Content from Drew Bagnell, Pieter Abeel

Iterative LQR & Model Predictive Control

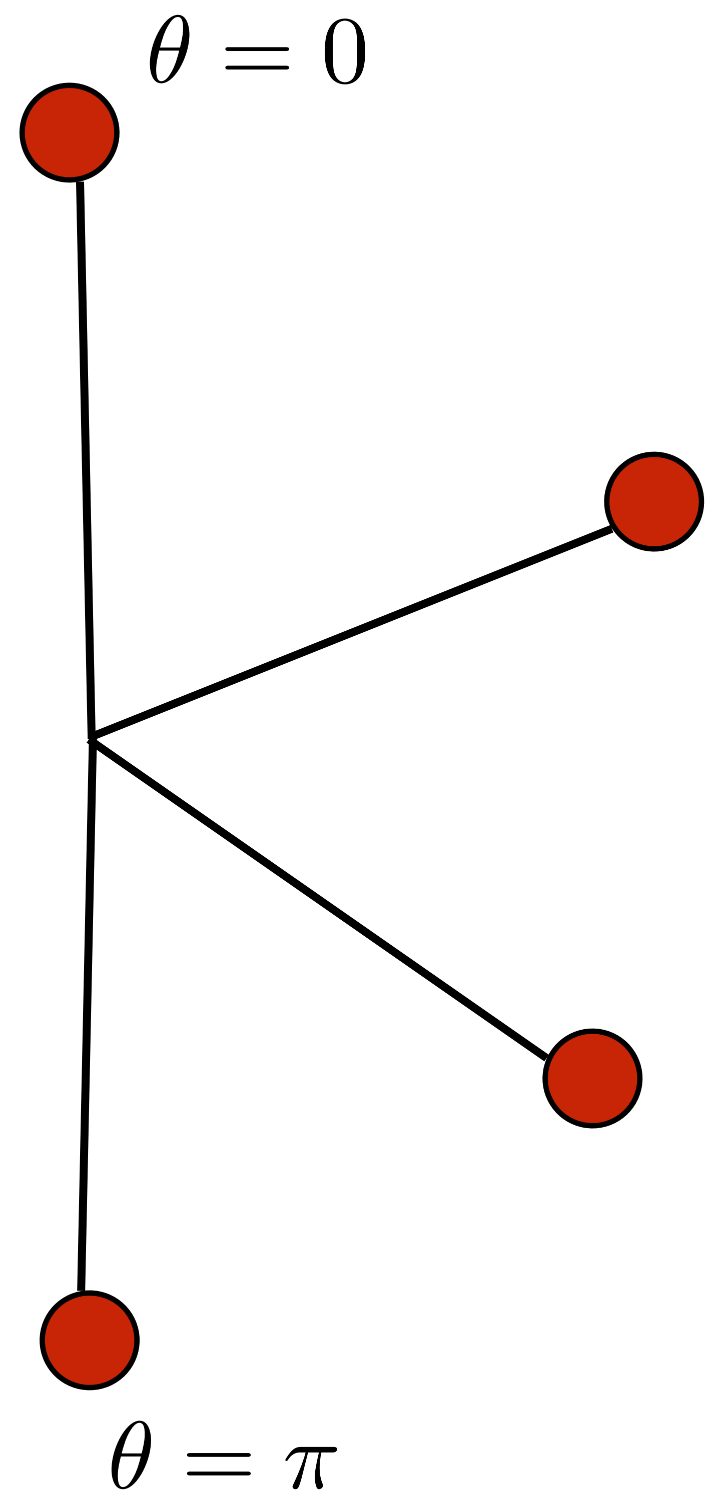
Sanjiban Choudhury

TAs: Matthew Rockett, Gilwoo Lee, Matt Schmittle

Table of Controllers

	Control Law	Uses model	Stability Guarantee	Minimize Cost
PID		No	No	No
Pure Pursuit		Circular arcs	Yes - with assumptions	No
Lyapunov		Non-linear	Yes	No
LQR		Linear	Yes	Quadratic

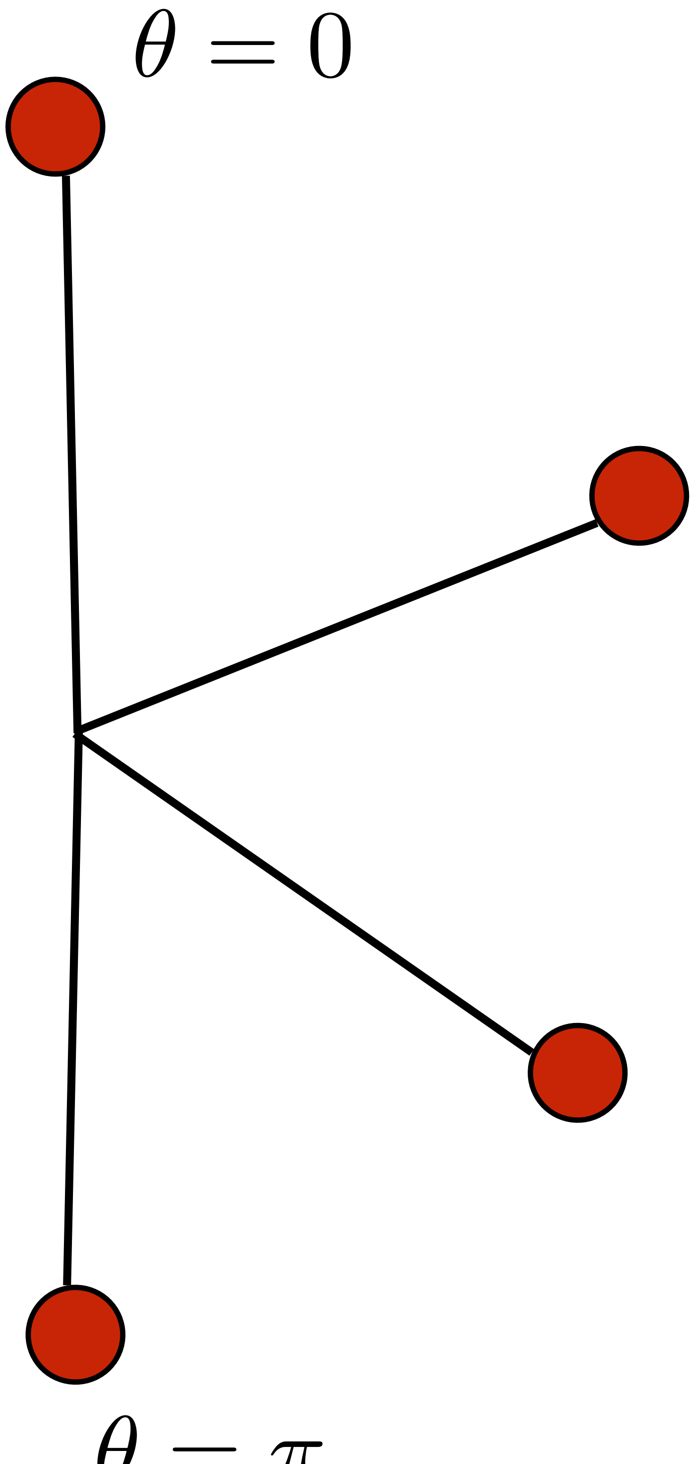
Can we use LQR to swing up a pendulum?



Can we use LQR to swing up a pendulum?

No!

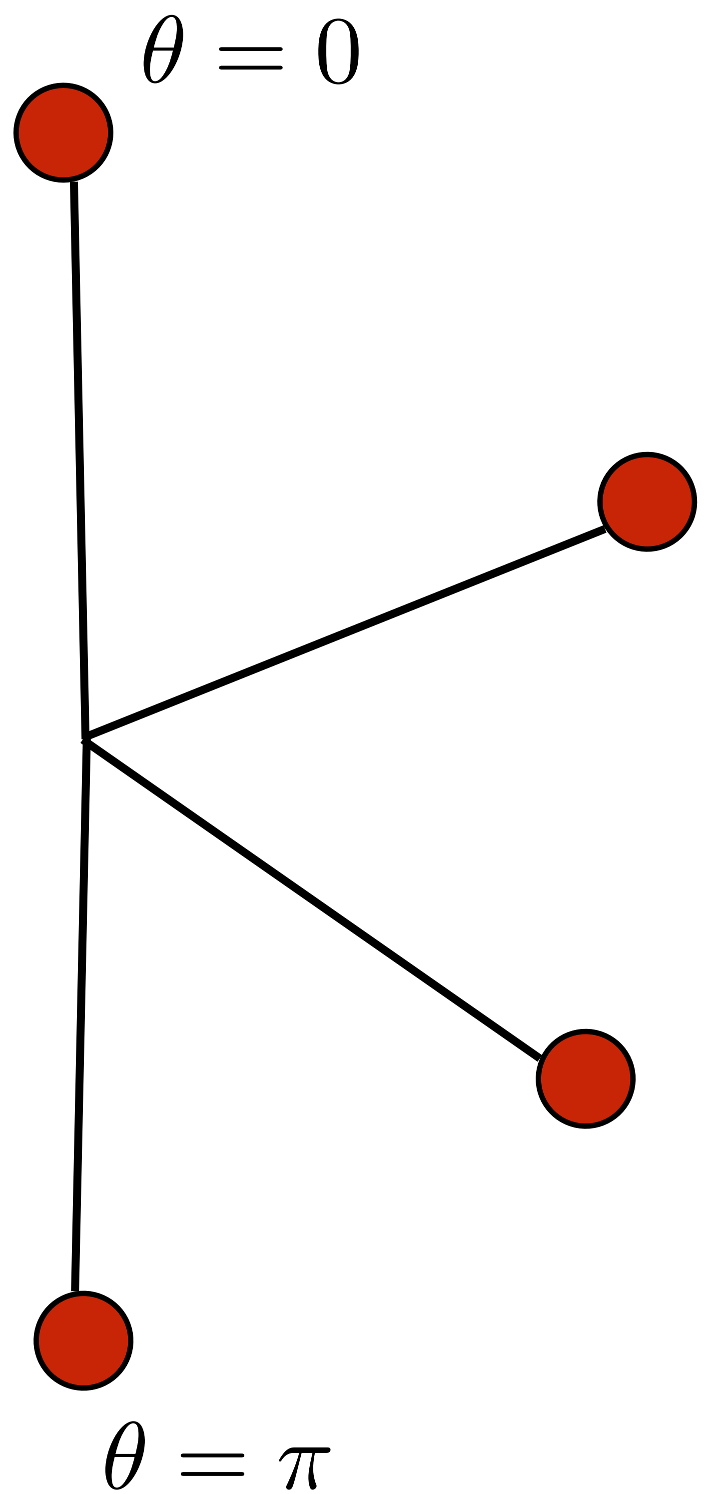
(Large angles imply large linearization error)



Can we use LQR to swing up a pendulum?

No!
(Large angles imply
large linearization error)

But we can track
a reference swing up trajectory
(small linearization error)



But, first we need to talk about time-varying systems

Today's objectives

1. LQR for time-varying systems
2. Trajectory following with iLQR
3. General nonlinear trajectory optimization with iLQR
4. Model predictive control (MPC)

LQR for Time-Varying Dynamical Systems

$$x_{t+1} = A_t x_t + B_t u_t$$

LQR for Time-Varying Dynamical Systems

$$x_{t+1} = A_t x_t + B_t u_t$$

$$c(x_t, u_t) = x_t^T Q_t x_t + u_t^T R_t u_t$$

LQR for Time-Varying Dynamical Systems

$$x_{t+1} = A_t x_t + B_t u_t$$

$$c(x_t, u_t) = x_t^T Q_t x_t + u_t^T R_t u_t$$

Straight forward to get LQR equations

$$K_t = -(R_t + B_t^T V_{t+1} B_t)^{-1} B_t^T V_{t+1} A_t$$

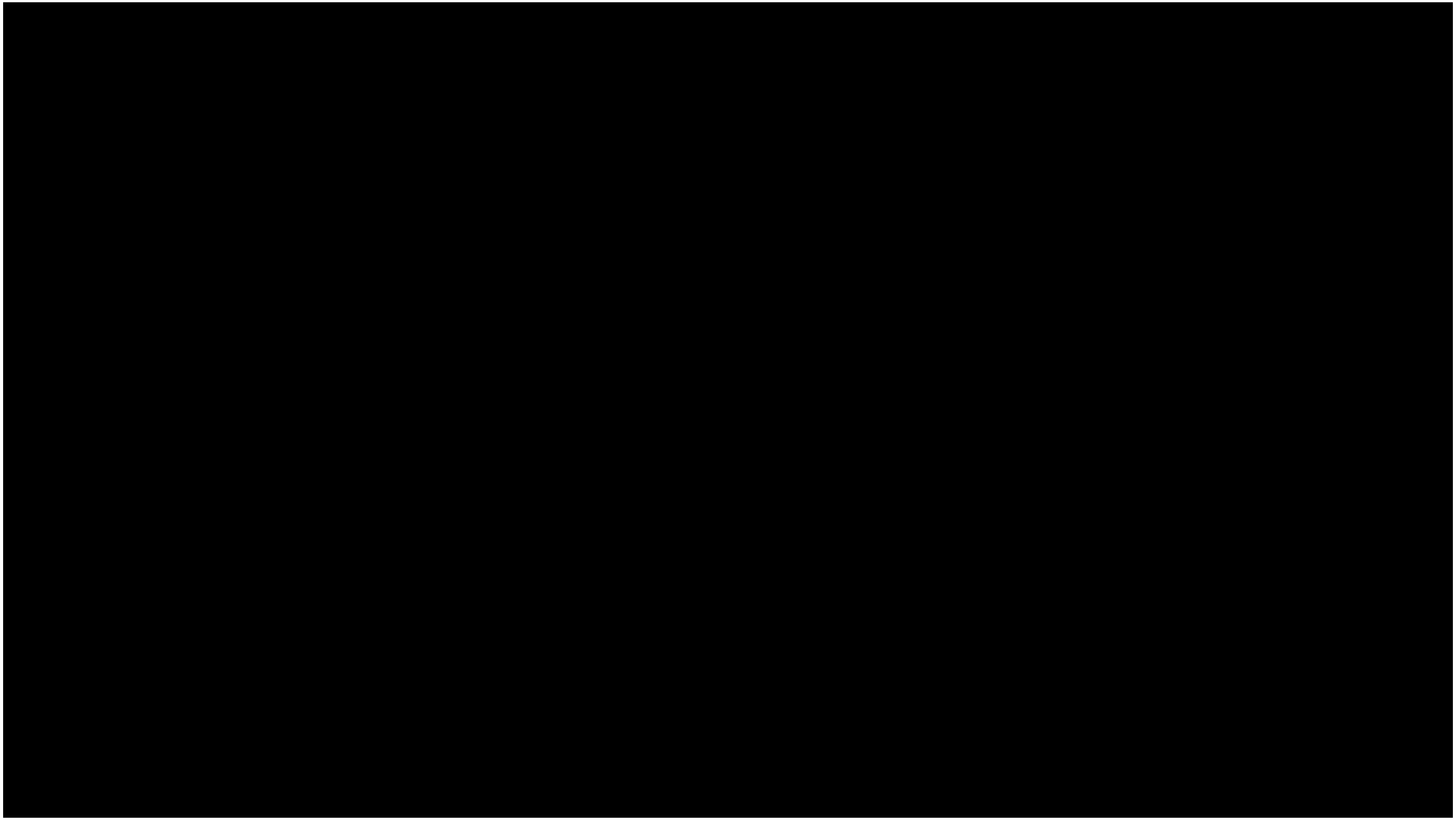
$$V_t = Q_t + K_t^T R_t K_t + (A_t + B_t K_t)^T V_{t+1} (A_t + B_t K_t)$$

Why do we care about time-varying?

Ans: Linearization about a trajectory

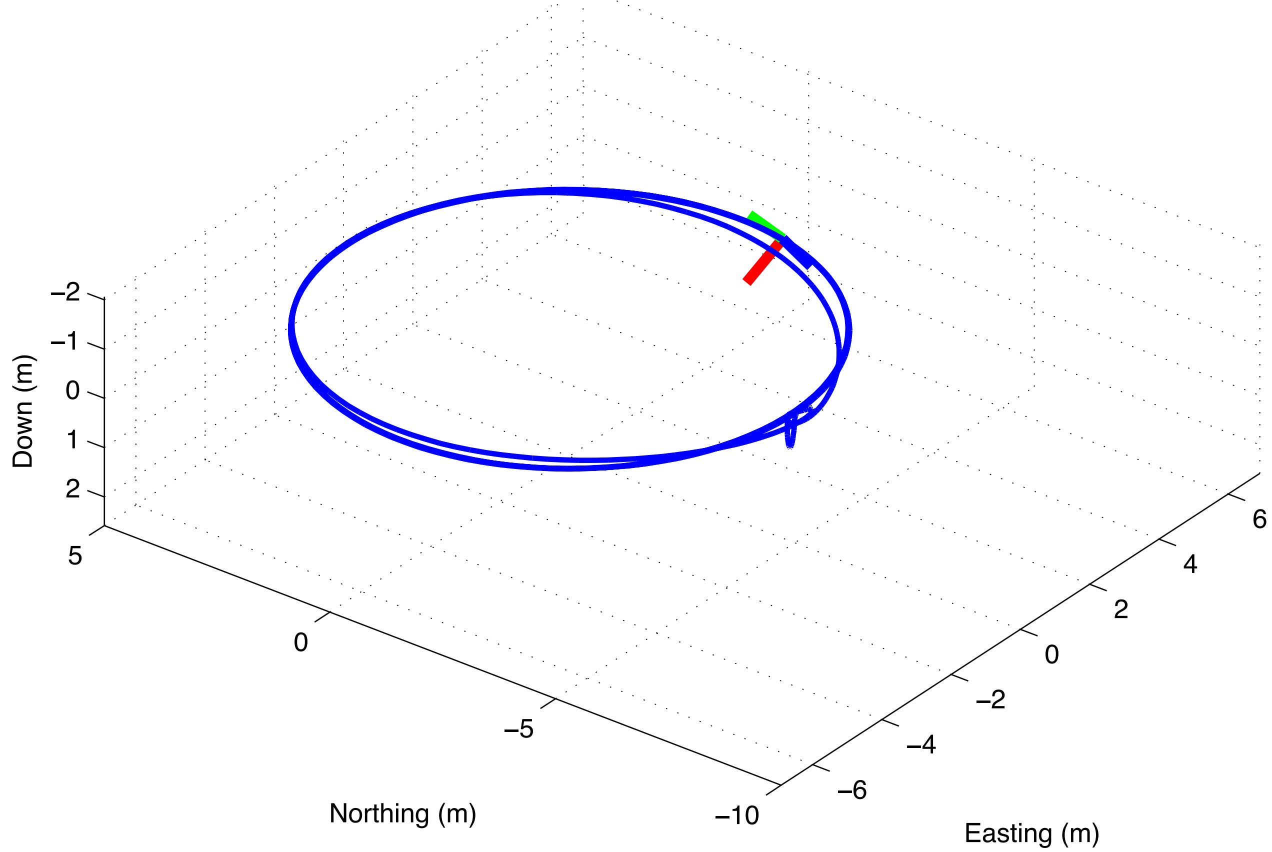
Trajectory tracking for stationary rolls?

Trajectory tracking for stationary rolls?

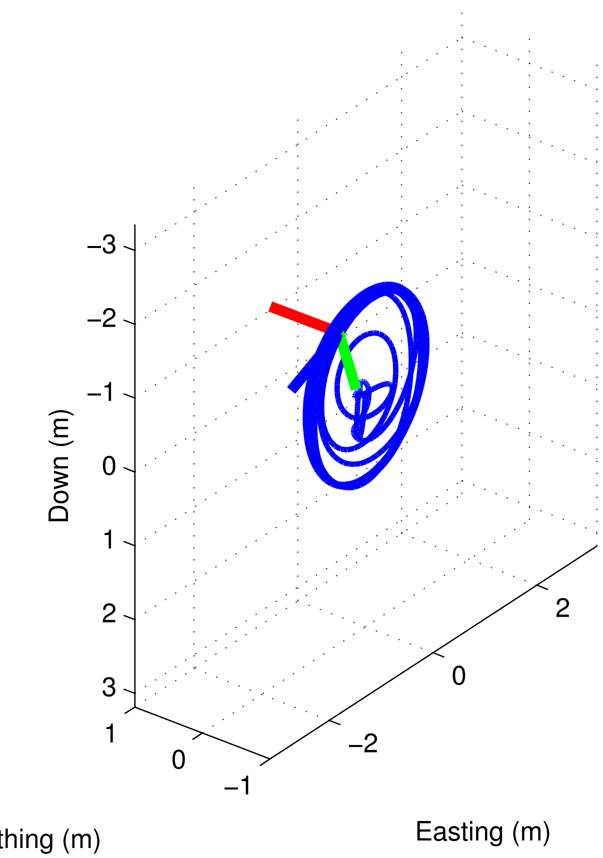


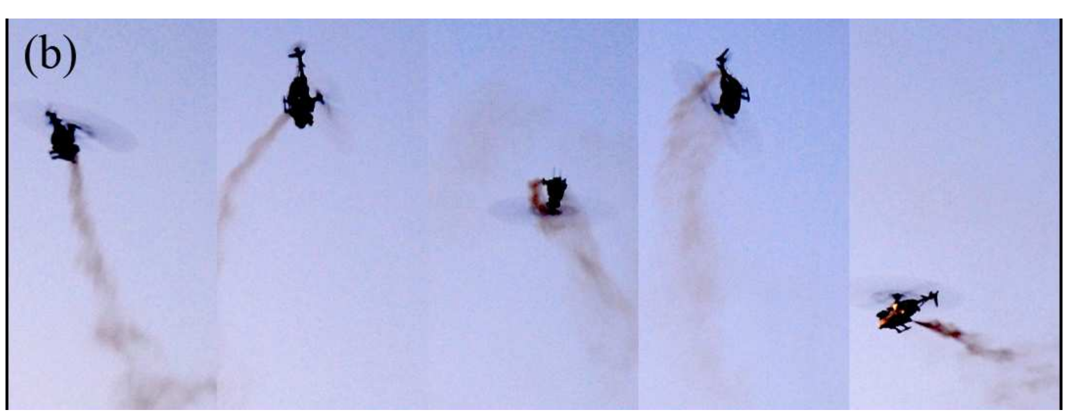
How do we get such behaviors?

Nose-in funnel



Stationary rolls





Task: Minimize tracking error

$$\min_{u_0, u_1, \ldots, u_{T-1}} \sum_{t=0}^{T-1} c(x_t, u_t)$$

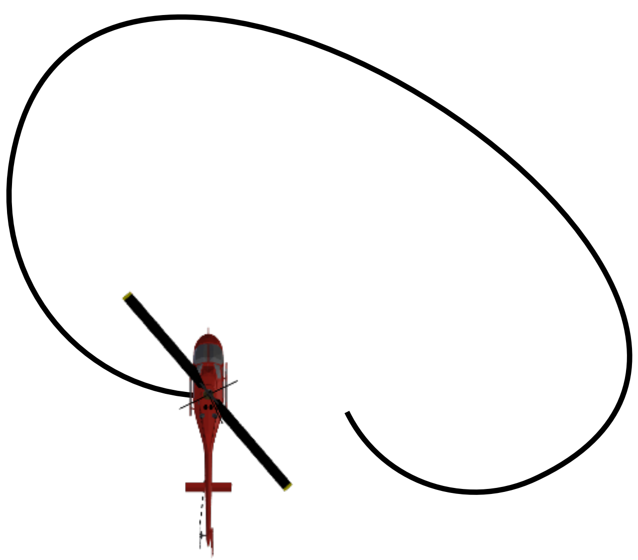
$$\text{subject to } \; x_{t+1} = f(x_t, u_t) \; \forall t$$

In this scenario, cost is simply a quadratic tracking cost

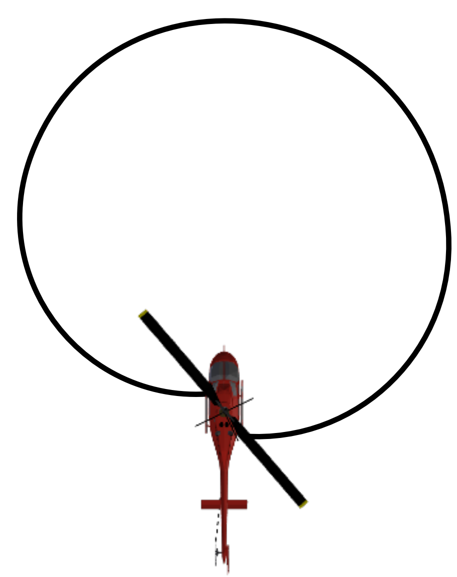
Why is this a hard optimization problem?

Iterative LQR (iLQR)

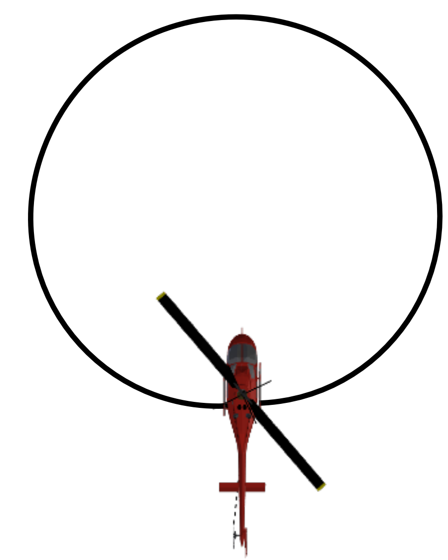
Start by guessing a control sequence, Forward simulate dynamics, Linearize about trajectory, Solve for new control sequence and repeat!



i=0



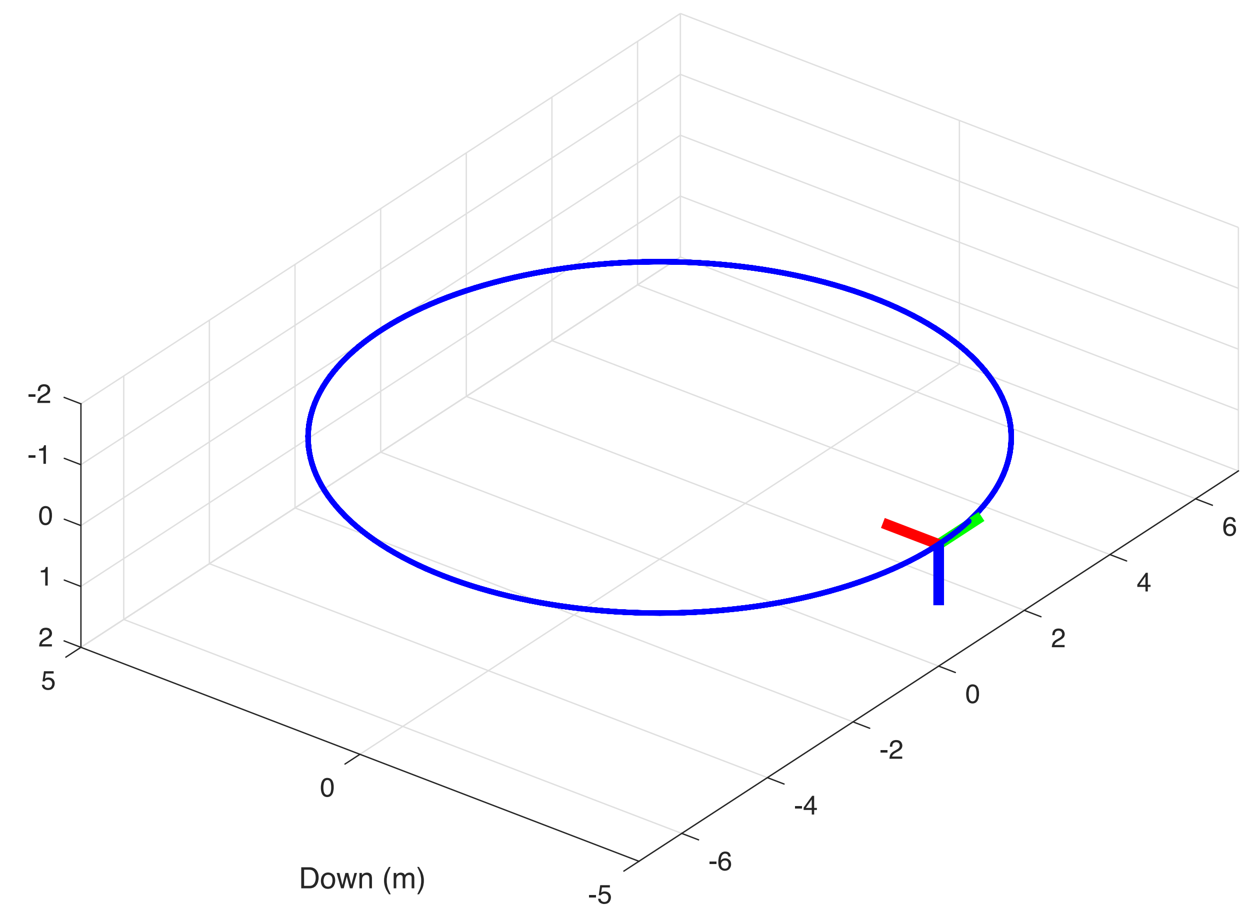
i=10



i=100

Step 1: Get a reference trajectory

$$x_0^{ref}, u_0^{ref}, x_1^{ref}, u_1^{ref}, \ldots, x_{T-1}^{ref}, u_{T-1}^{ref}$$



Note: Simply executing open loop trajectory wont work!

Step 2: Initialize your algorithm

Choose initial trajectory at iteration 0 to linearize about

$$x^0(t), u^0(t) = \{x_0^0, u_0^0, x_1^0, u_1^0, \ldots, x_{T-1}^0, u_{T-1}^0\}$$

Step 2: Initialize your algorithm

Choose initial trajectory at iteration 0 to linearize about

$$x^0(t), u^0(t) = \{x_0^0, u_0^0, x_1^0, u_1^0, \ldots, x_{T-1}^0, u_{T-1}^0\}$$

It's a good idea to choose the reference trajectory

Step 2: Initialize your algorithm

Choose initial trajectory at iteration 0 to linearize about

$$x^0(t), u^0(t) = \{x_0^0, u_0^0, x_1^0, u_1^0, \ldots, x_{T-1}^0, u_{T-1}^0\}$$

It's a good idea to choose the reference trajectory

Initialization is very important!

We will be perturbing this initial trajectory

Step 3: Linearize your dynamics!

At a given iteration i, we are going to linearize about

$$x_0^i, u_0^i, x_1^i, \dots$$

Step 3: Linearize your dynamics!

At a given iteration i, we are going to linearize about

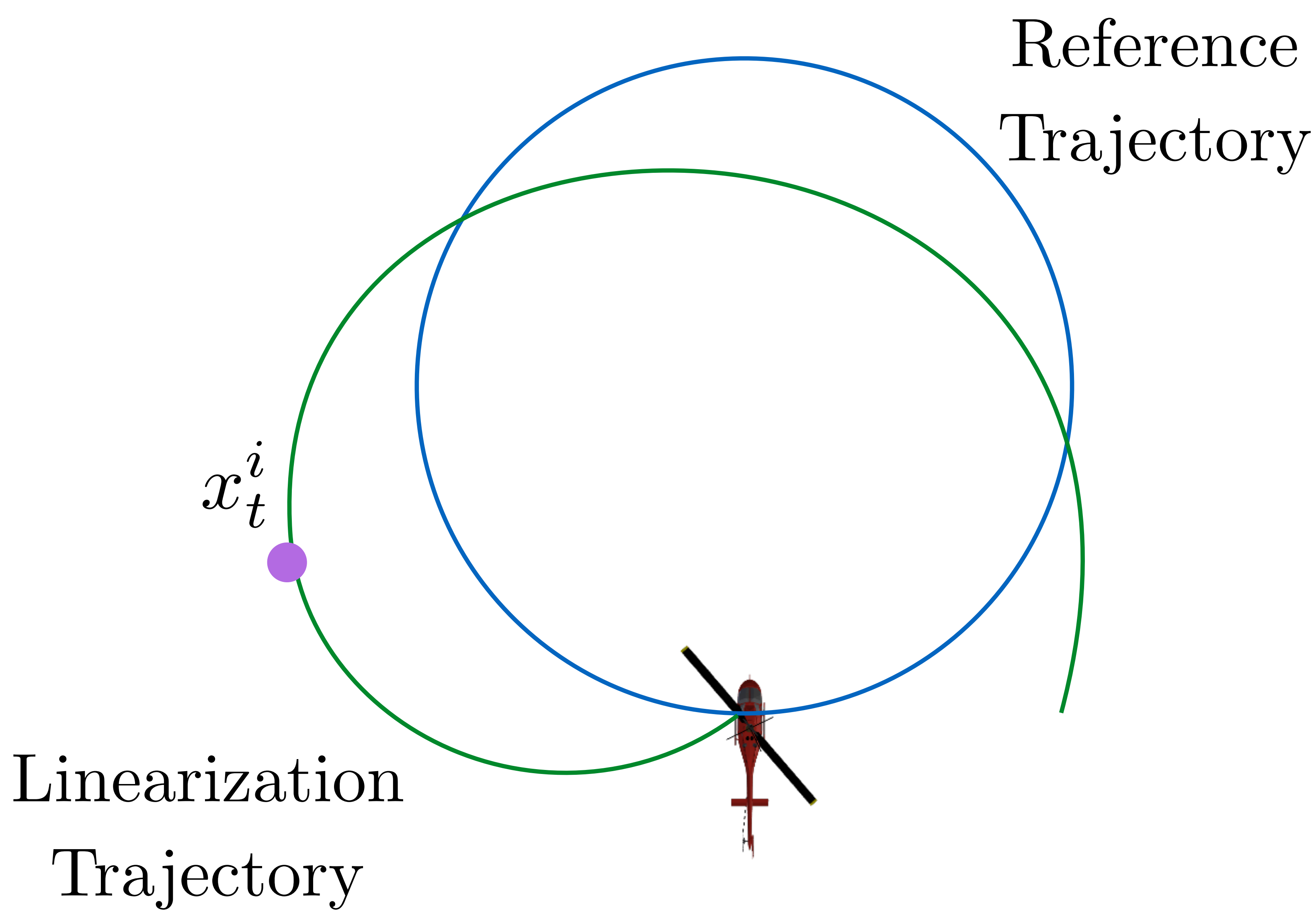
$$x_0^i, u_0^i, x_1^i, \ldots$$

Change of variable - we will track the delta perturbations

$$\delta x_t = x_t - x_t^i$$

$$\delta u_t = u_t - u_t^i$$

Step 3: Linearize your dynamics!



Step 3: Linearize your dynamics!

$$\delta x_t = x_t - x_t^i \qquad \delta u_t = u_t - u_t^i$$

Step 3: Linearize your dynamics!

$$\delta x_t = x_t - x_t^i \qquad \delta u_t = u_t - u_t^i$$

$$\delta x_{t+1} = A_t \delta x_t + B_t \delta u_t + \left(f(x_t^i, u_t^i) - x_{t+1}^i\right)$$

$$A_t = \left.\frac{\partial f}{\partial x}\right|_{x_t^i} \qquad B_t = \left.\frac{\partial f}{\partial u}\right|_{u_t^i}$$

Step 3: Linearize your dynamics!

$$\delta x_t = x_t - x_t^i \qquad \delta u_t = u_t - u_t^i$$

This is an affine system, not linear

$$\delta x_{t+1} = A_t \delta x_t + B_t \delta u_t + \left(f(x_t^i, u_t^i) - x_{t+1}^i\right)$$

$$A_t = \left.\frac{\partial f}{\partial x}\right|_{x_t^i} \qquad B_t = \left.\frac{\partial f}{\partial u}\right|_{u_t^i}$$

Step 3: Linearize your dynamics!

Homogenous coordinate system $\begin{bmatrix} \delta x_t \\ 1 \end{bmatrix}$

Affine dynamics is now linear!

$$\begin{bmatrix} \delta x_{t+1} \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} A_{t+1} & f(x_t^i, u_t^i) - x_{t+1}^i \\ 0 & 1 \end{bmatrix}}_{\tilde{A}_t} \begin{bmatrix} \delta x_t \\ 1 \end{bmatrix} + \underbrace{\begin{bmatrix} B_{t+1} \\ 0 \end{bmatrix}}_{\tilde{B}_t} \delta u_t$$

Step 4: Quadricize cost about trajectory

Our cost function is already quadratic, otherwise we would apply Taylor expansion

$$c(x_t, u_t) = (x_t - x_t^{ref})^T Q (x_t - x_t^{ref}) + (u_t - u_t^{ref})^T R (u_t - u_t^{ref})$$

Step 4: Quadricize cost about trajectory

Our cost function is already quadratic, otherwise we would apply Taylor expansion

$$c(x_t, u_t) = (x_t - x_t^{ref})^T Q (x_t - x_t^{ref}) + (u_t - u_t^{ref})^T R (u_t - u_t^{ref})$$

$$= \begin{bmatrix} \delta x_t \\ 1 \end{bmatrix}^T \underbrace{\begin{bmatrix} Q & Q(x_t^i - x_t^{ref}) \\ (x_t^i - x_t^{ref})^T Q & (x_t^i - x_t^{ref})^T (x_t^i - x_t^{ref}) \end{bmatrix}}_{\tilde{Q}_t} \begin{bmatrix} \delta x_t \\ 1 \end{bmatrix}$$

$$+$$

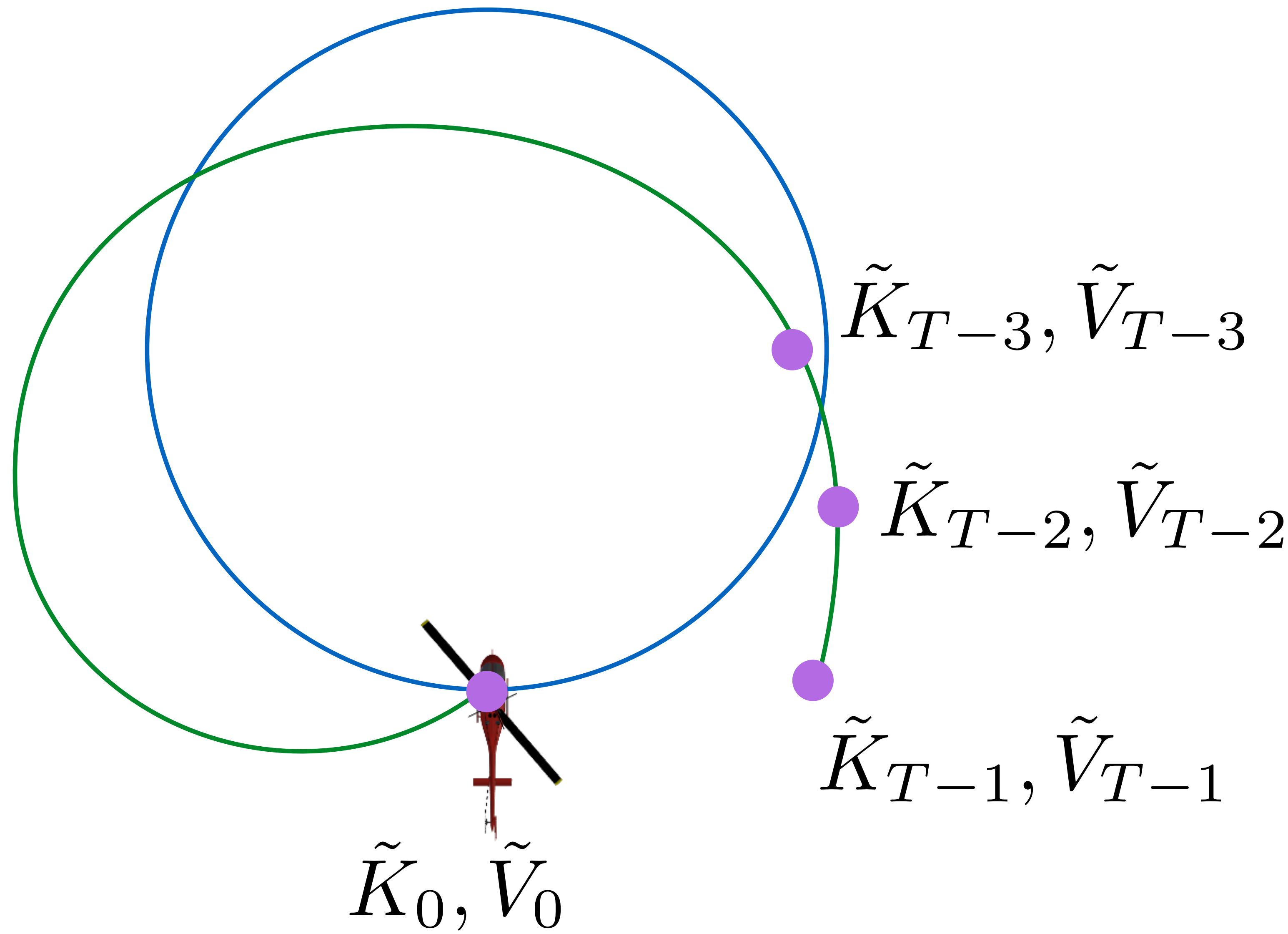
$$\begin{bmatrix} \delta u_t \\ 1 \end{bmatrix}^T \underbrace{\begin{bmatrix} R & R(u_t^i - u_t^{ref}) \\ (u_t^i - u_t^{ref})^T R & (u_t^i - u_t^{ref})^T (u_t^i - u_t^{ref}) \end{bmatrix}}_{\tilde{R}_t} \begin{bmatrix} \delta u_t \\ 1 \end{bmatrix}$$

We have all the ingredients to call LQR!

$$\tilde{K}_t = -(\tilde{R}_t + \tilde{B}_t^T \tilde{V}_{t+1} \tilde{B}_t)^{-1} \tilde{B}_t^T \tilde{V}_{t+1} \tilde{A}_t$$

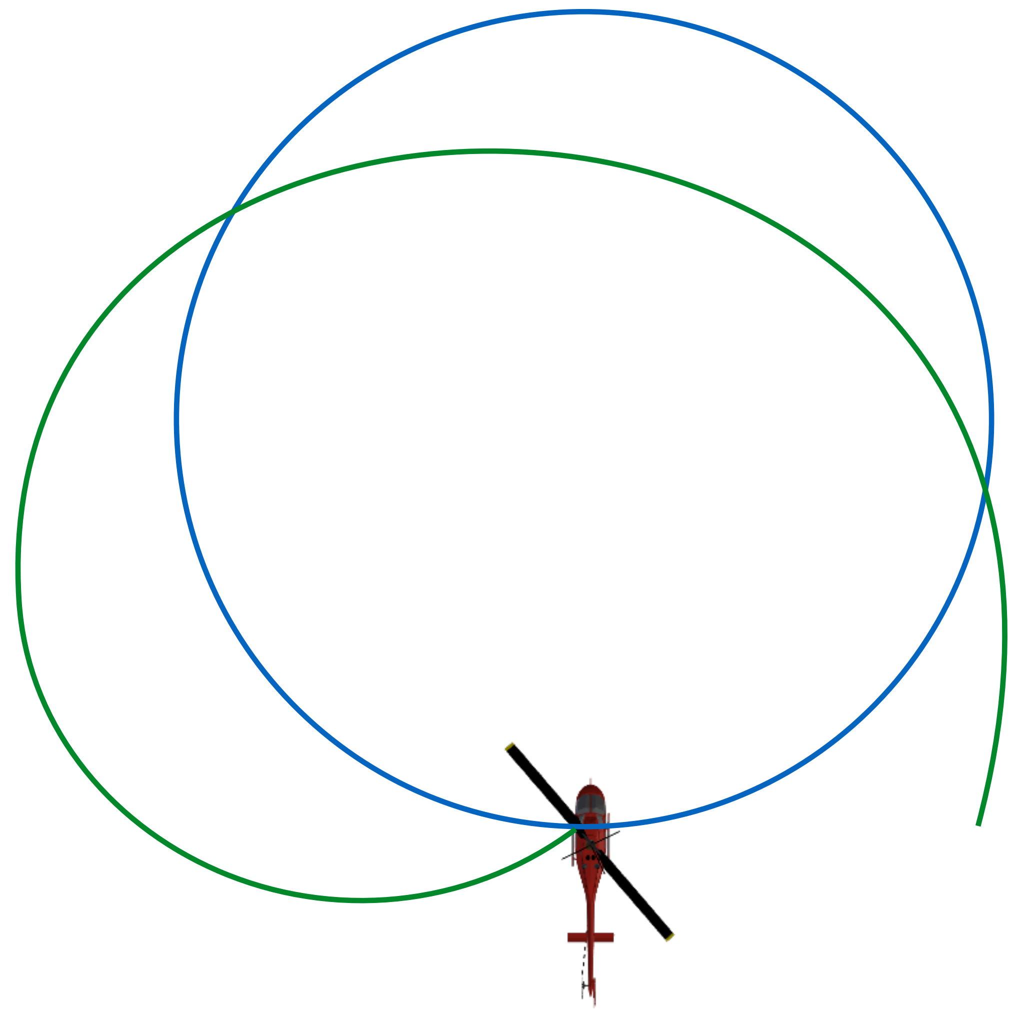
similarly calculate the value function ...

Step 5: Do a backward pass



Calculate controller gains for all time steps

Step 6: Do a forward pass to get new trajectory



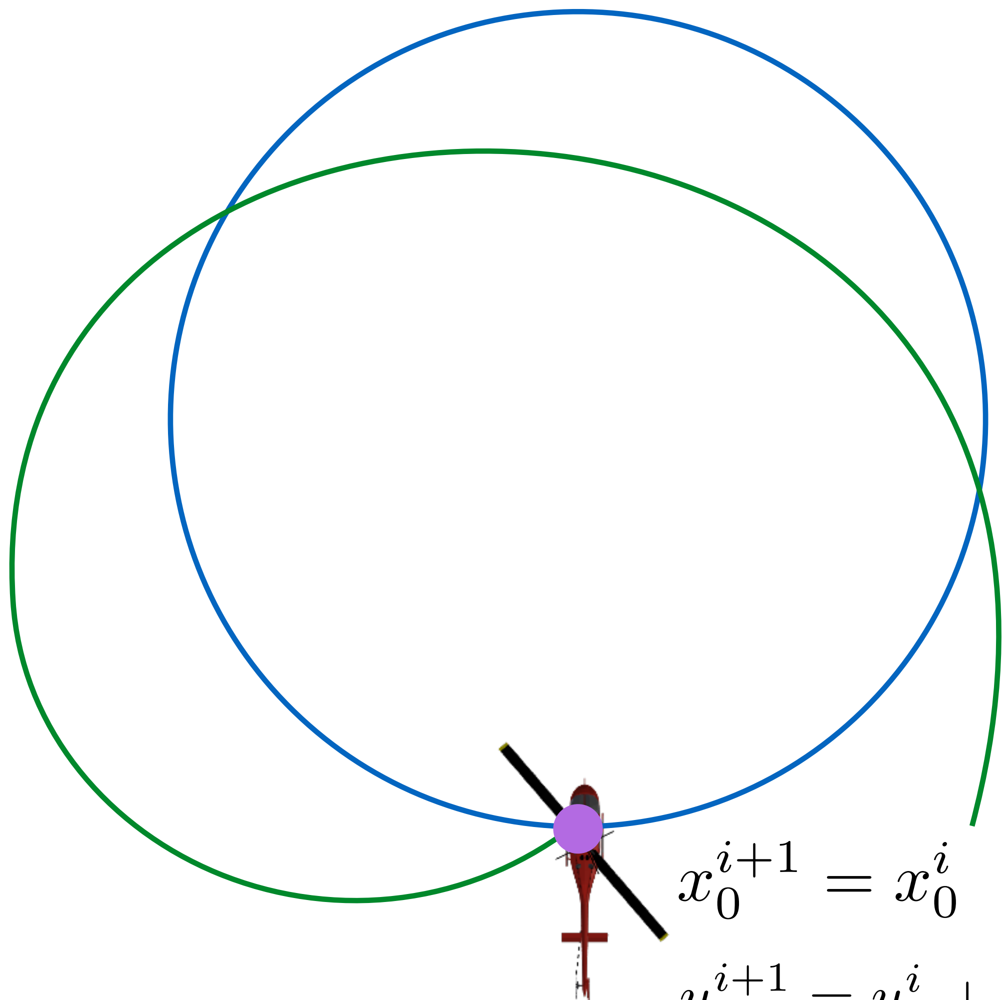
Compute
control action

$$u_t^{i+1} = u_t^i + \tilde{K}_t \begin{bmatrix} x_t^{i+1} - x_t^i \\ 1 \end{bmatrix}$$

Apply dynamics

$$x_t^{i+1} = f(x_t^{i+1}, u_t^{i+1})$$

Step 6: Do a forward pass to get new trajectory



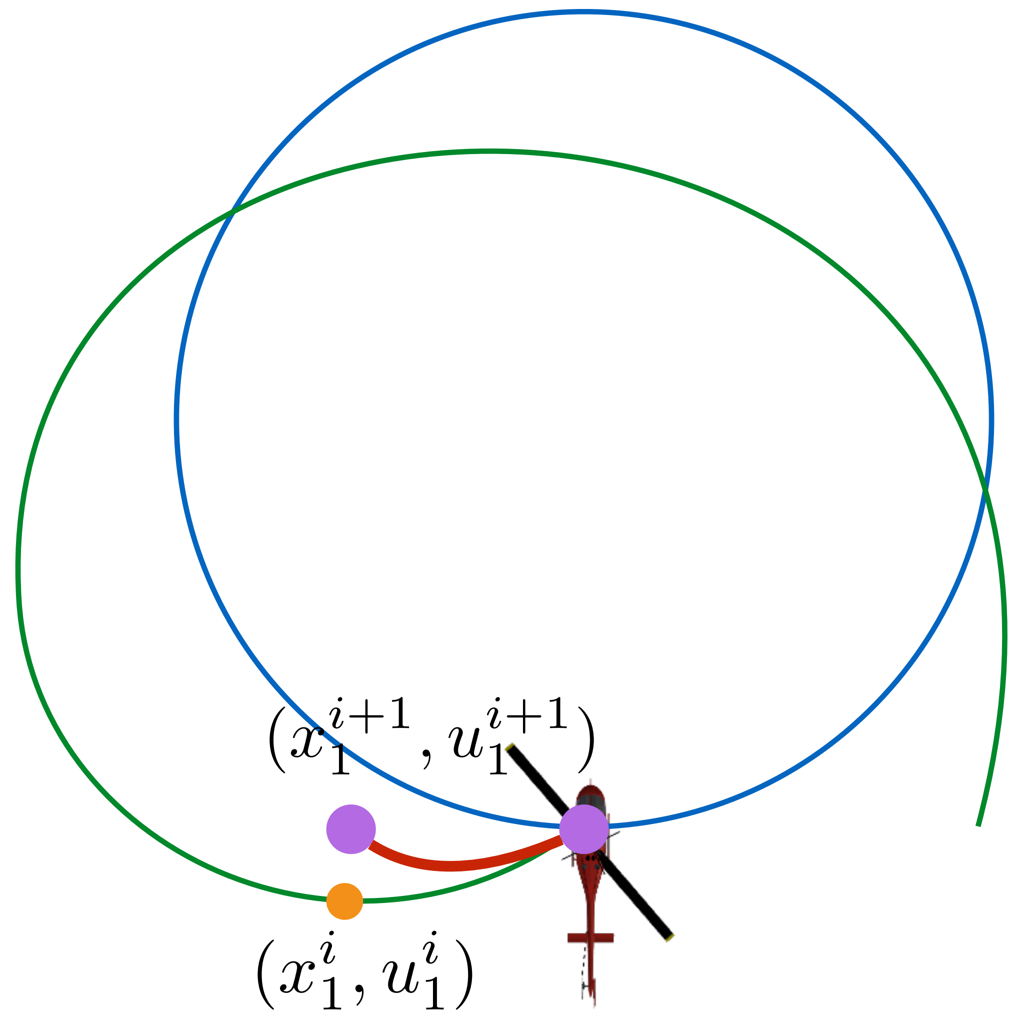
Compute control action

$$u_t^{i+1} = u_t^i + \tilde{K}_t \begin{bmatrix} x_t^{i+1} - x_t^i \\ 1 \end{bmatrix}$$

Apply dynamics

$$x_t^{i+1} = f(x_t^{i+1}, u_t^{i+1})$$

Step 6: Do a forward pass to get new trajectory



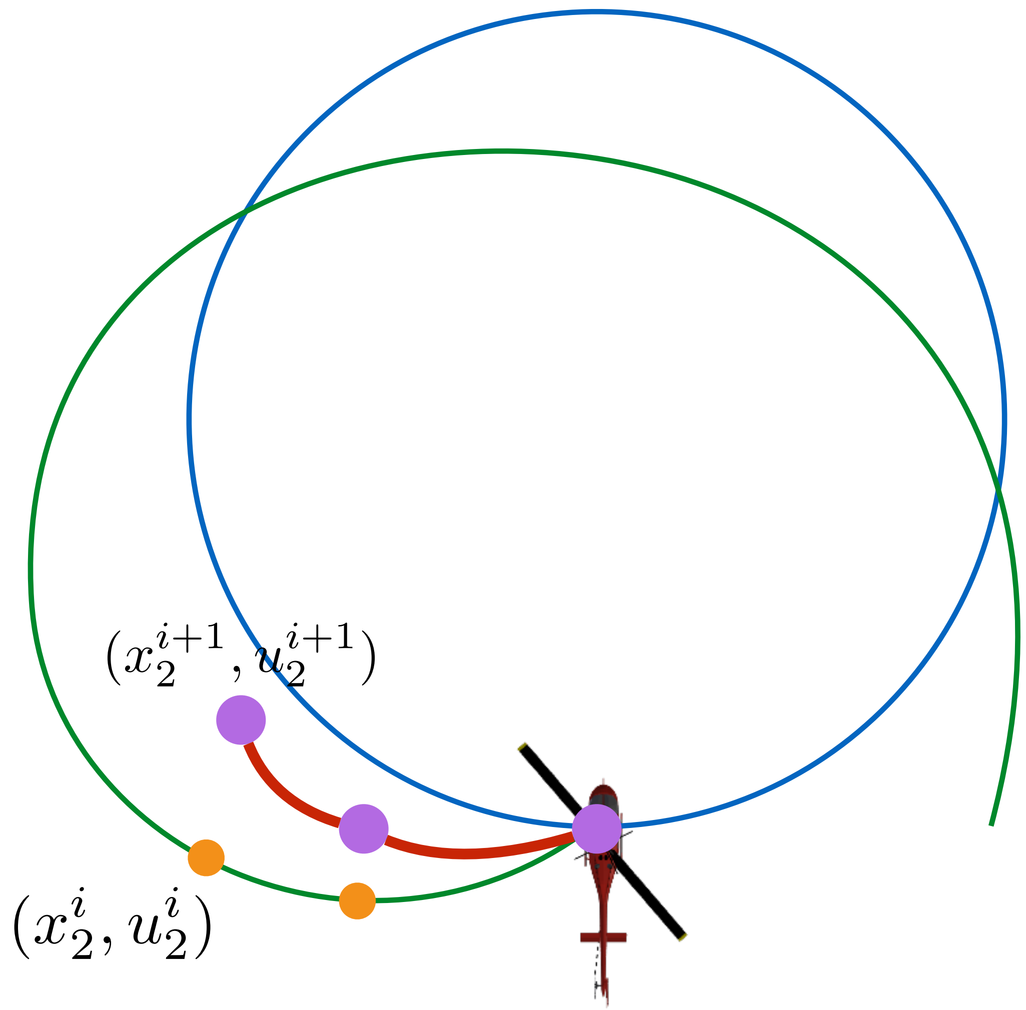
Compute
control action

$$u_t^{i+1} = u_t^i + \tilde{K}_t \begin{bmatrix} x_t^{i+1} - x_t^i \\ 1 \end{bmatrix}$$

Apply dynamics

$$x_t^{i+1} = f(x_t^{i+1}, u_t^{i+1})$$

Step 6: Do a forward pass to get new trajectory



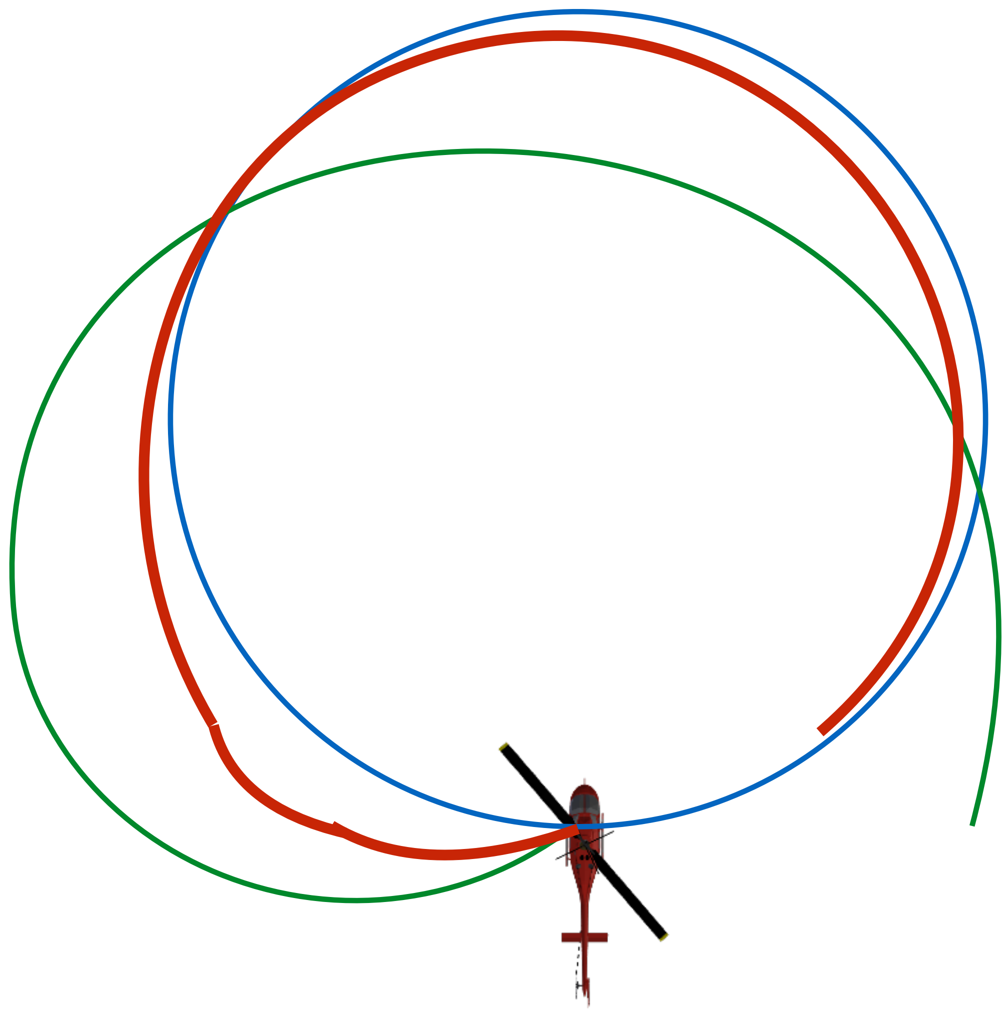
Compute
control action

$$u_t^{i+1} = u_t^i + \tilde{K}_t \begin{bmatrix} x_t^{i+1} - x_t^i \\ 1 \end{bmatrix}$$

Apply dynamics

$$x_t^{i+1} = f(x_t^{i+1}, u_t^{i+1})$$

Step 6: Do a forward pass to get new trajectory



Compute
control action

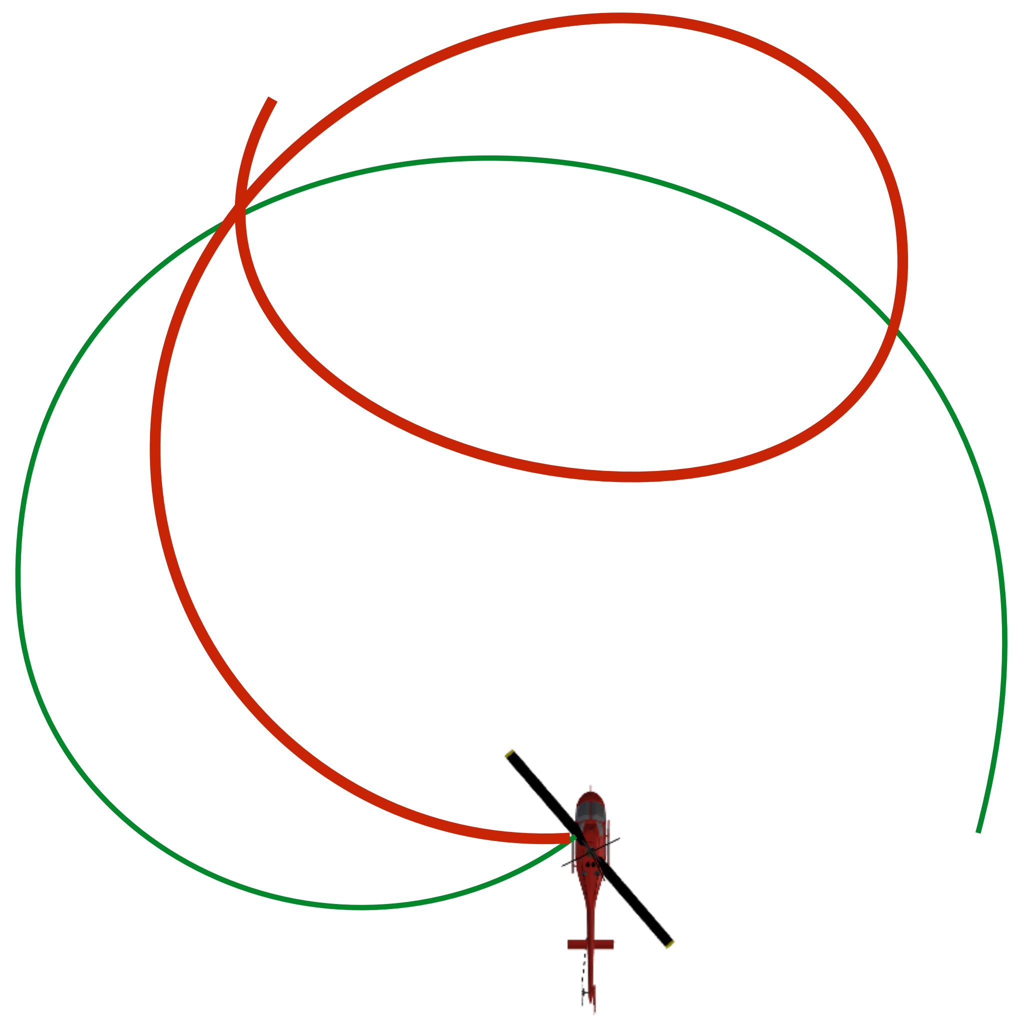
$$u_t^{i+1} = u_t^i + \tilde{K}_t \begin{bmatrix} x_t^{i+1} - x_t^i \\ 1 \end{bmatrix}$$

Apply dynamics

$$x_t^{i+1} = f(x_t^{i+1}, u_t^{i+1})$$

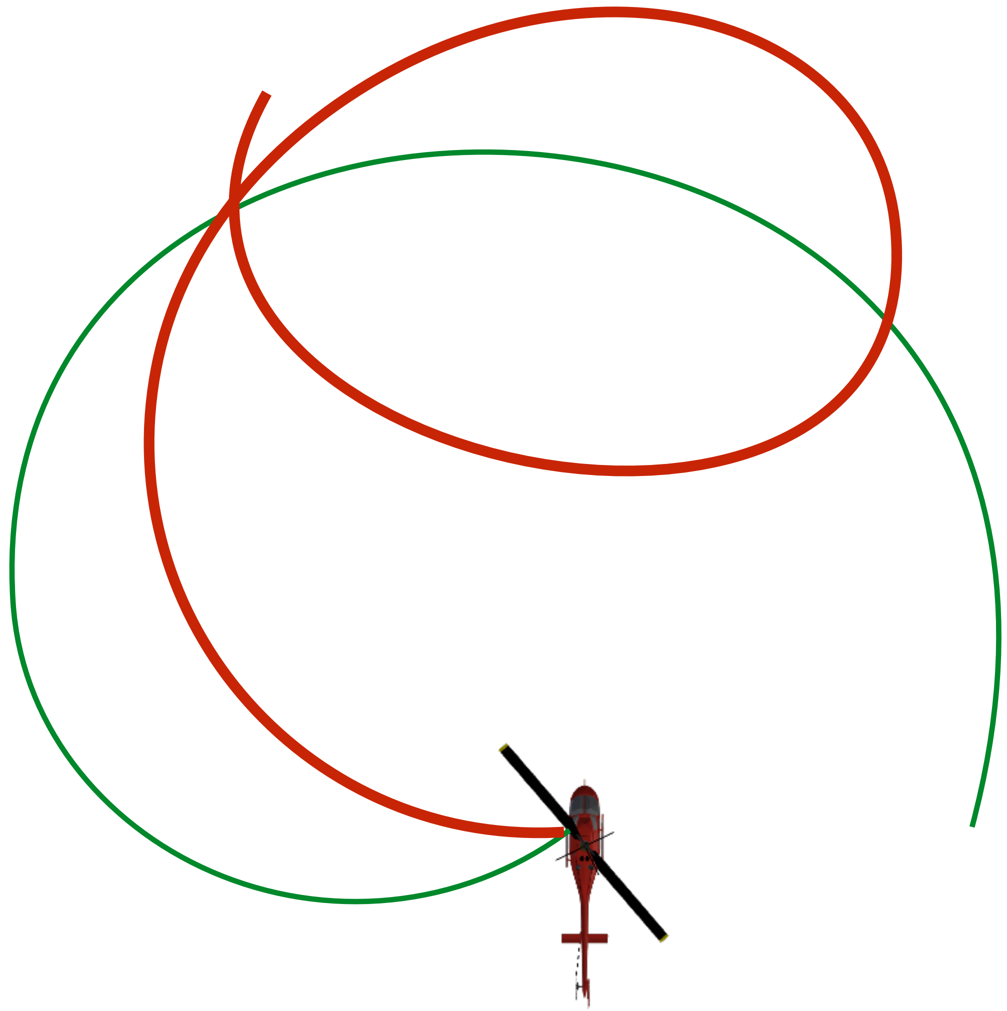
Problem: Forward pass will go bonkers

Why?



Problem: Forward pass will go bonkers

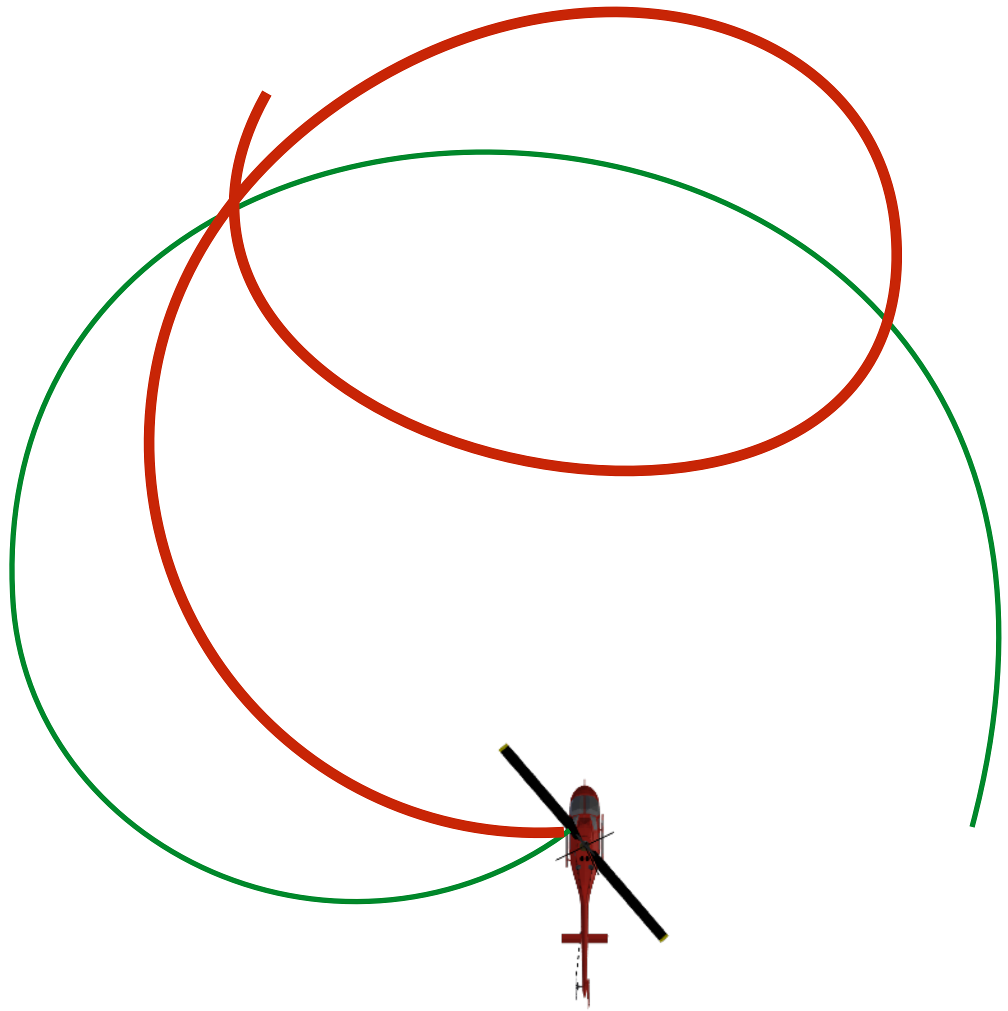
Why?



Linearization error gets bigger and bigger and bigger

Problem: Forward pass will go bonkers

Why?



Linearization error gets bigger and bigger and bigger

Remedies: Change cost function to penalize deviation from linearization

Questions

Questions

1. Can we solve LQR for continuous time dynamics?

Questions

1. Can we solve LQR for continuous time dynamics?

Yes! Refer to Continuous Algebraic Ricatti Equations (CARE)

Questions

1. Can we solve LQR for continuous time dynamics?

   Yes! Refer to Continuous Algebraic Ricatti Equations (CARE)

2. Can LQR handle arbitrary costs (not just tracking)?

Questions

1. Can we solve LQR for continuous time dynamics?

   Yes! Refer to Continuous Algebraic Ricatti Equations (CARE)

2. Can LQR handle arbitrary costs (not just tracking)?

   Yes! Just quadricize the cost

Questions

1. Can we solve LQR for continuous time dynamics?

   Yes! Refer to Continuous Algebraic Ricatti Equations (CARE)

2. Can LQR handle arbitrary costs (not just tracking)?

   Yes! Just quadricize the cost

3. What if I want to penalize control derivatives?

Questions

1. Can we solve LQR for continuous time dynamics?

   Yes! Refer to Continuous Algebraic Ricatti Equations (CARE)

2. Can LQR handle arbitrary costs (not just tracking)?

   Yes! Just quadricize the cost

3. What if I want to penalize control derivatives?

   No problem! Add control as part of state space

Questions

1. Can we solve LQR for continuous time dynamics?

   Yes! Refer to Continuous Algebraic Ricatti Equations (CARE)

2. Can LQR handle arbitrary costs (not just tracking)?

   Yes! Just quadricize the cost

3. What if I want to penalize control derivatives?

   No problem! Add control as part of state space

4. Can we handle noisy dynamics?

Questions

1. Can we solve LQR for continuous time dynamics?

    Yes! Refer to Continuous Algebraic Ricatti Equations (CARE)

2. Can LQR handle arbitrary costs (not just tracking)?

    Yes! Just quadricize the cost

3. What if I want to penalize control derivatives?

    No problem! Add control as part of state space

4. Can we handle noisy dynamics?

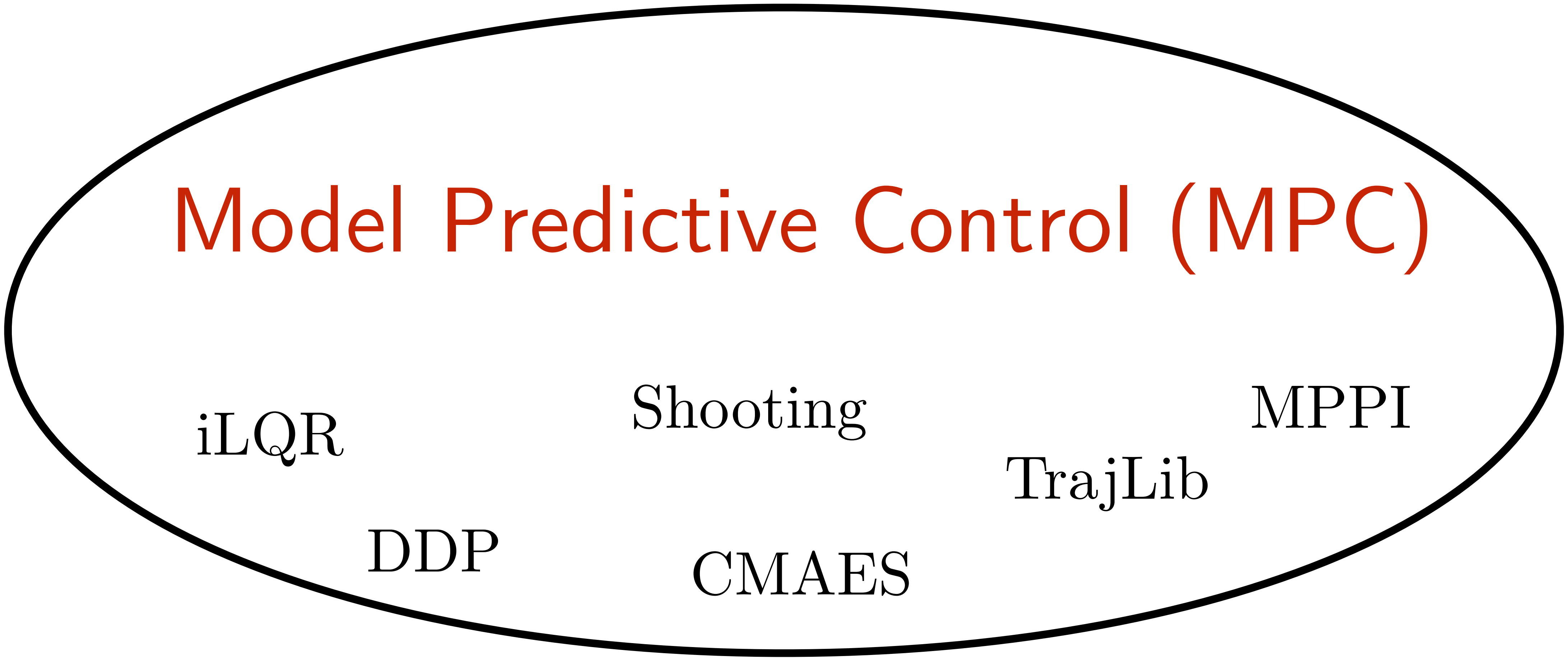
    Yes! Gaussian noise does not change the answer

Table of Controllers

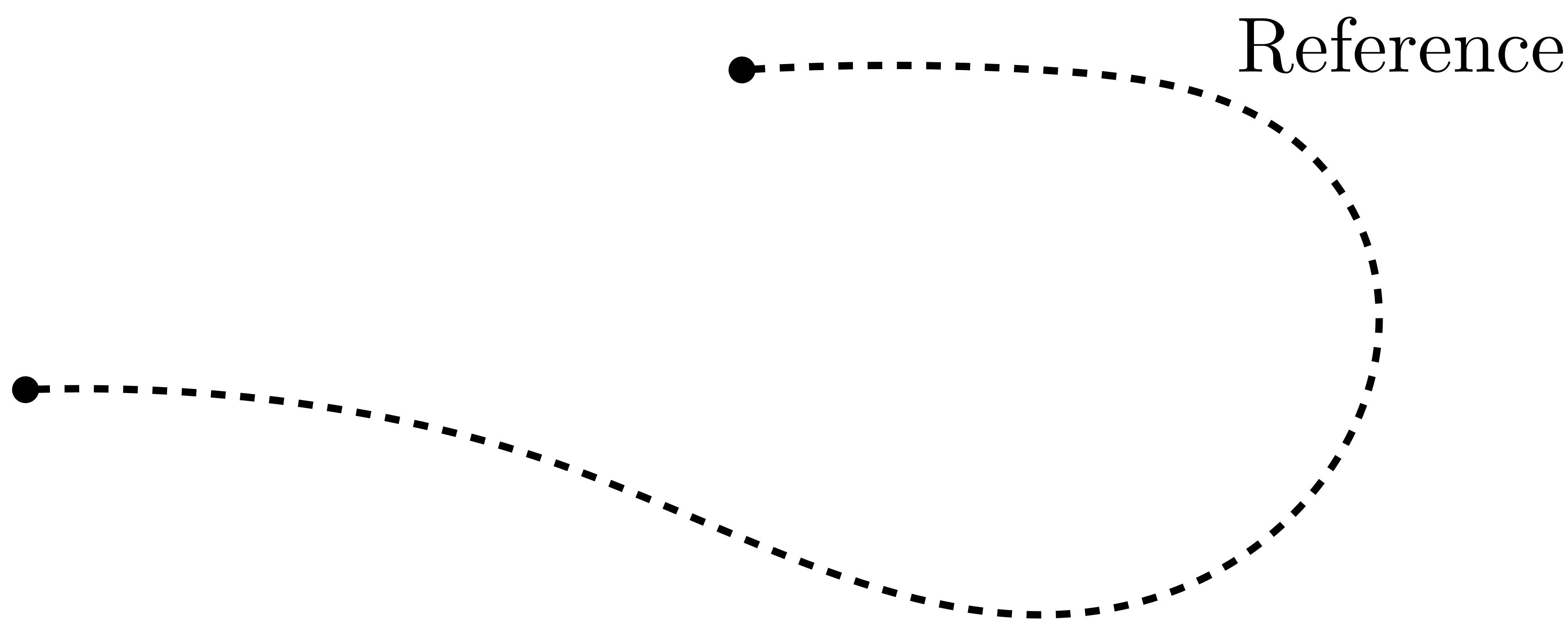
	Control Law	Uses model	Stability Guarantee	Minimize Cost
PID		No	No	No
Pure Pursuit		Circular arcs	Yes - with assumptions	No
Lyapunov		Non-linear	Yes	No
LQR		Linear	Yes	Quadratic
iLQR		Non-linear	Yes	Yes

iLQR is just one technique

It's far from perfect - can't deal with model errors / constraints ...

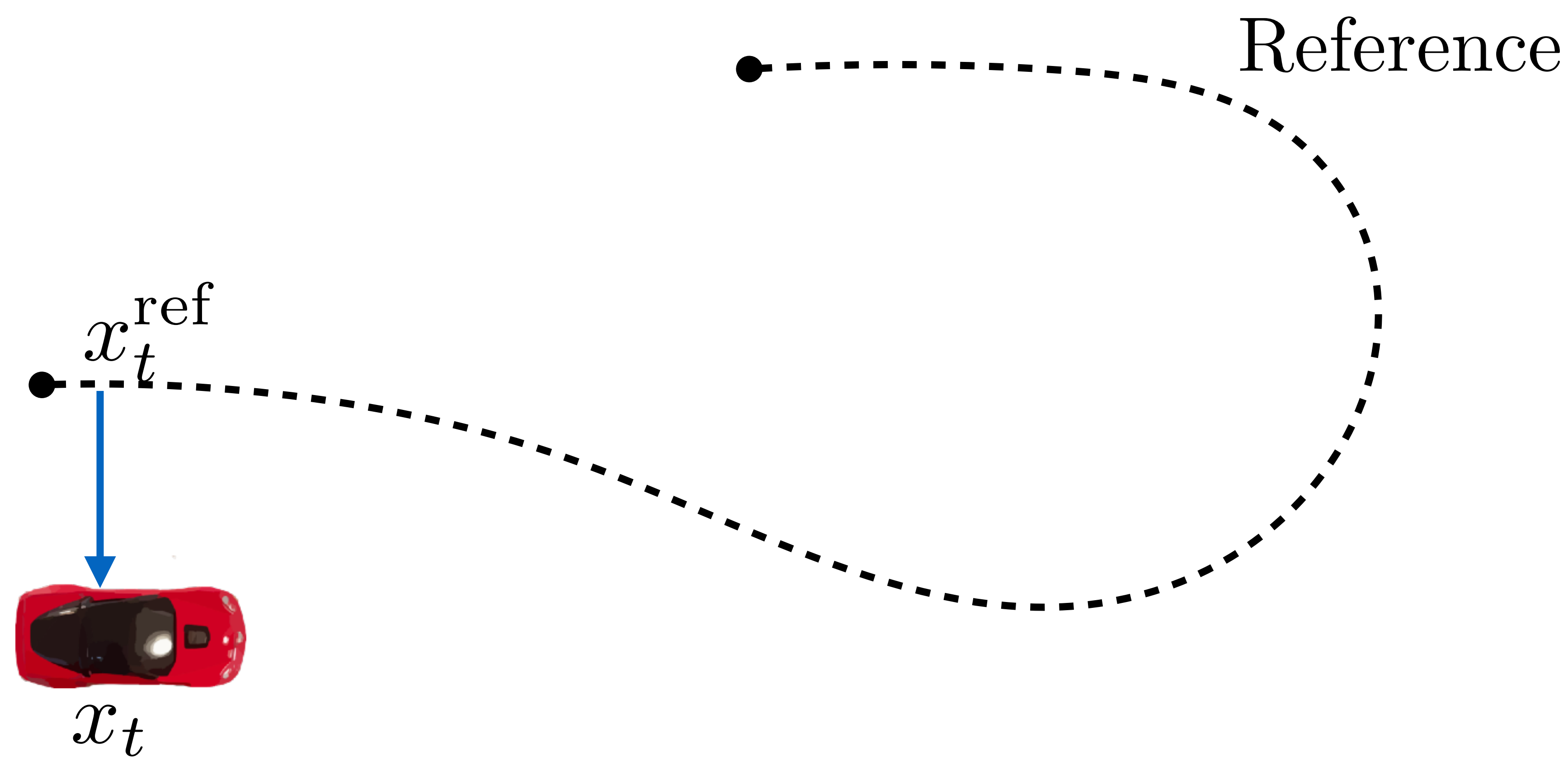


Recap: Feedback control framework



Look at current state error and compute control actions

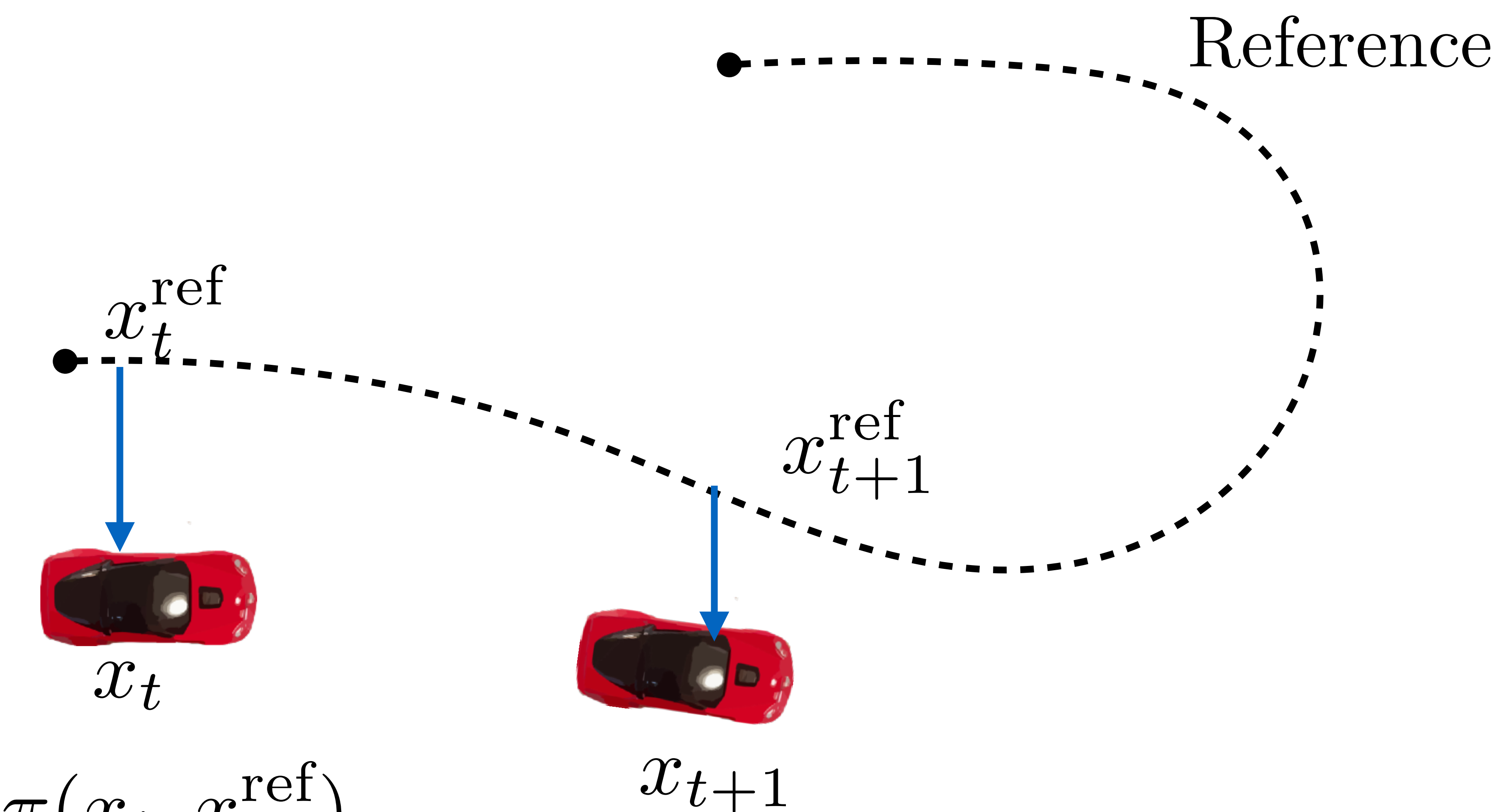
Recap: Feedback control framework



$$u_t = \pi(x_t, x_t^{\text{ref}})$$

Look at current state error and compute control actions

Recap: Feedback control framework

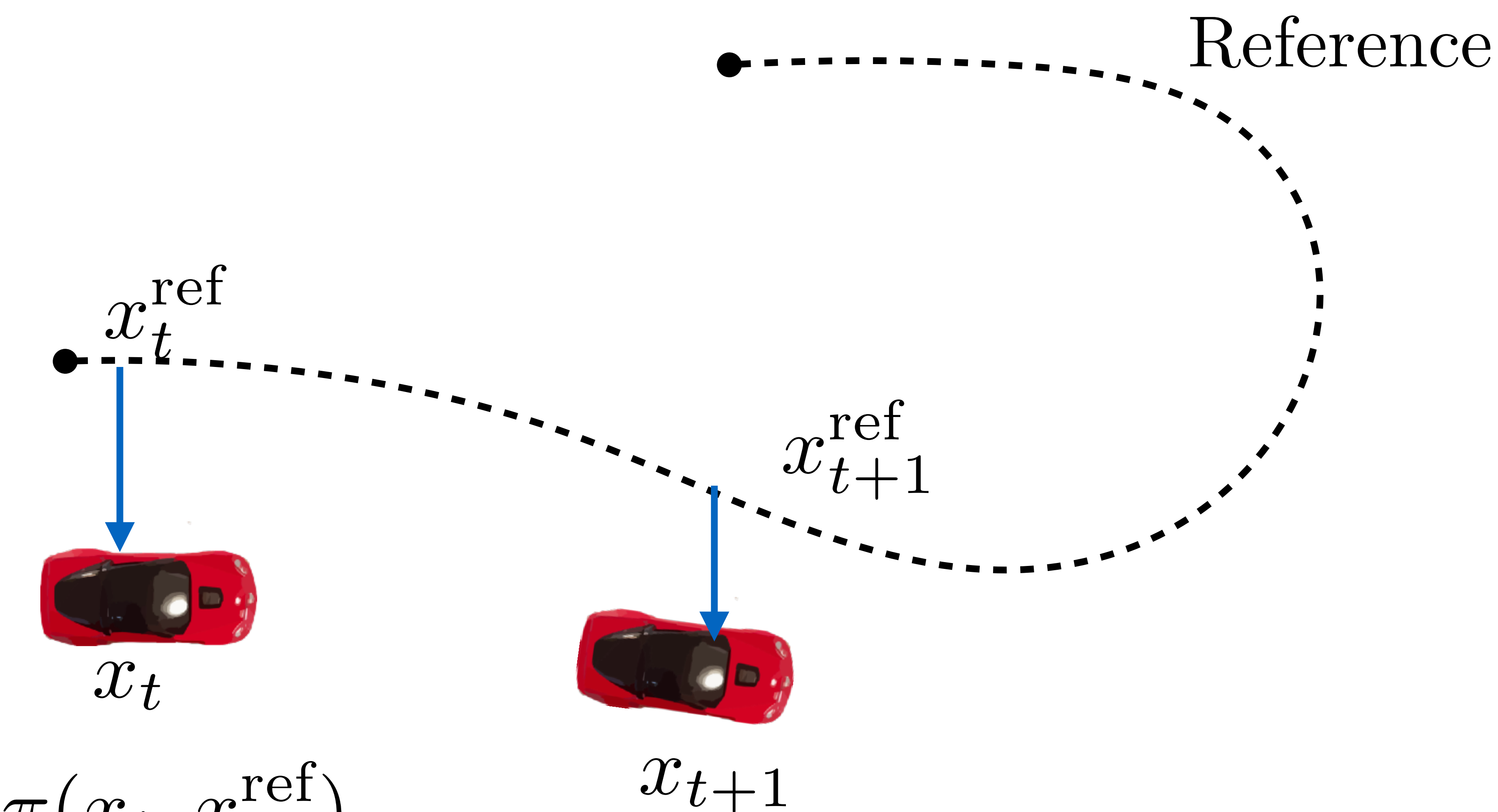


$$u_t = \pi(x_t, x_t^{\text{ref}})$$

$$u_{t+1} = \pi(x_{t+1}, x_{t+1}^{\text{ref}})$$

Look at current state error and compute control actions

Recap: Feedback control framework



$$u_t = \pi(x_t, x_t^{\text{ref}})$$

$$u_{t+1} = \pi(x_{t+1}, x_{t+1}^{\text{ref}})$$

Look at current state error and compute control actions

Goal: To drive error to 0 … to optimally drive it to 0

Limitations of this framework

A fixed control law that looks at instantaneous feedback

$$u_t = \underbrace{\pi}_{\text{Fixed}}(x_t, \underbrace{x_t^{\text{ref}}}_{\text{Reference}})$$

Why is it so difficult to create a magic control law?

Problem 1: What if we have constraints?

Simple scenario: Car tracking a straight line

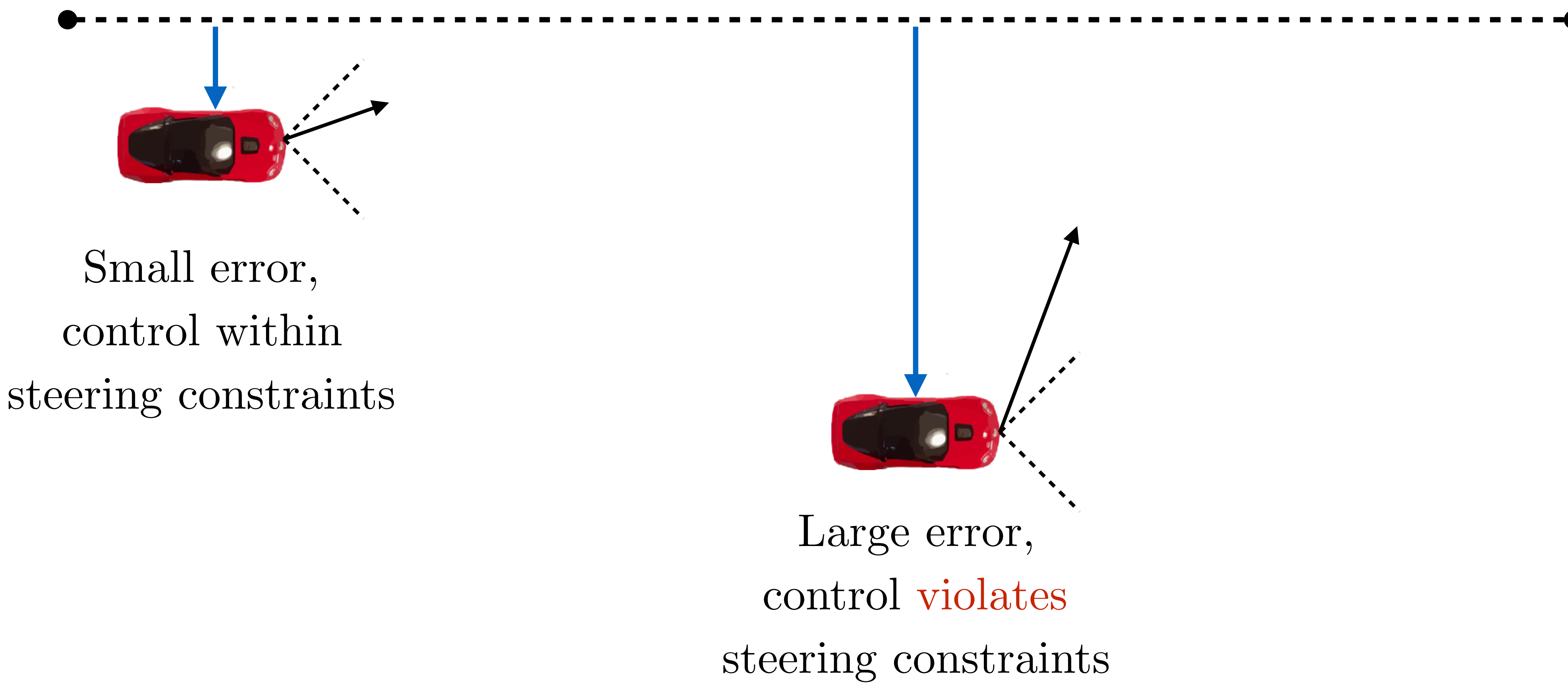
Problem 1: What if we have constraints?

Simple scenario: Car tracking a straight line

Small error,
control within
steering constraints

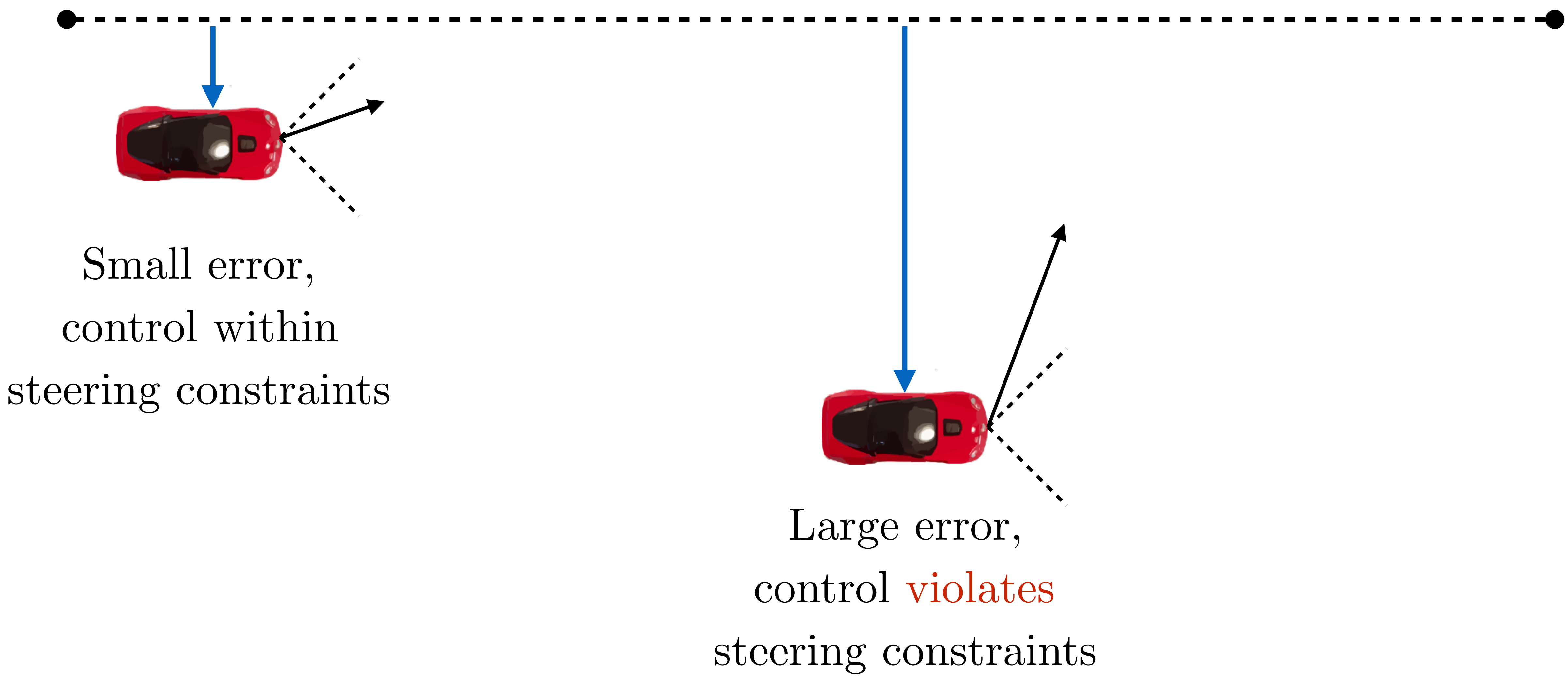
Problem 1: What if we have constraints?

Simple scenario: Car tracking a straight line



Problem 1: What if we have constraints?

Simple scenario: Car tracking a straight line



We could "clamp control command" ...
but what are the implications?

General problem: Complex models

Dynamics $$x_{t+1} = f(x_t, u_t)$$

Constraints $$g(x_t, u_t) \leq 0$$

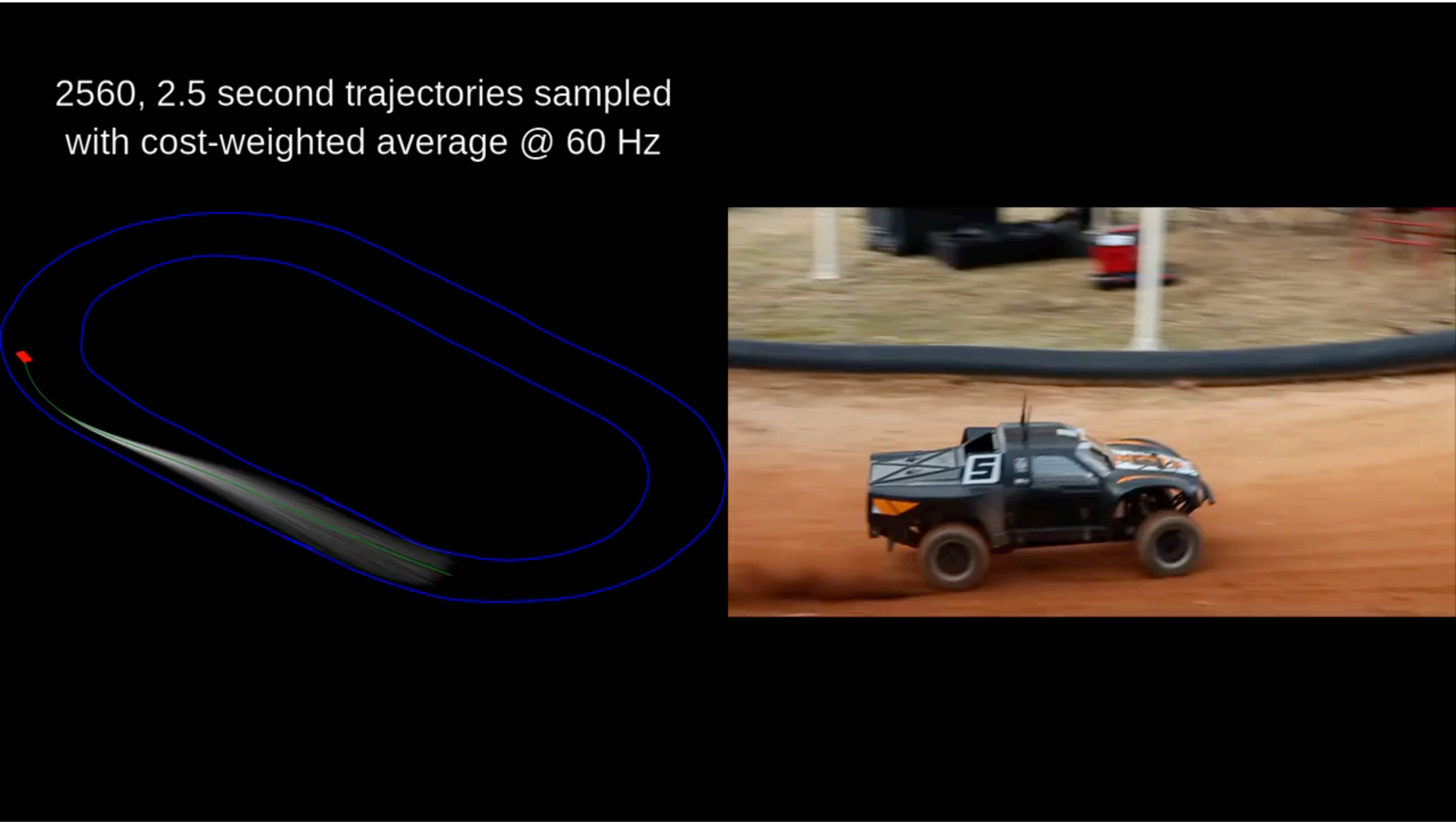
Such complex models imply we need to:

1. Predict the implications of control actions

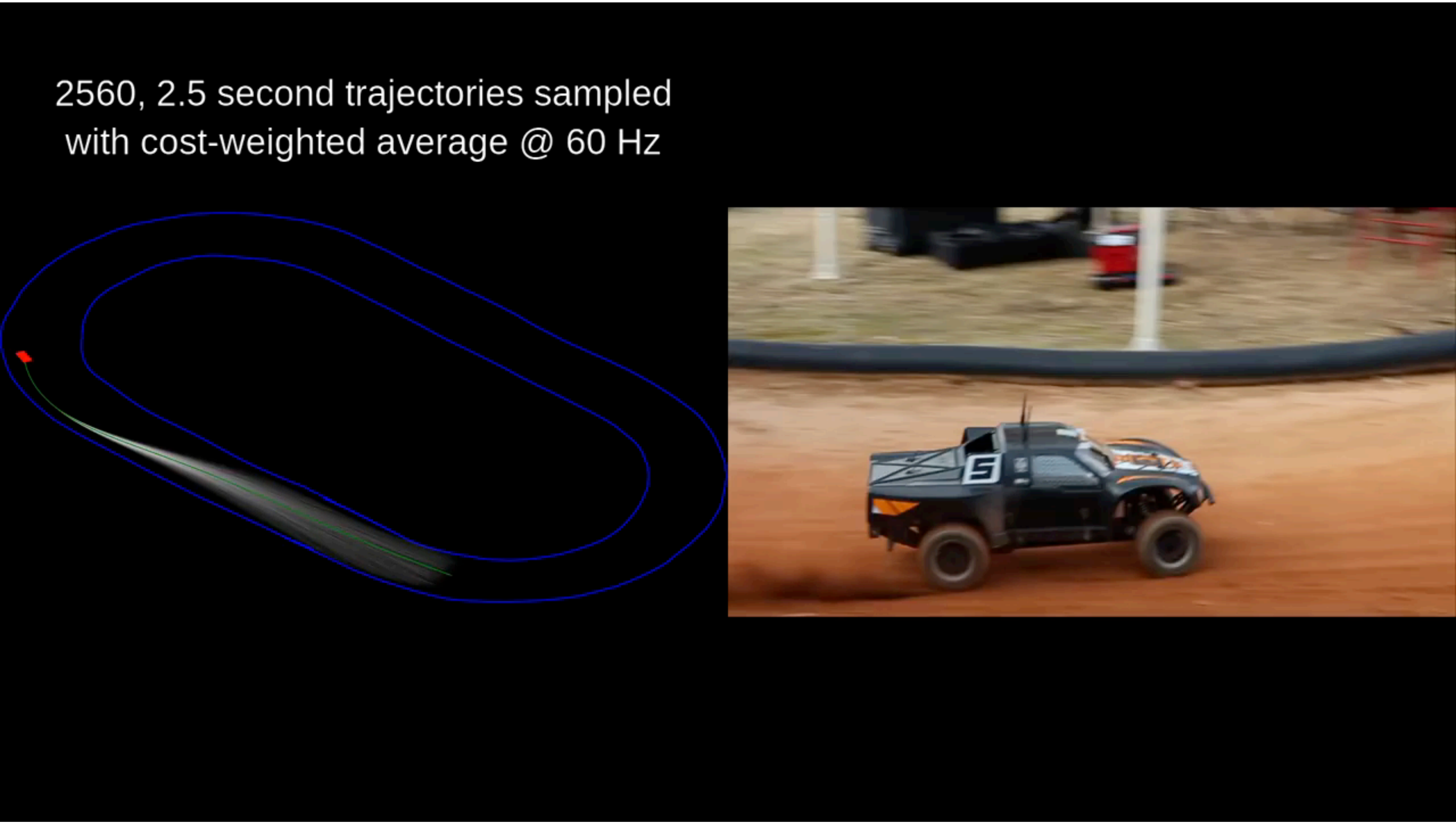
2. Do corrections NOW that would affect the future

3. It may not be possible to find one law - might need to predict

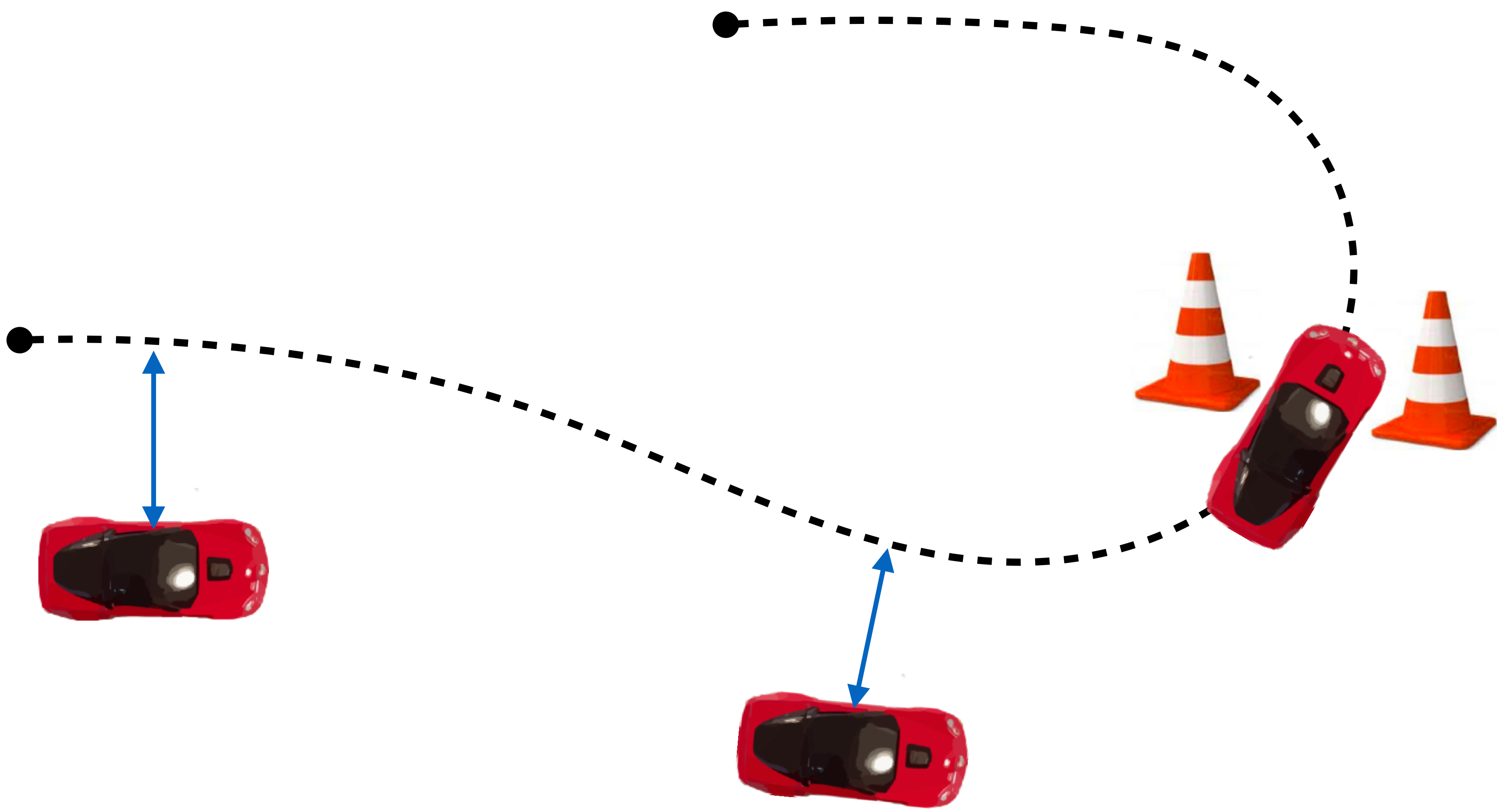
Example: Rough terrain mobility



Example: Rough terrain mobility



Problem 2: What if some errors are worse than others?



We need a cost function that penalizes states non-uniformly

Key Idea:

Frame control as an optimization problem

Model predictive control (MPC)

1. Plan a sequence of control actions

2. Predict the set of next states unto a horizon H

3. Evaluate the cost / constraint of the states and controls

4. Optimize the cost

Model predictive control (MPC)

$$\min_{u_{t+1},\ldots u_{t+H}} \sum_{k=t}^{t+H-1} J(x_k, u_{k+1})$$

(plan till horizon H) (Cost)

$$x_{k+1} = f(x_k, u_{k+1})$$

(Predict next state with dynamics)

$$g(x_k, u_{k+1}) \leq 0$$

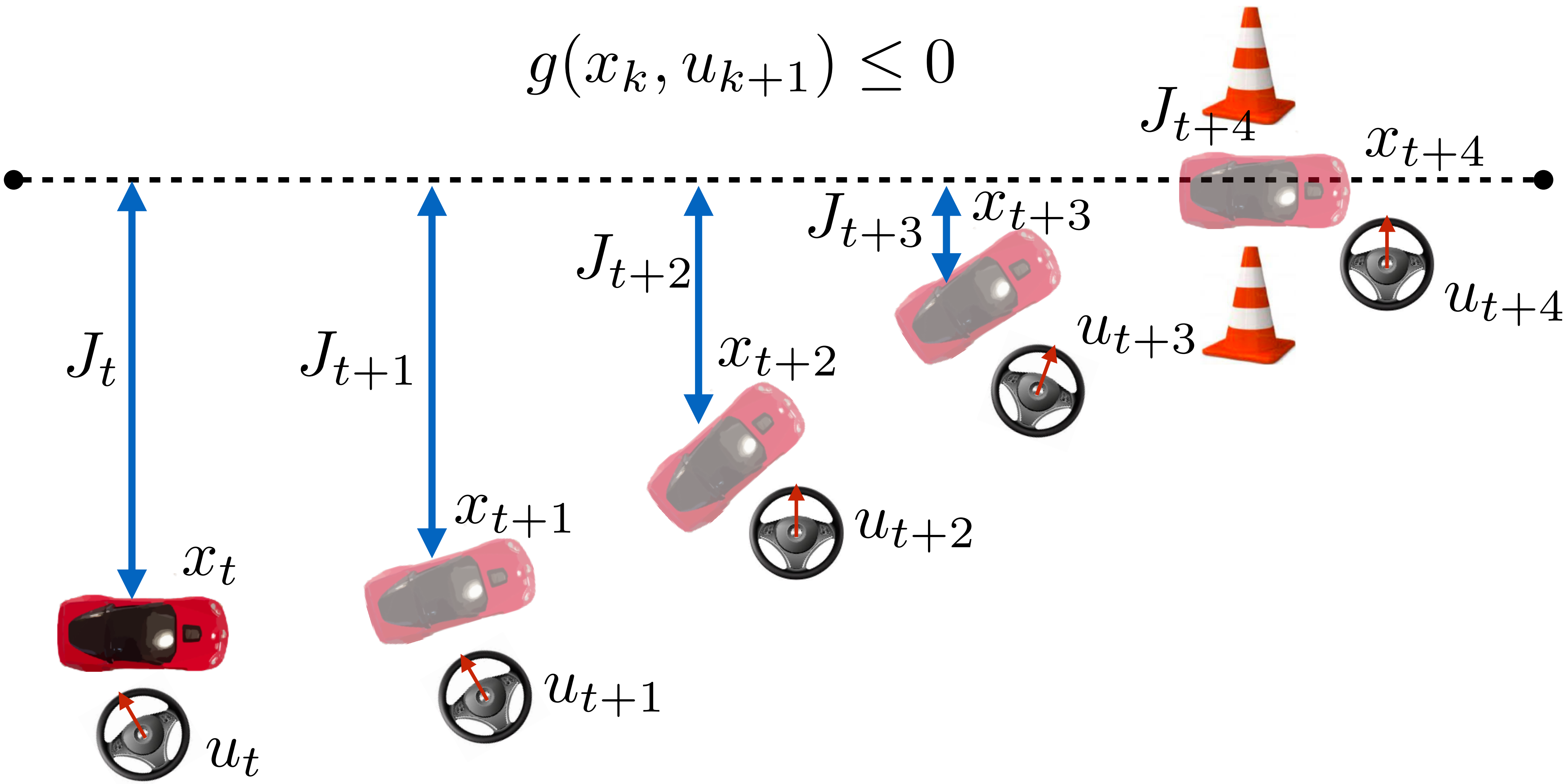
(Constraints)

Model predictive control (MPC)

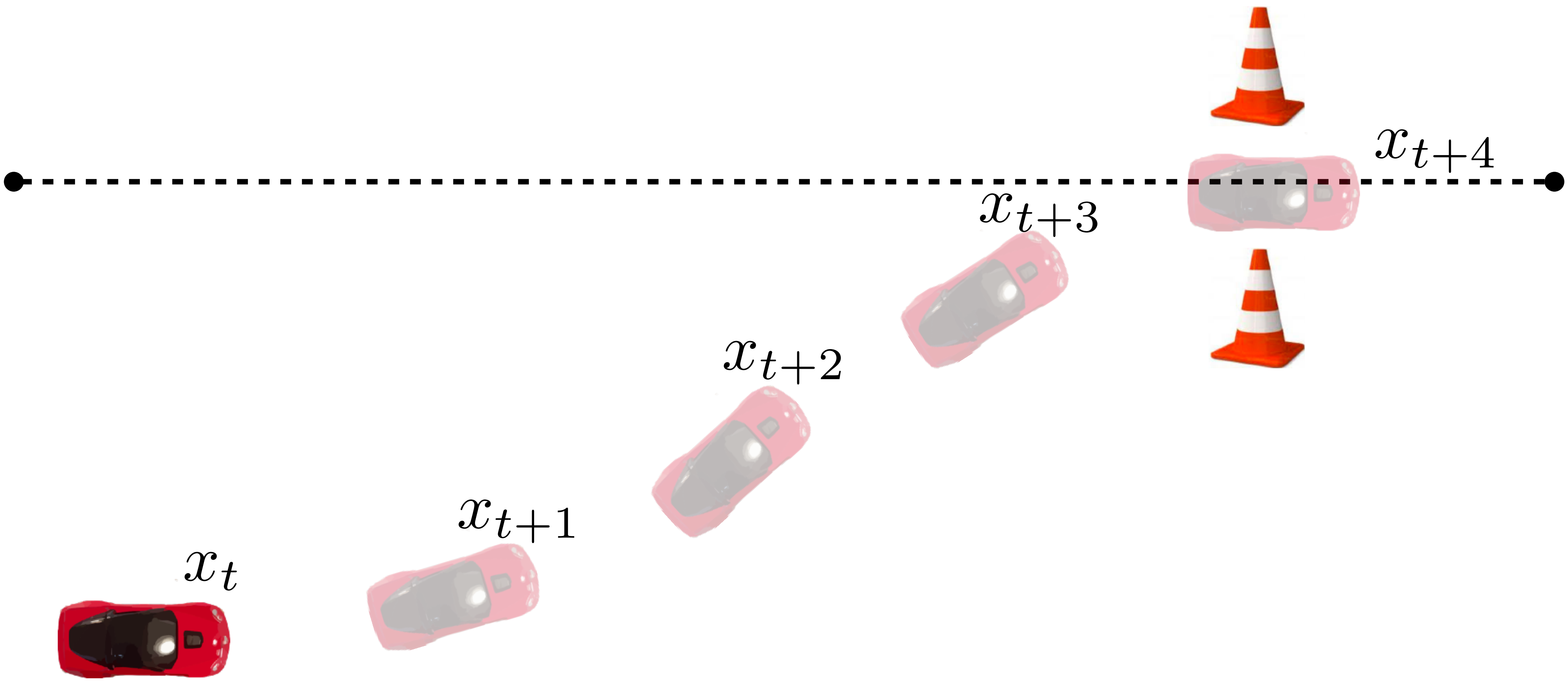
$$\min_{u_{t+1},\ldots u_{t+H}} \sum_{k=t}^{t+H-1} J(x_k, u_{k+1})$$

$$x_{k+1} = f(x_k, u_{k+1})$$

$$g(x_k, u_{k+1}) \leq 0$$

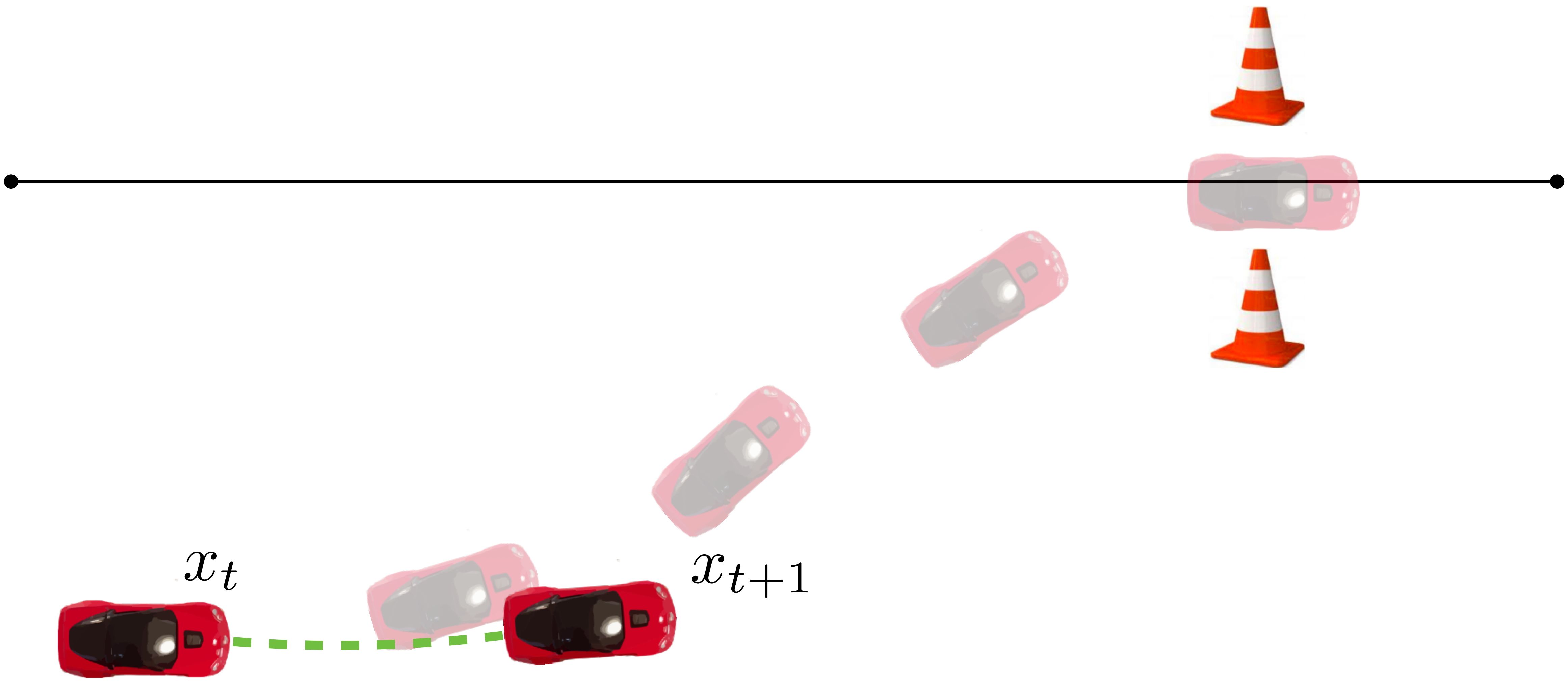


How are the controls executed?



Step 1: Solve optimization problem to a horizon

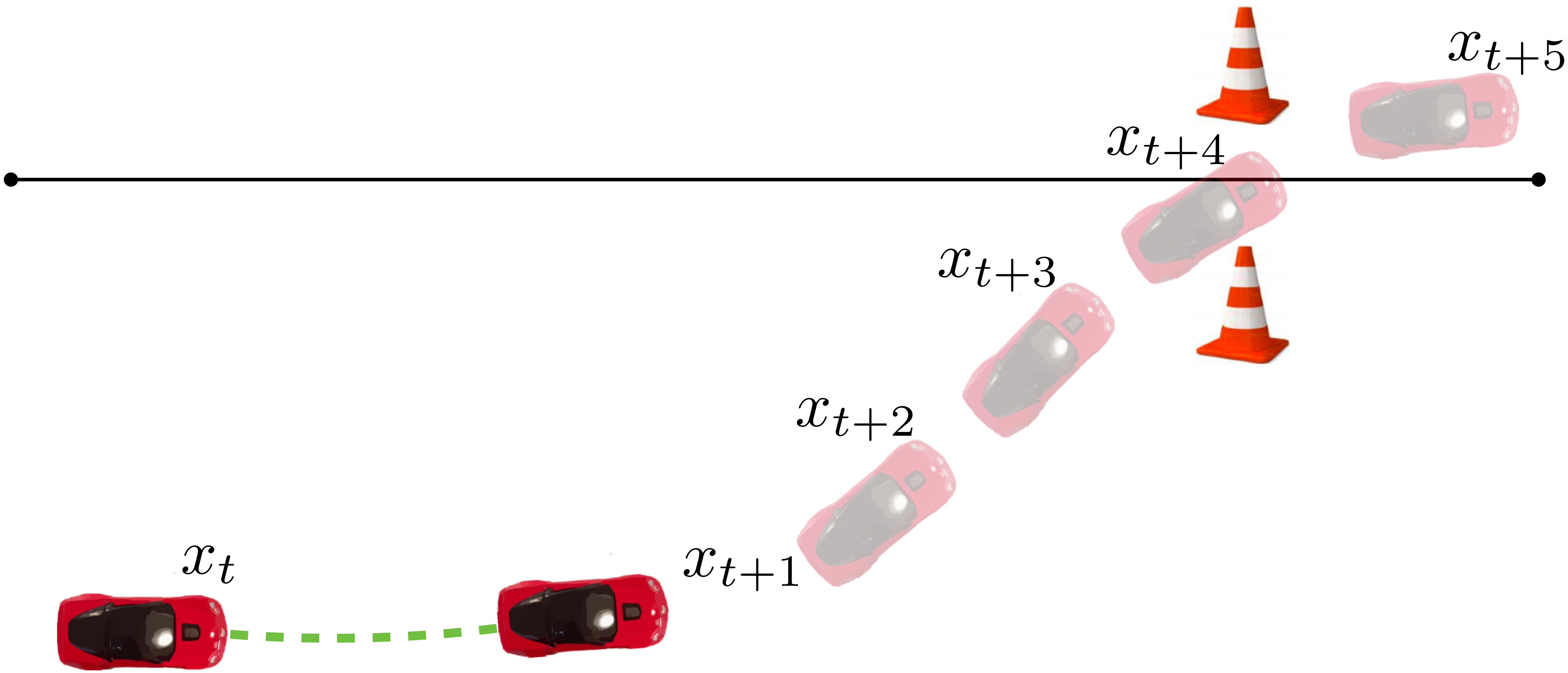
How are the controls executed?



Step 1: Solve optimization problem to a horizon

Step 2: Execute the first control

How are the controls executed?

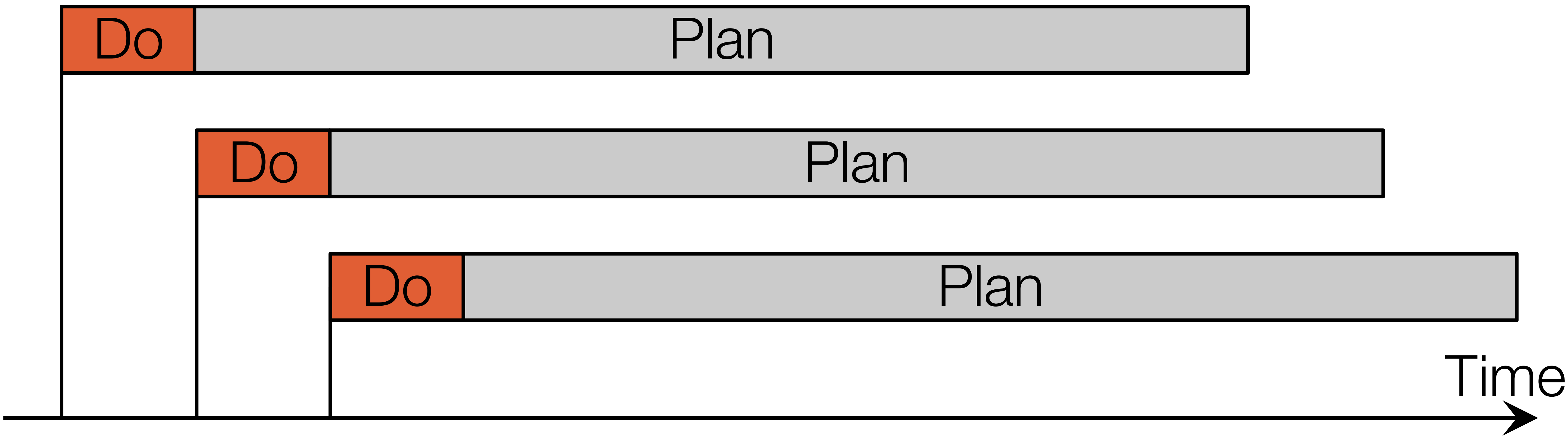


Step 1: Solve optimization problem to a horizon

Step 2: Execute the first control

Step 3: Repeat!

MPC is a framework

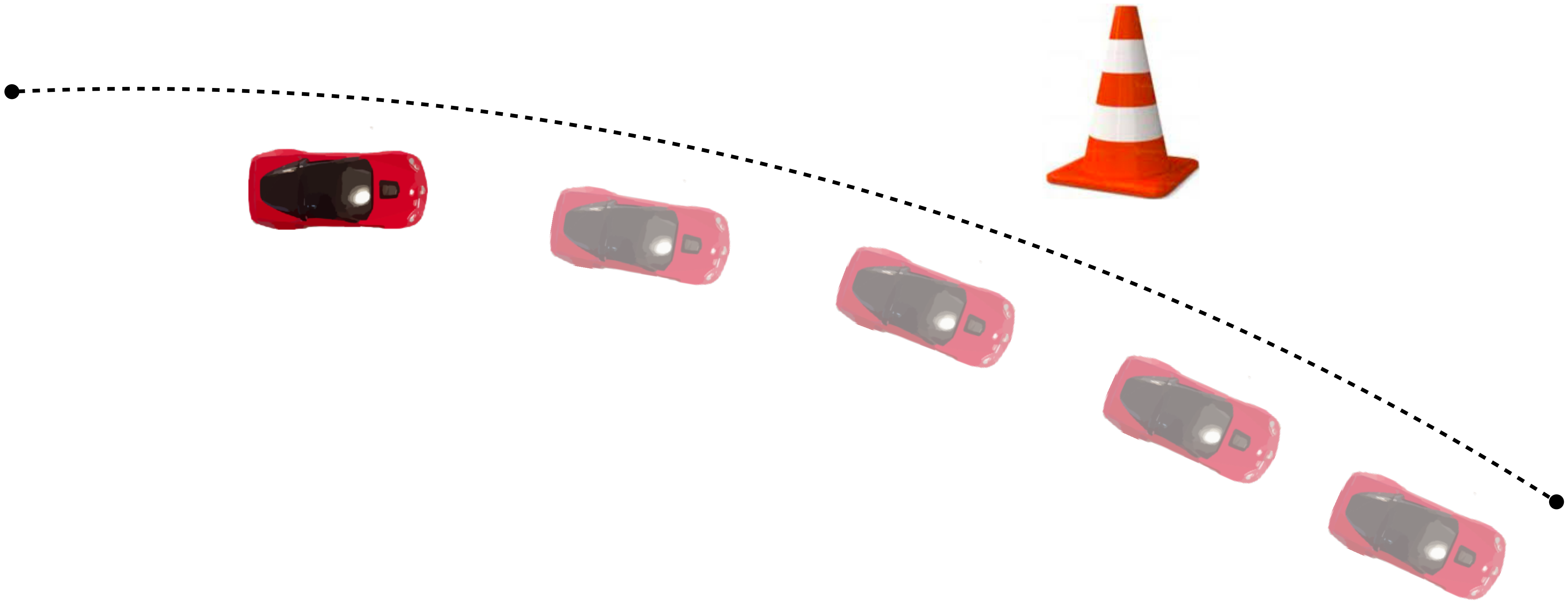


Step 1: Solve optimization problem to a horizon

Step 2: Execute the first control

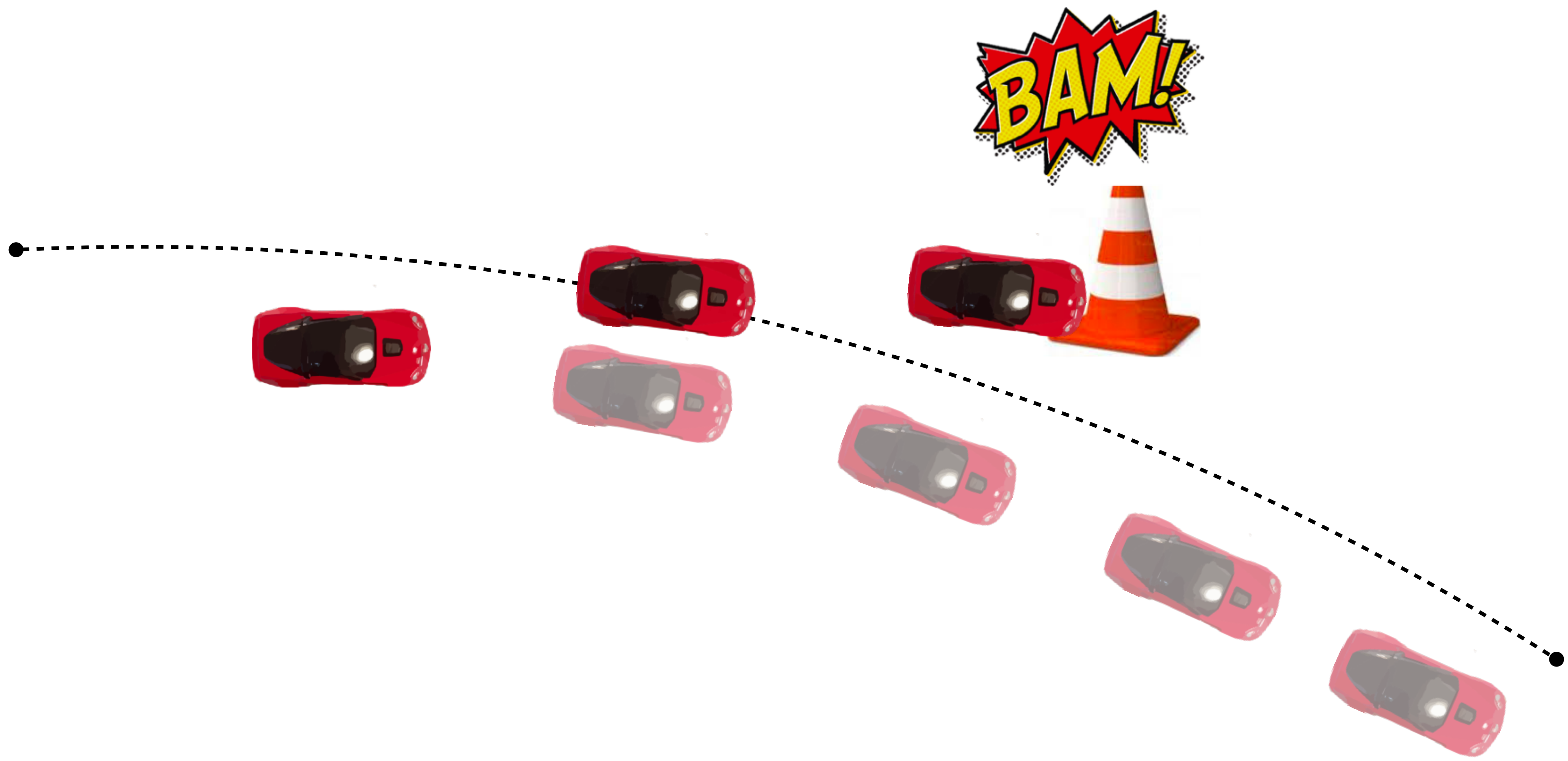
Step 3: Repeat!

Why do we need to replan?



What happens if the controls are planned once and executed?

Why do we need to replan?



What happens if the controls are planned once and executed?

Model predictive control (MPC)

$$\min_{u_{t+1},\ldots u_{t+H}} \sum_{k=t}^{t+H-1} J(x_k, u_{k+1})$$

(plan till horizon H) (Cost)

$$x_{k+1} = f(x_k, u_{k+1})$$

(Predict next state with dynamics)

$$g(x_k, u_{k+1}) \leq 0$$

(Constraints)