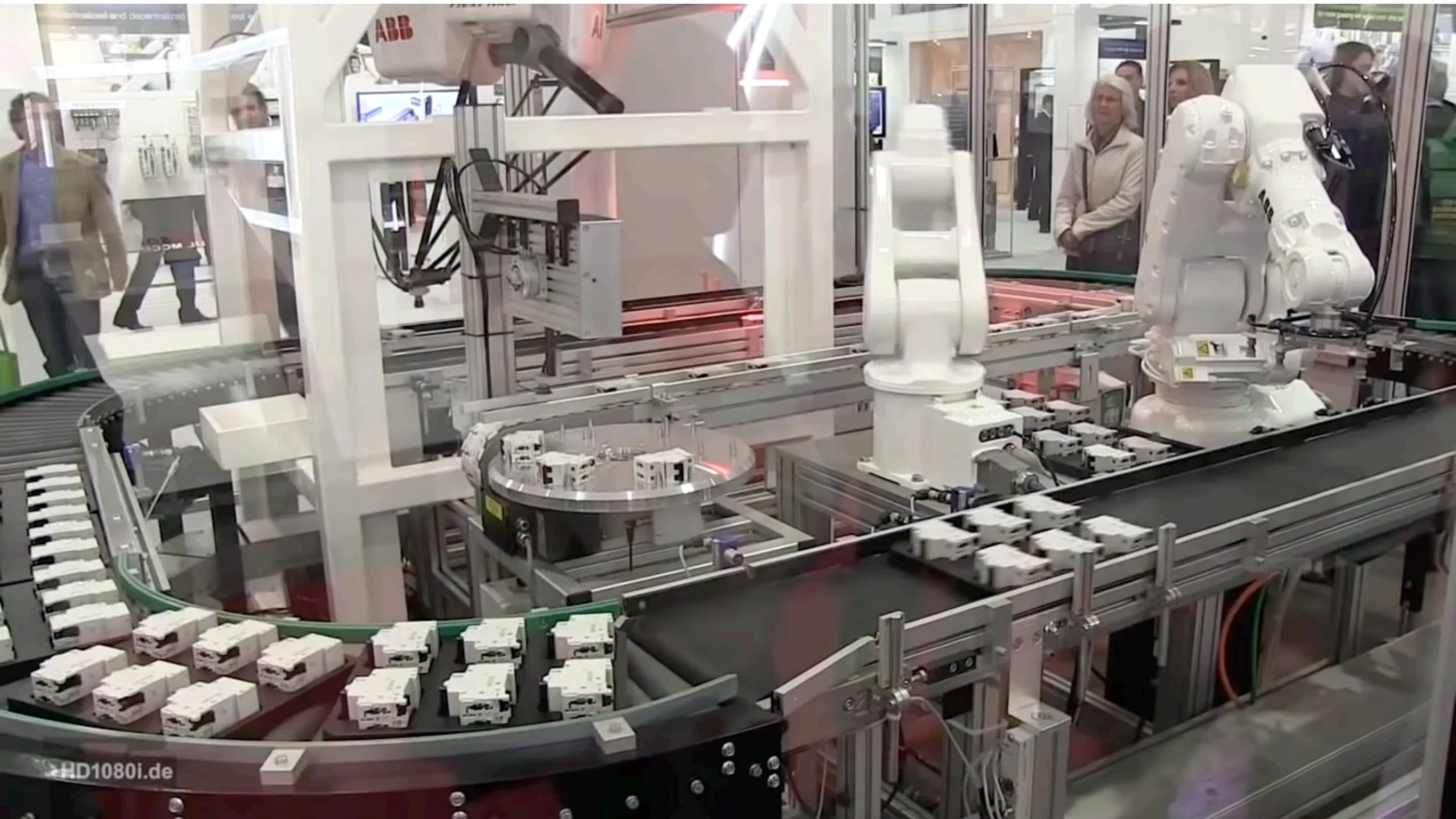


Useful to think of control laws as vector fields



Is this still a research problem?

# Industrial robots hard at work



[https://www.youtube.com/watch?v=J\\_8OnDsQVZE&t=315s](https://www.youtube.com/watch?v=J_8OnDsQVZE&t=315s)

# Assumptions made by such controllers

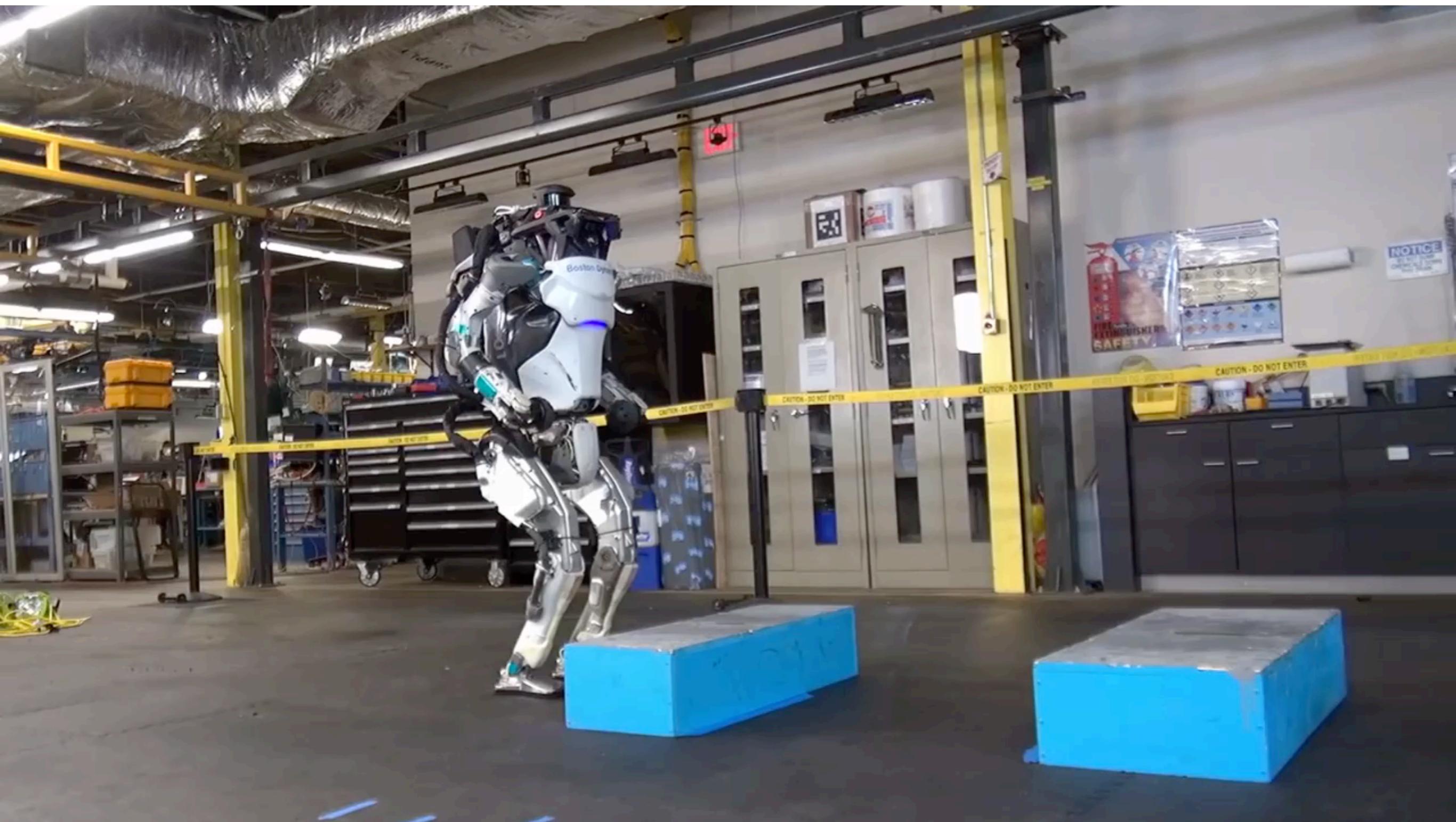
1. Fully actuated: There exists an inverse mapping from reference to control actions

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

2. Almost no execution error or state estimation error
3. Enough control authority to clamp down errors / overcome disturbances

Instead, what are harder  
problems we want to think about?

# The Atlas robot hard at ... play?



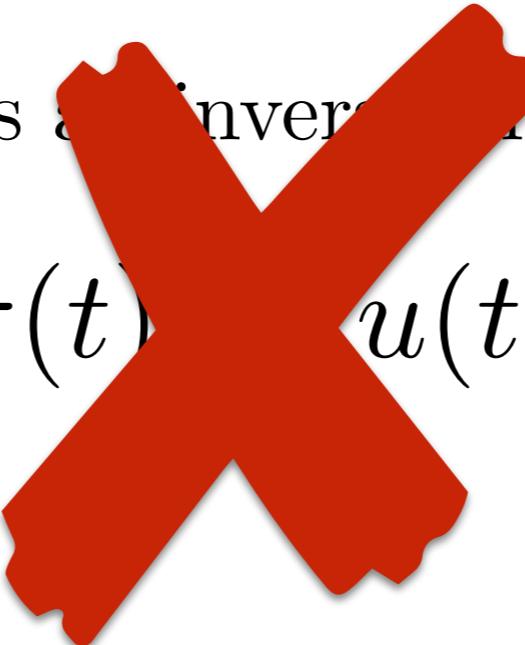
<https://www.youtube.com/watch?v=fRj34o4hN4I>

Why is this a hard problem?

# Challenge 1: Underactuated systems

Fully actuated: There exists a universal mapping from reference to control actions

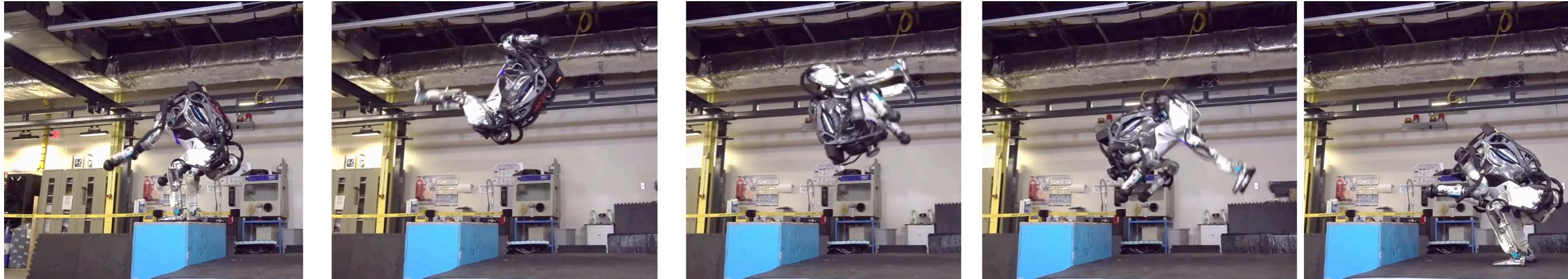
$$\sigma(t) \xrightarrow{\quad} u(t)$$



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?



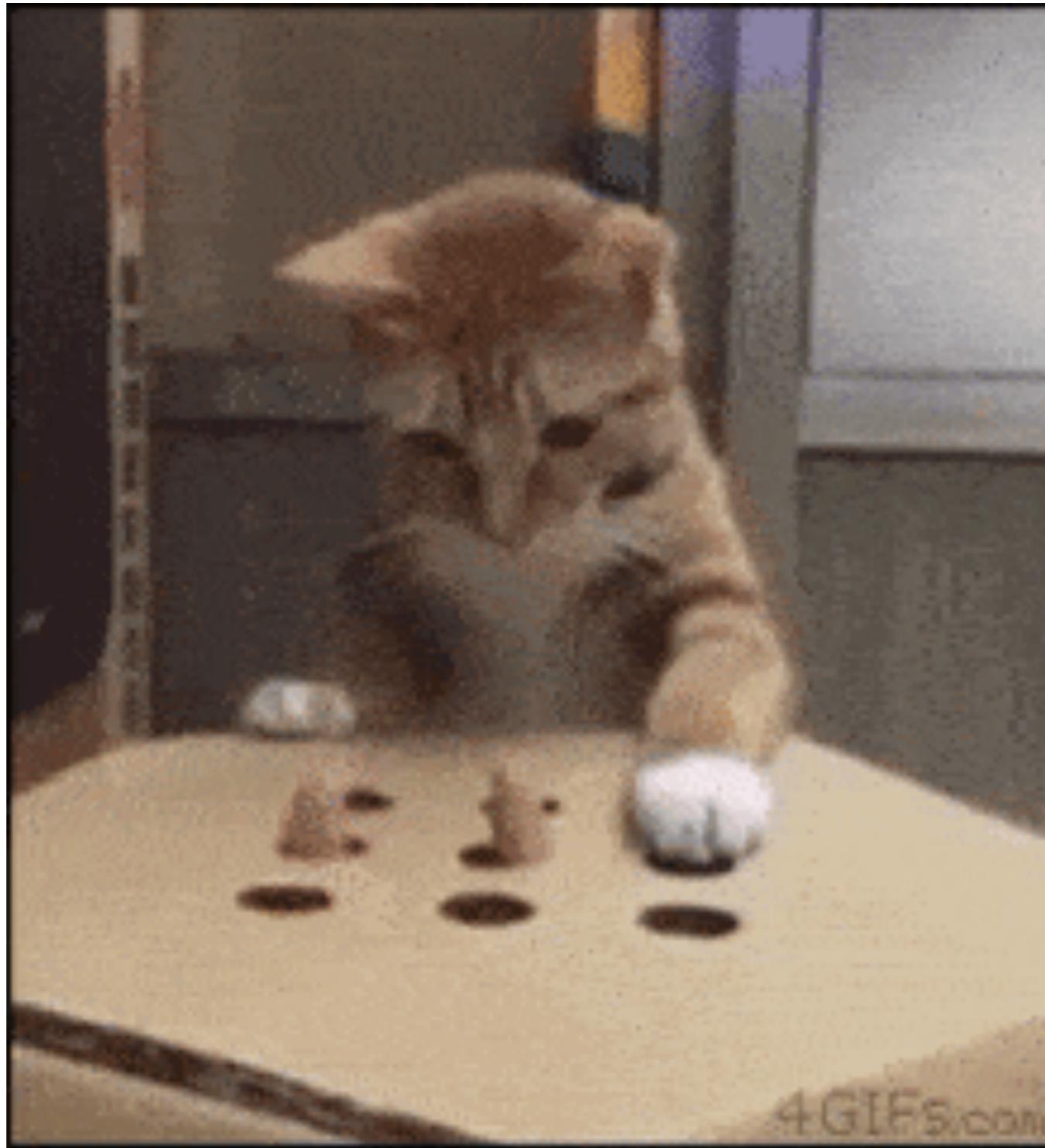
Some  
initial  
motor  
thrust ...

Whole lot of gravity!

Whole lot of momentum!

... some precise  
control adjustments

# Myth: Control is a battle with nature!



Is control like playing whack-a-mole with error terms??

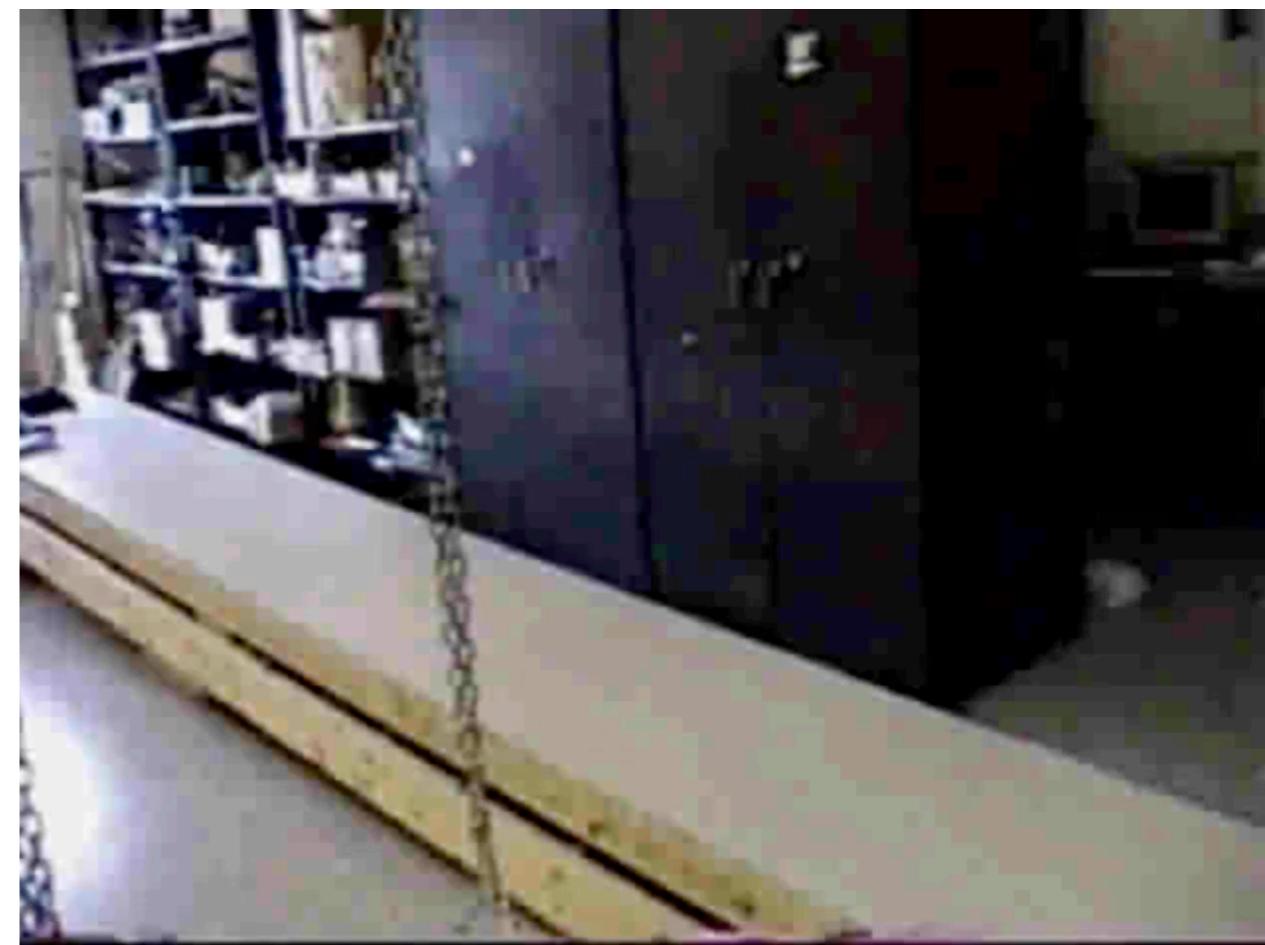
# Embrace dynamics instead of fighting it

State-of-the-art humanoid  
(at the time)



Honda, 1996

Passive walker  
(no motors, powered by gravity!)



Steven H. Collins, Martijn Wisse, and Andy Ruina, 2001

Which one looks more natural? Which one consumes less energy?

## Question:

If we know the model of our robot,  
can't we solve a ginormous  
optimization to figure out control ...?

# Doing backflips with a helicopter

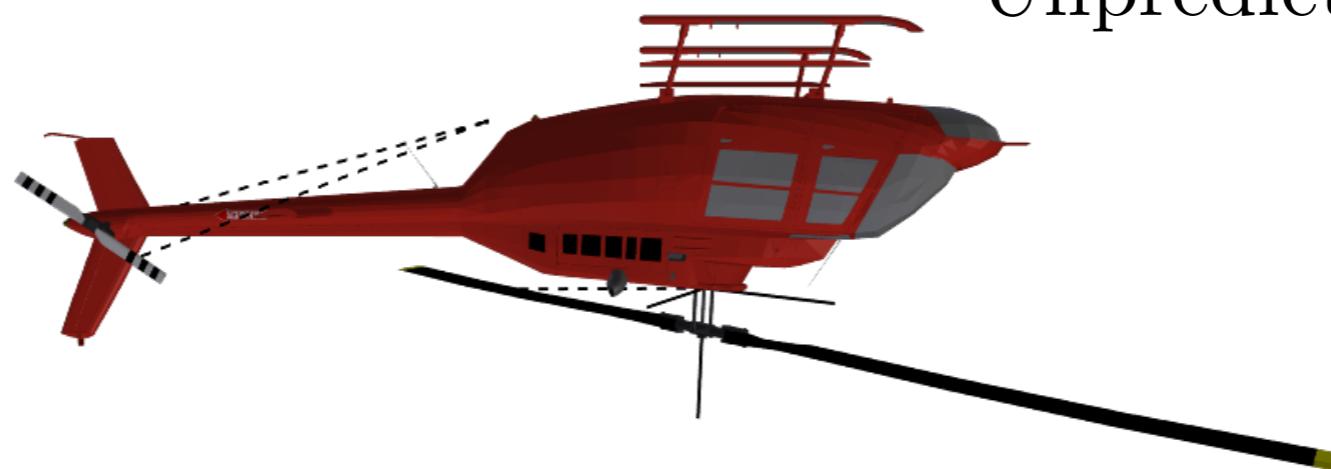


Redbull Eurocopter BO-105

[https://www.youtube.com/watch?v=RGu45s1\\_QPU](https://www.youtube.com/watch?v=RGu45s1_QPU)

# And just what is this model ?!?

Nothing  
countering  
gravity!

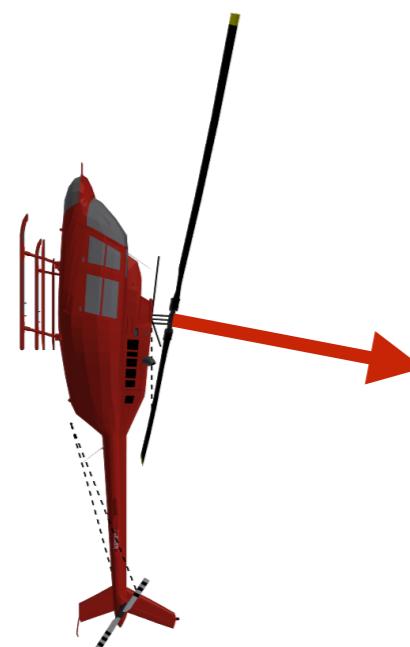


Unpredictable drag forces!

Chaotic vortex around blades!

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

# Challenge 2: Choose good closed-loop models

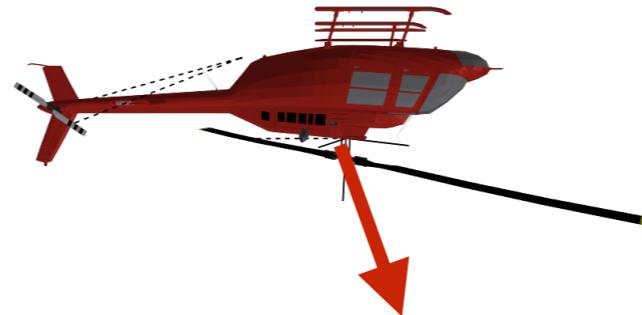


Complex  
dynamics

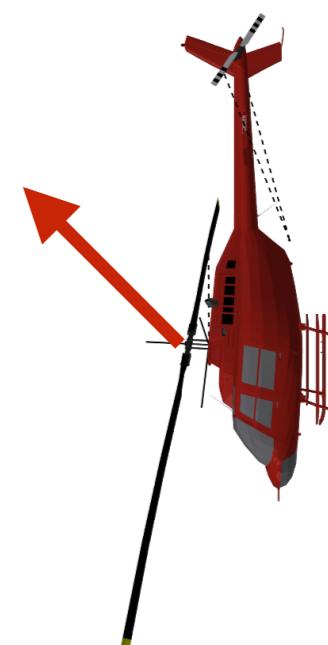
+

Feedback  
control law

=



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



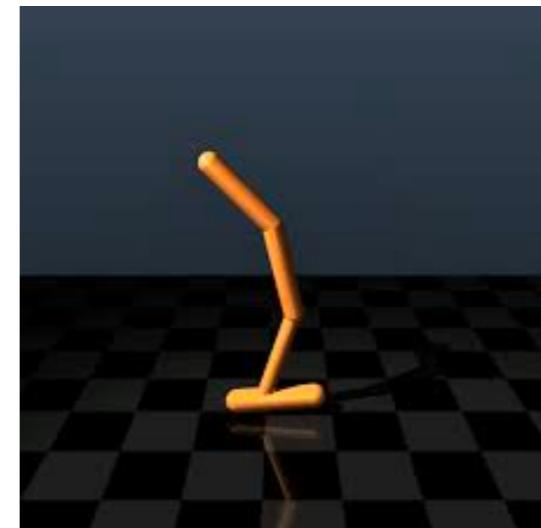
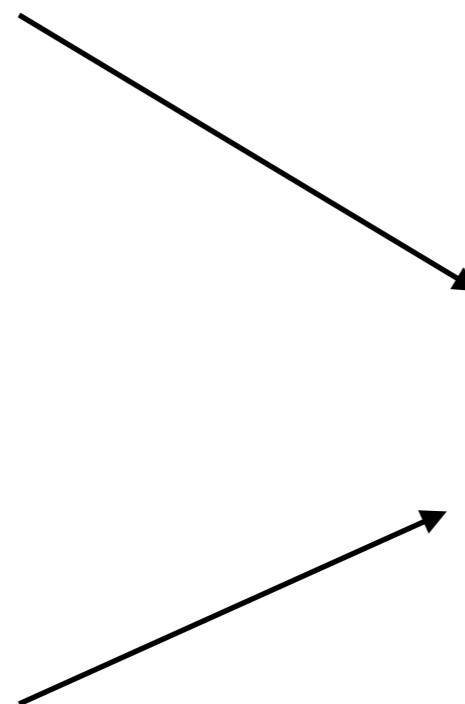
Well-behaved  
system

# Trotting quadruped = biped = one-leg

Quadruped



Biped



One-leg

Marc Raibert showed how these closed loop systems are equivalent

## Question:

Is this all offline? Can I pre-compute  
a bunch of controllers once and  
call it a day?

# Figure 8s while drifting



Stanford's MARTY

<https://www.youtube.com/watch?v=nTK56vPb8Zo>

# Challenge 3: Model changing on the fly!

Run **real-time estimators** for wheel characteristics

Need control laws for all possible model parameters

# Other challenges for mobile robot control

1. Unexpected obstacle avoidance
2. Noisy state estimation
3. Controllers that guarantee safety

Enough with the motivation ...

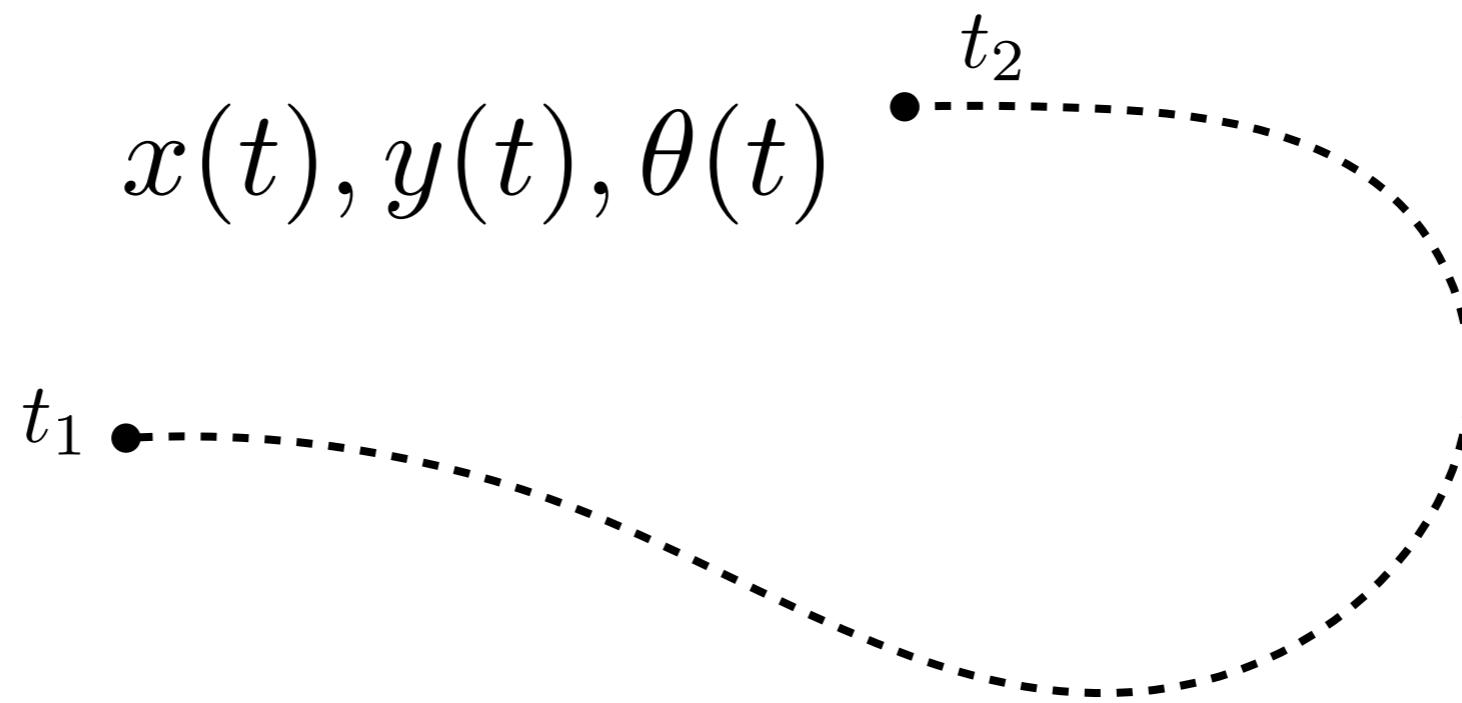
Let's start writing some controllers

# A generic template for a controller

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

# Step 1: Get a reference

Reference can be a **time-parameterized trajectory**



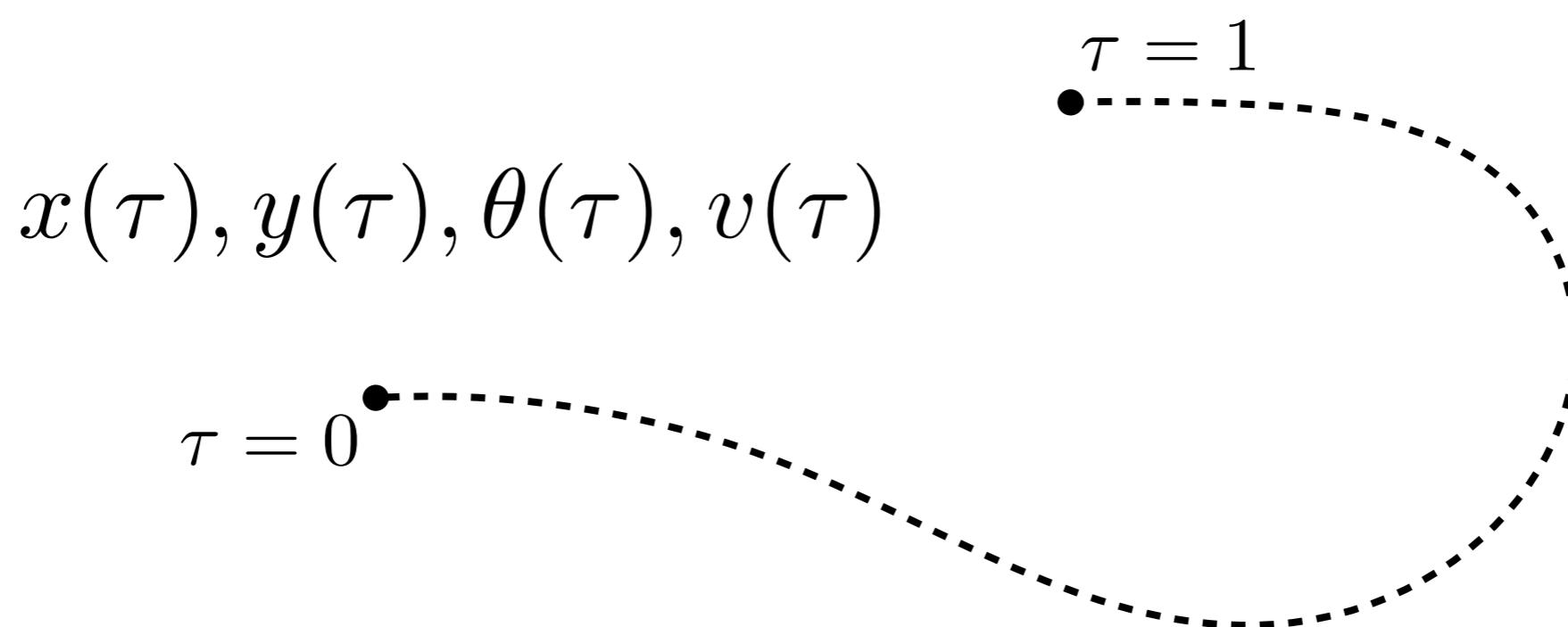
The time index refers to a desired pose we want the system to achieve at a given **time**

**Pro:** Useful if we want the robot to respect time constraints

**Con:** Sometimes we care only about deviation from reference

# Step 1: Get a reference

Reference can be a index-parameterized path



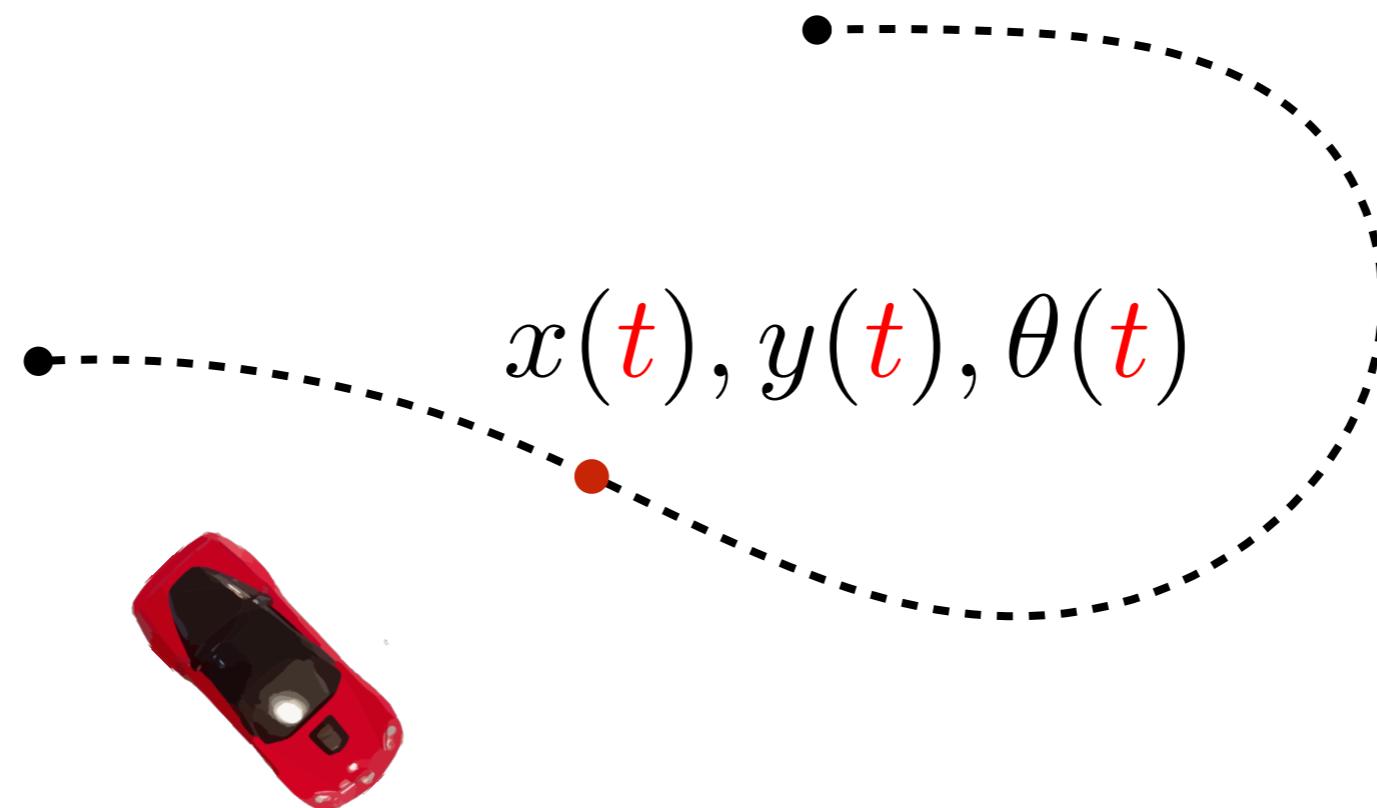
The index is simply a way to access the path; there is no notion of time

**Pro:** Useful for conveying the shape you want the robot to follow

**Con:** Can't control when robot will reach a point

# Step 2: Pick a point on the reference

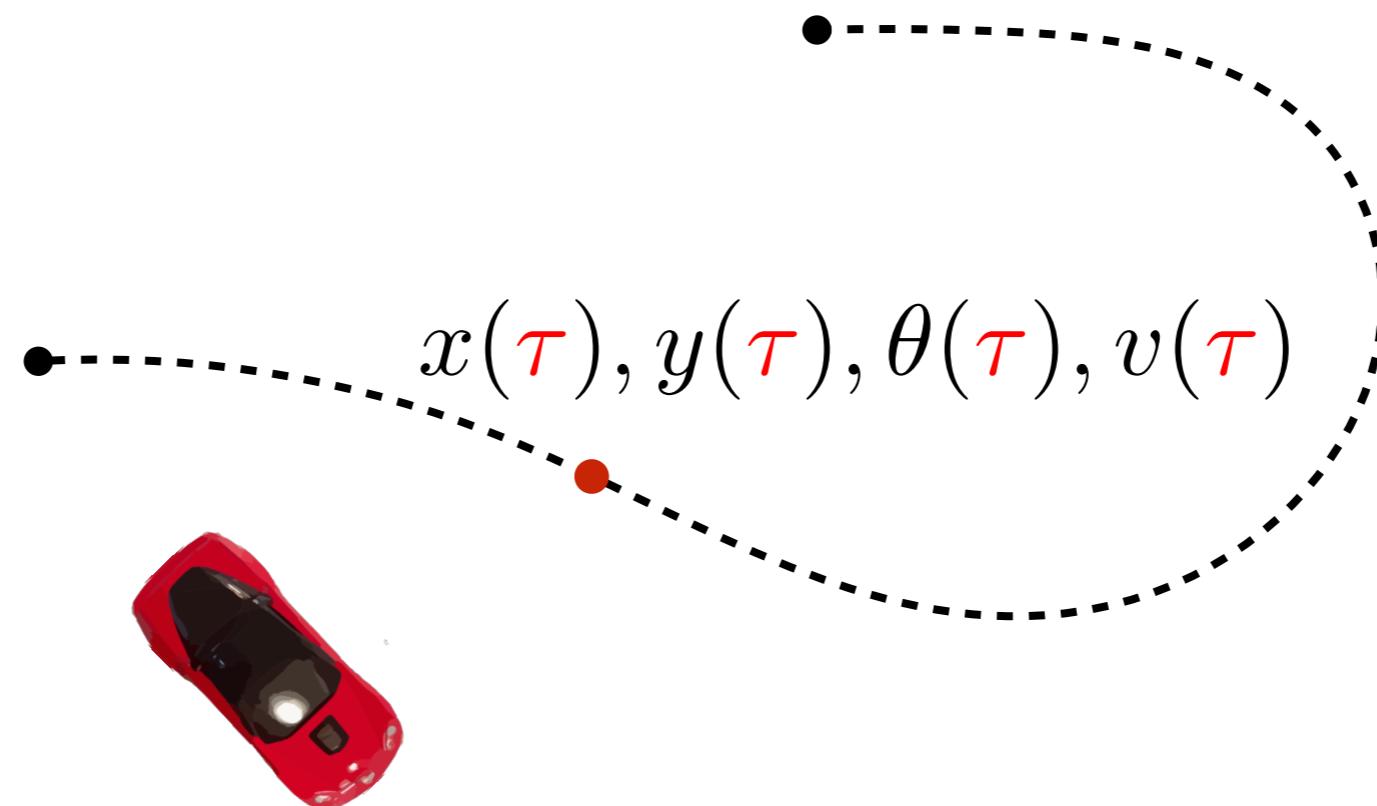
Pick a **reference point** on the trajectory / path



If we are using a time-parameterized trajectory,  
the current time is the natural reference

# Step 2: Pick a point on the reference

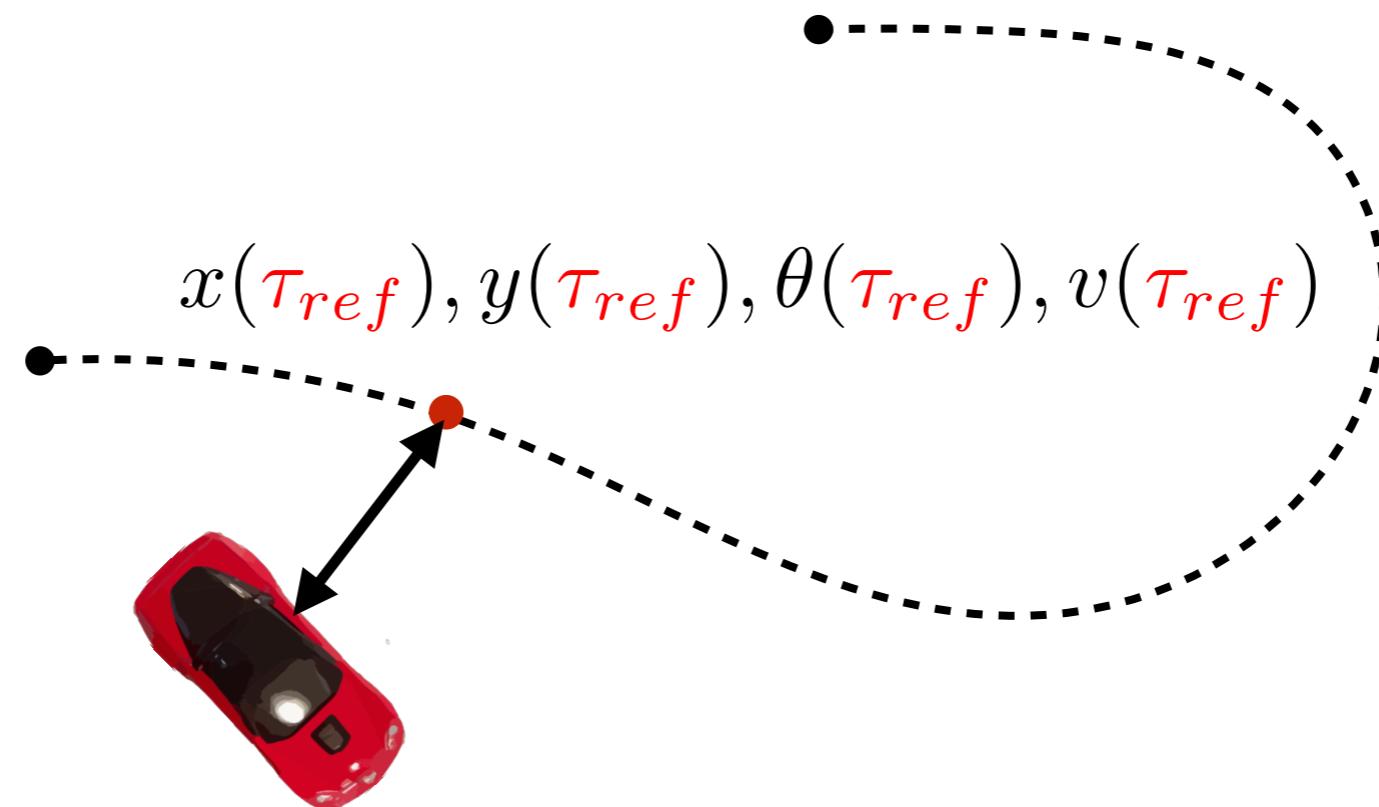
Pick a **reference point** on the trajectory / path



If we are using a index-parameterized trajectory,  
there are multiple options

# Step 2: Pick a point on the reference

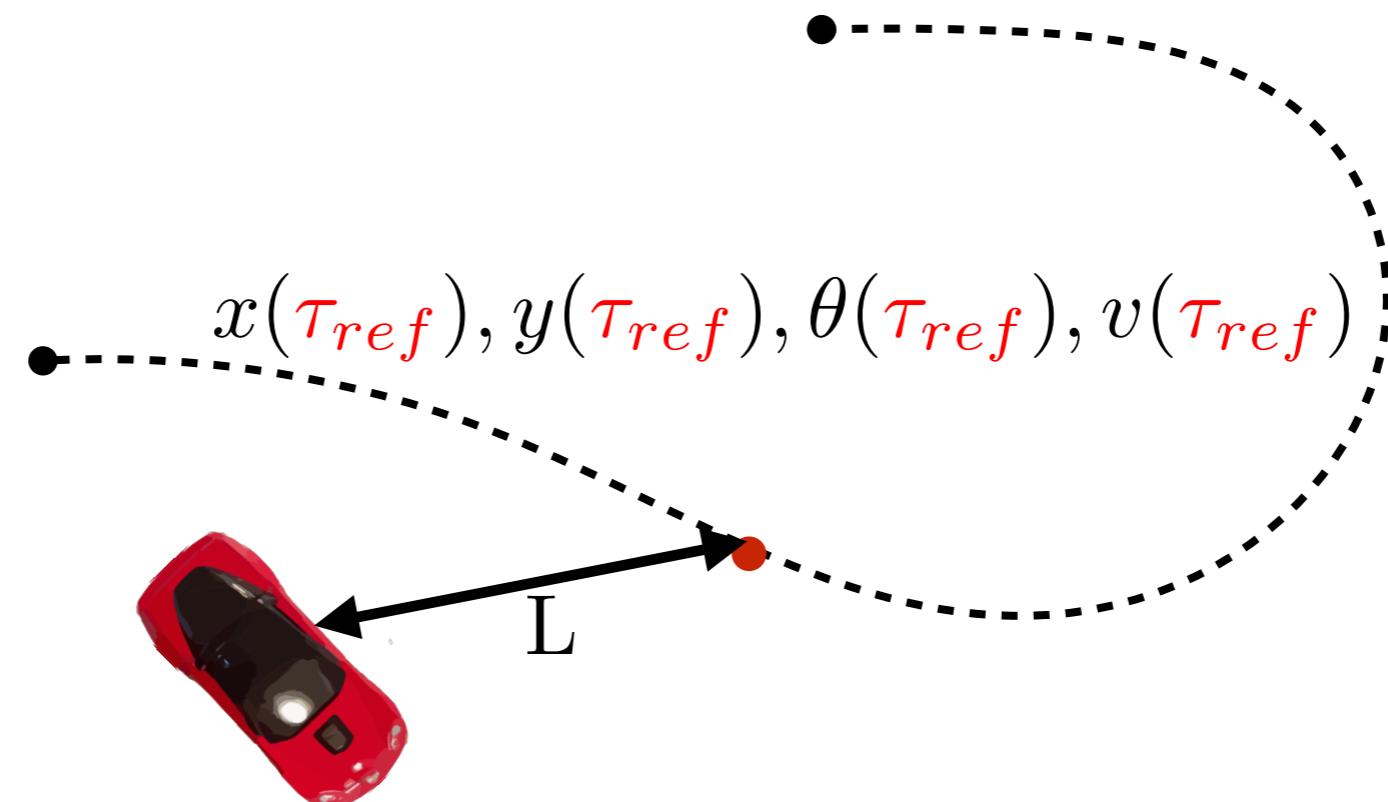
Option 1: Pick the closest point on the path



$$\tau_{ref} = \arg \min_{\tau} \| [x \ y]^T - [x(\tau) \ y(\tau)]^T \|$$

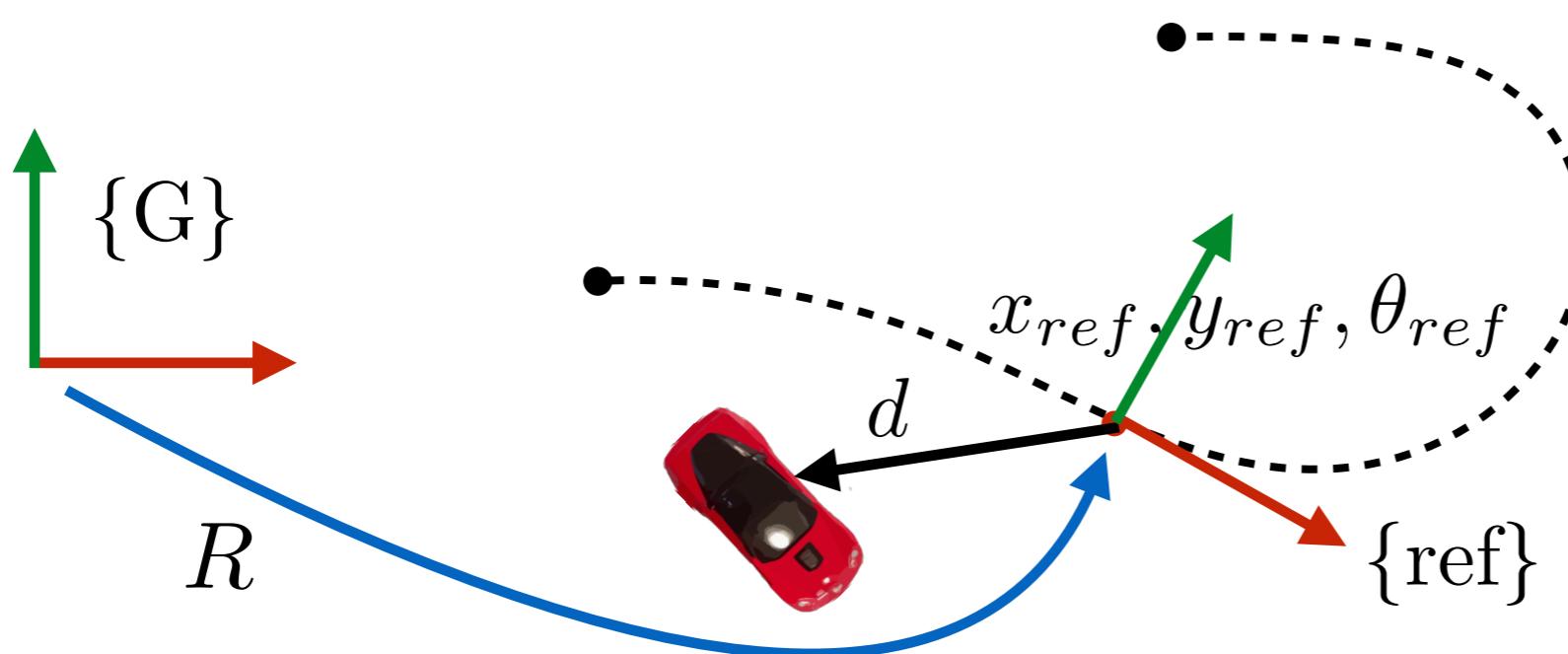
# Step 2: Pick a point on the reference

Option 2: Pick a lookahead point on the path



$$\| [x \ y]^T - [x(\tau_{ref}) \ y(\tau_{ref})]^T \| = L$$

# Step 3: Compute error to reference point



1. Define error vector

$$d = \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_{ref} \\ y_{ref} \end{bmatrix}$$

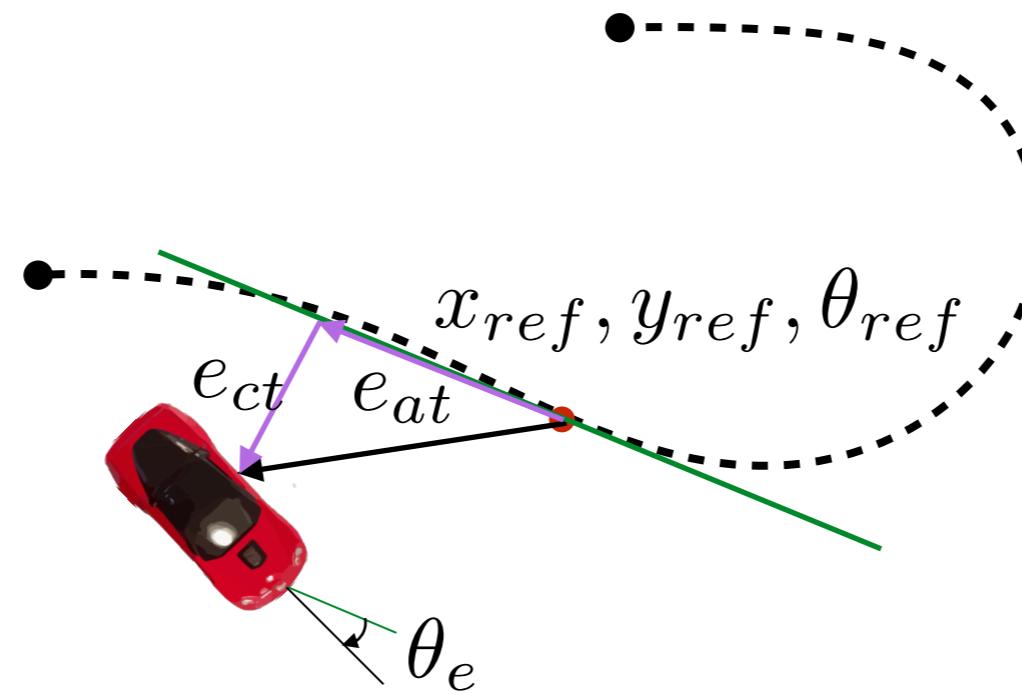
2. Rotate error vector to be in the **reference** frame

Rotation  
Matrix

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

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

# Step 3: Compute error to reference point



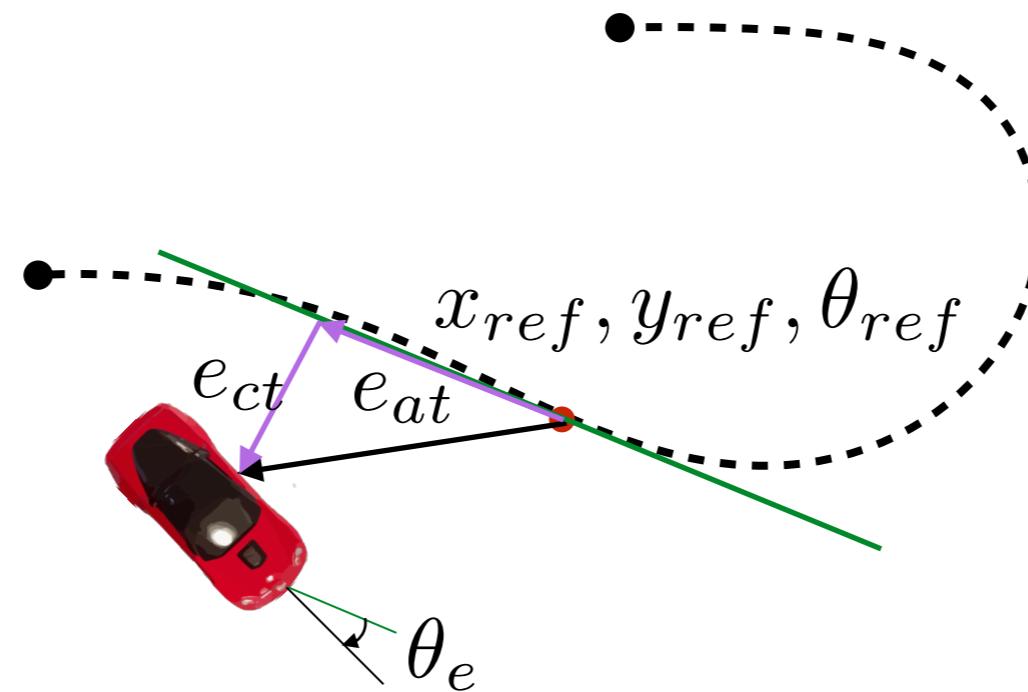
Error has two components: along-track and cross-track

$$e = \begin{bmatrix} e_{at} \\ e_{ct} \end{bmatrix} \quad \begin{array}{l} \text{(along track)} \\ \text{(cross track)} \end{array}$$

$$\theta_e = \theta - \theta_{ref} \quad \text{(heading error)}$$

Let's consider only the cross track error for now

# Step 3: Compute error to reference point



Cross-track error

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

Derivative of cross-track error

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

# 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

We assume that control speed is set to be equal to reference speed

Hence control = steering angle

# Different control laws

1. Bang-bang control
2. PID control
3. Pure-pursuit control
4. Lyapunov control
5. Linear Quadratic Regulator (LQR)
6. Model Predictive Control (MPC)