CS 8803

Deep Reinforcement Learning

Lec 9: Optimal Control & Planning
Fall 2024

Animesh Garg

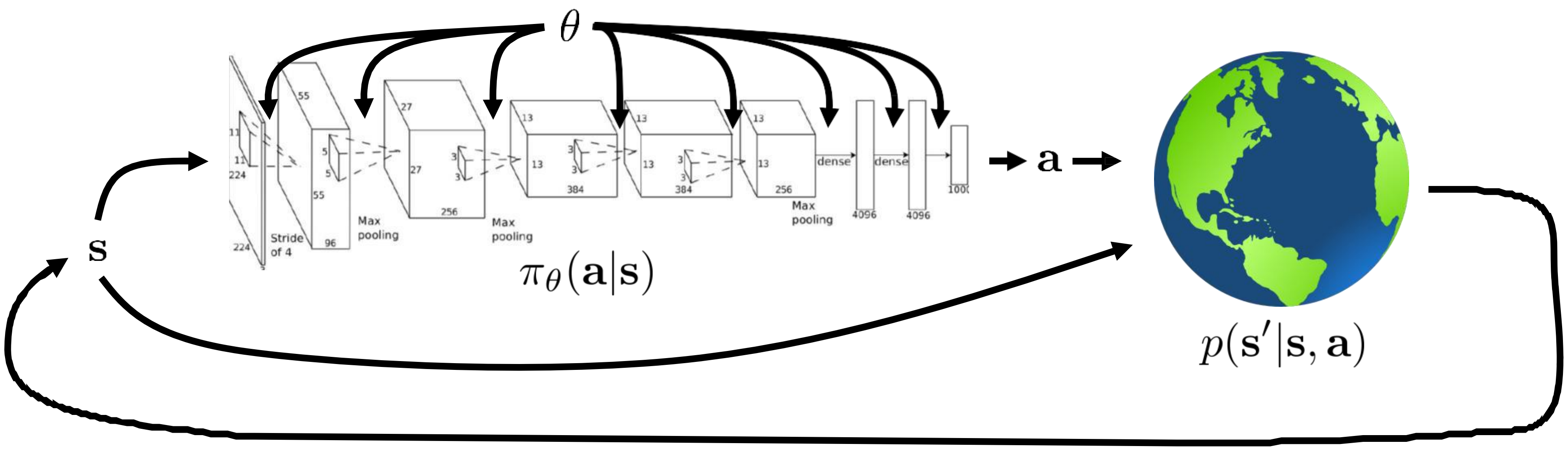
Slides from Sergey Levine

Today's Lecture

1. Introduction to model-based reinforcement learning
2. What if we know the dynamics? How can we make decisions?
3. Stochastic optimization methods
4. Monte Carlo tree search (MCTS)
5. Trajectory optimization

- Goals:
  - Understand how we can perform planning with known dynamics models in discrete and continuous spaces

Recap: the reinforcement learning objective

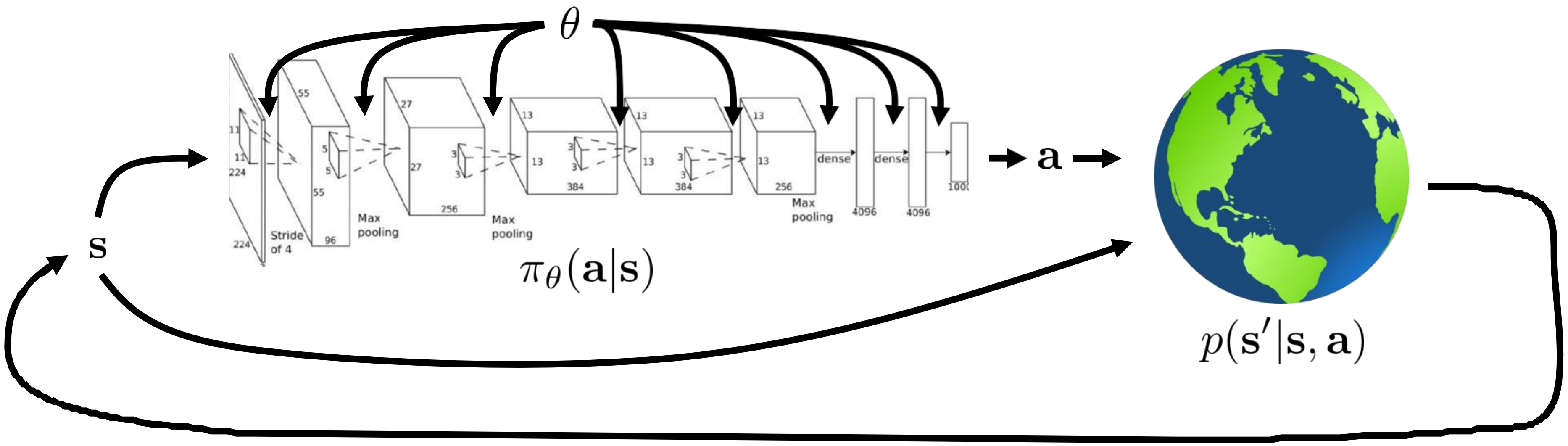


θ
s
a
$\pi_\theta(\mathbf{a}|\mathbf{s})$
$p(\mathbf{s}'|\mathbf{s},\mathbf{a})$
224
11
Stride of 4
55
96
5
27
Max pooling
256
3
13
384
dense
4096
1000

$$\underbrace{p_\theta(\mathbf{s}_1,\mathbf{a}_1,\ldots,\mathbf{s}_T,\mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1)\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$$

$$\theta^\star = \arg\max_\theta E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t,\mathbf{a}_t)\right]$$

Recap: model-free reinforcement learning



$$\underbrace{p_\theta(\mathbf{s}_1,\mathbf{a}_1,\ldots,\mathbf{s}_T,\mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1)\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\xcancel{p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}$$

assume this is unknown

don’t even attempt to learn it

$$\theta^\star = \arg\max_\theta E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t,\mathbf{a}_t)\right]$$

What if we knew the transition dynamics?

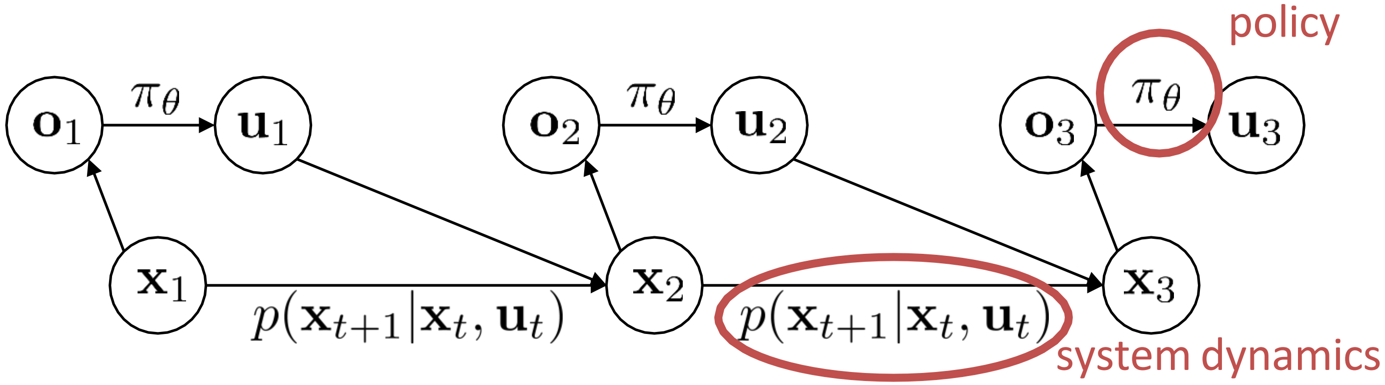
- Often we do know the dynamics
  1. Games (e.g., Atari games, chess, Go)
  2. Easily modeled systems (e.g., navigating a car)
  3. Simulated environments (e.g., simulated robots, video games)
- Often we can learn the dynamics
  1. System identification – fit unknown parameters of a known model
  2. Learning – fit a general-purpose model to observed transition data

Does knowing the dynamics make things easier?

Often, yes!

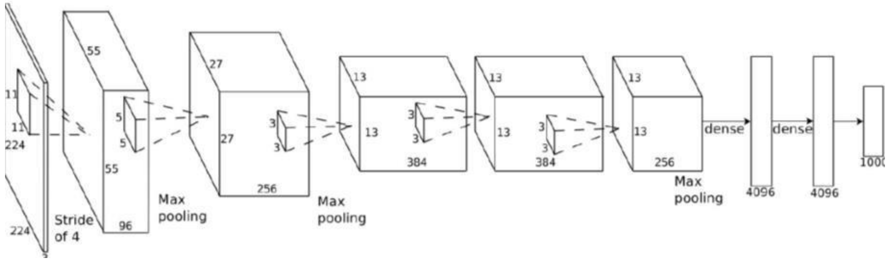
Model-based reinforcement learning

1. Model-based reinforcement learning: learn the transition dynamics, then figure out how to choose actions
2. Today: how can we make decisions if we *know* the dynamics?
   a. How can we choose actions under perfect knowledge of the system dynamics?
   b. Optimal control, trajectory optimization, planning
3. Next week: how can we learn *unknown* dynamics?
4. How can we then also learn policies? (*e.g. by imitating optimal control*)

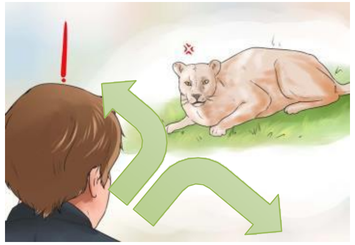


The objective

$\mathbf{s}_t$



$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$

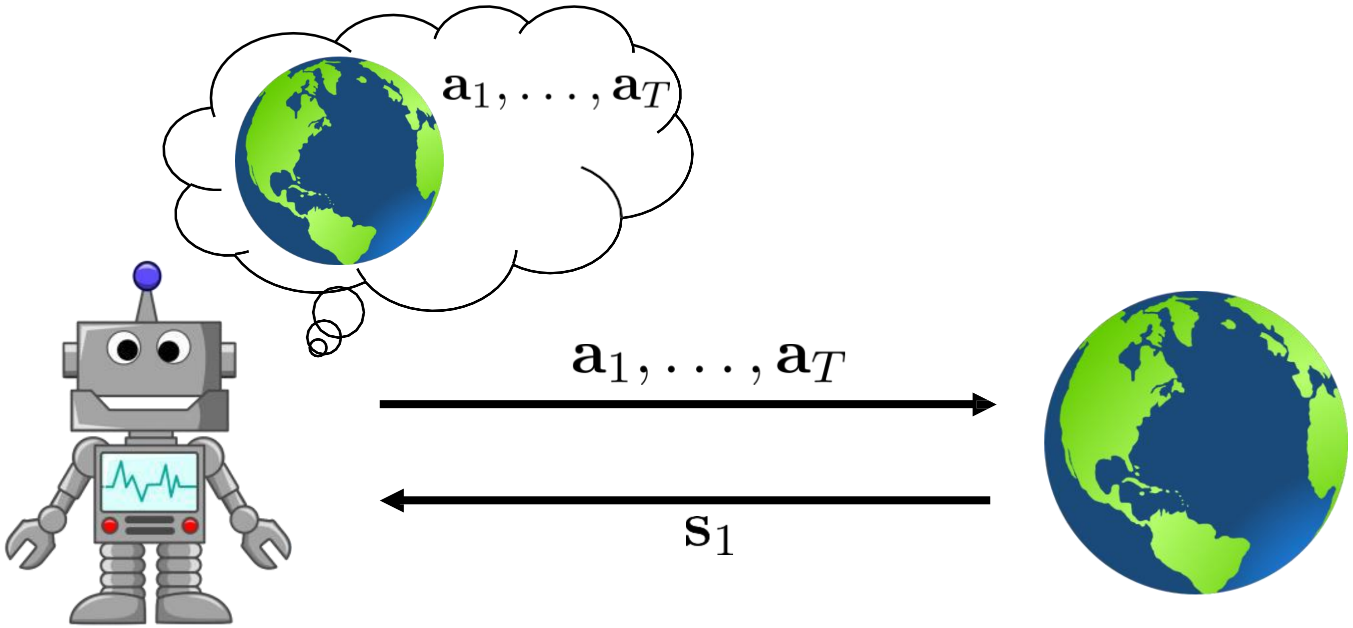


$\mathbf{a}_t$

$$\min_{\mathbf{a}_1,\ldots,\mathbf{a}_T} \log p(\text{eaten by tiger}|\mathbf{a}_1,\ldots,\mathbf{a}_T)$$

$$\min_{\mathbf{a}_1,\ldots,\mathbf{a}_T} \sum_{t=1}^{T} c(\mathbf{s}_t,\mathbf{a}_t) \text{ s.t. } \mathbf{s}_t = f(\mathbf{s}_{t-1},\mathbf{a}_{t-1})$$

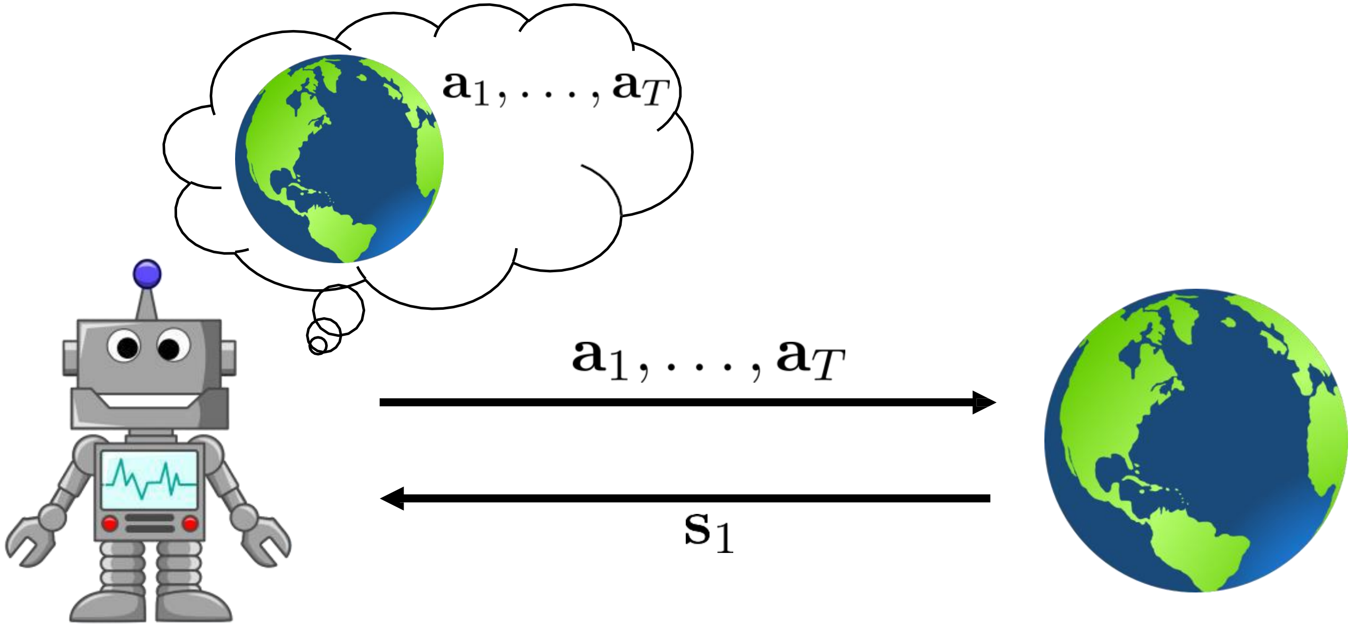
The deterministic case



$\mathbf{a}_1, \ldots, \mathbf{a}_T$
$\mathbf{a}_1, \ldots, \mathbf{a}_T$
$\mathbf{s}_1$

$$\mathbf{a}_1, \ldots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \ldots, \mathbf{a}_T} \sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \text{ s.t. } \mathbf{a}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$$

The stochastic open-loop case



$\mathbf{a}_1, \ldots, \mathbf{a}_T$
$\mathbf{a}_1, \ldots, \mathbf{a}_T$
$\mathbf{s}_1$

$$p_\theta(\mathbf{s}_1, \ldots, \mathbf{s}_T | \mathbf{a}_1, \ldots, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^{T} p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$$

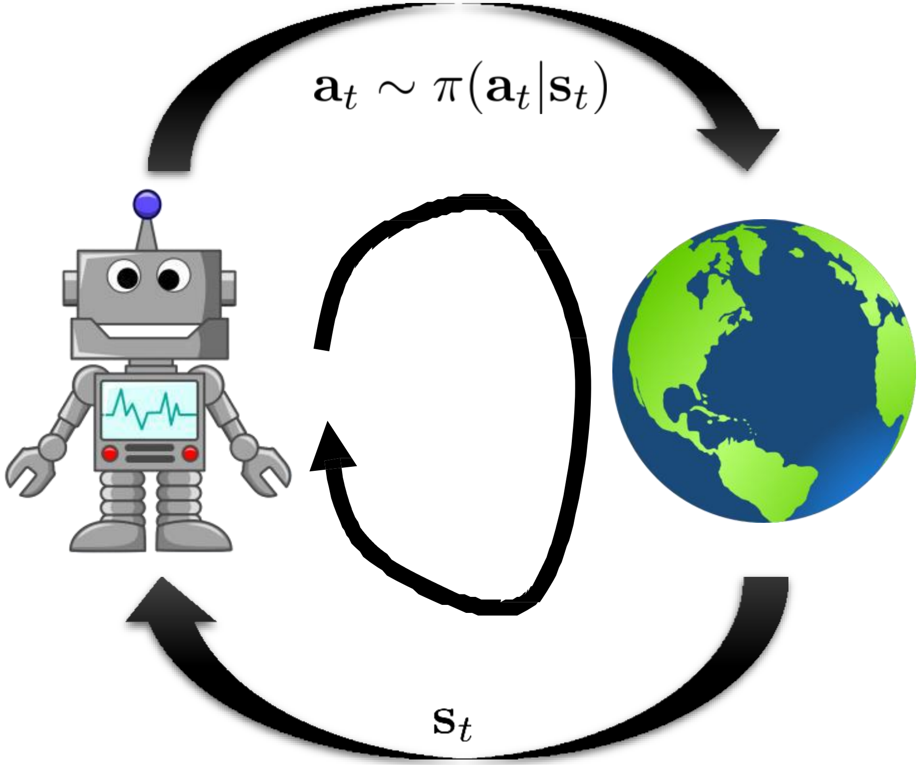
$$\mathbf{a}_1, \ldots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \ldots, \mathbf{a}_T} E\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) | \mathbf{a}_1, \ldots, \mathbf{a}_T\right]$$

why is this suboptimal?

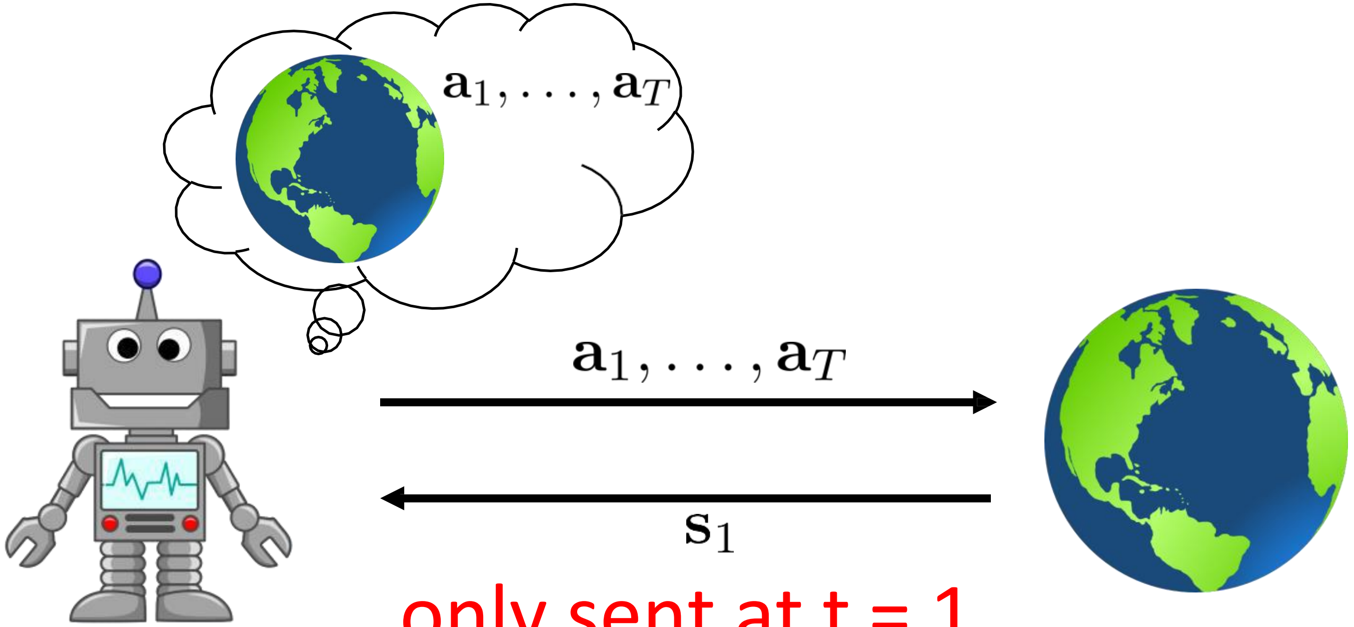
Aside: terminology

what is this “loop”?

closed-loop

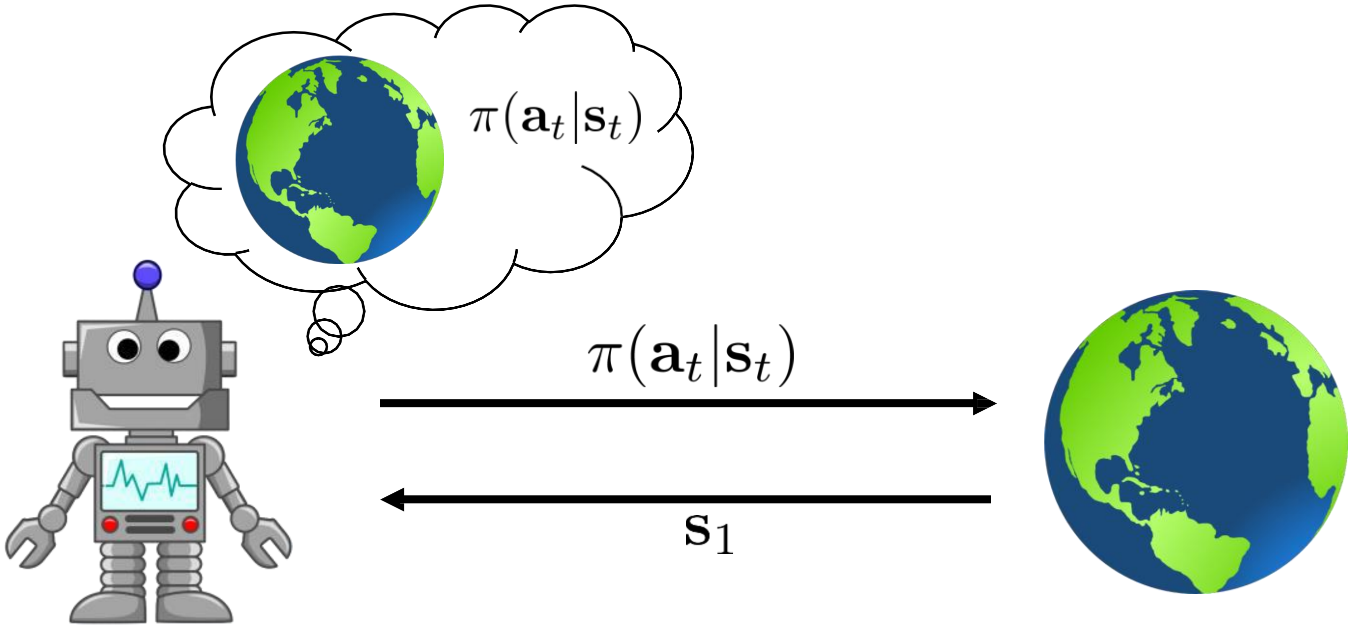


open-loop


$\mathbf{a}_1, \ldots, \mathbf{a}_T$
$\mathbf{a}_1, \ldots, \mathbf{a}_T$
$\mathbf{s}_1$

only sent at t = 1,
then it’s one-way!

The stochastic closed-loop case


$\pi(\mathbf{a}_t|\mathbf{s}_t)$
$\pi(\mathbf{a}_t|\mathbf{s}_t)$
$\mathbf{s}_1$

$$p(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

$$\pi = \arg\max_{\pi} E_{\tau \sim p(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

form of π?

neural net global

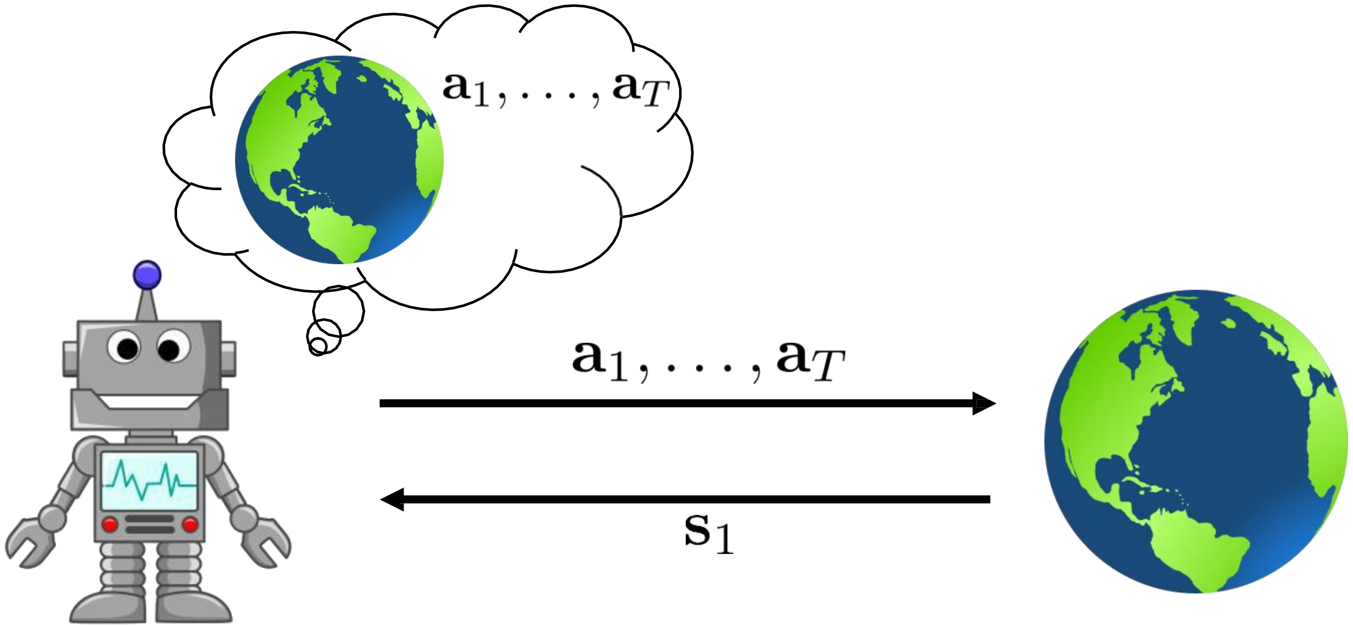
time-varying linear

$\mathbf{K}_t \mathbf{s}_t + \mathbf{k}_t$ local

(more on this later)

Open-Loop Planning

But for now, open-loop planning


$\mathbf{a}_1, \ldots, \mathbf{a}_T$
$\mathbf{a}_1, \ldots, \mathbf{a}_T$
$\mathbf{s}_1$

$$\mathbf{a}_1, \ldots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \ldots, \mathbf{a}_T} \sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \text{ s.t. } \mathbf{a}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$$

Stochastic optimization

abstract away optimal control/planning:

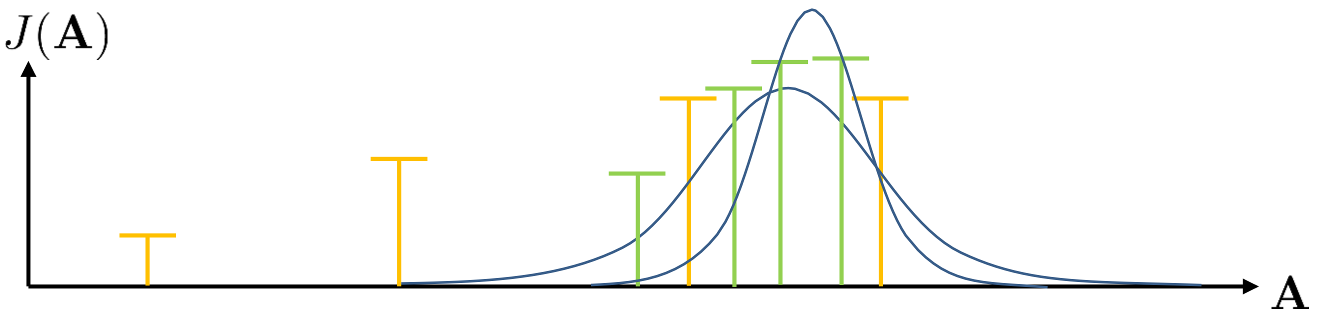
$$\mathbf{a}_1, \ldots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \ldots, \mathbf{a}_T} \underbrace{J(\mathbf{a}_1, \ldots, \mathbf{a}_T)}_{\text{don't care what this is}} \qquad \mathbf{A} = \arg\max_{\mathbf{A}} J(\mathbf{A})$$

simplest method: guess & check "random shooting method"

1. pick $\mathbf{A}_1, \ldots, \mathbf{A}_N$ from some distribution (e.g., uniform)
2. choose $\mathbf{A}_i$ based on $\arg\max_i J(\mathbf{A}_i)$

Cross-entropy method (CEM)

1. pick $\mathbf{A}_1, \dots, \mathbf{A}_N$ from some distribution (e.g., uniform)
2. choose $\mathbf{A}_i$ based on $\arg\max_i J(\mathbf{A}_i)$



can we do better?

typically use Gaussian distribution

see also: CMA-ES (sort of like CEM with momentum)

cross-entropy method with continuous-valued inputs:

1. sample $\mathbf{A}_1, \dots, \mathbf{A}_N$ from $p(\mathbf{A})$
2. evaluate $J(\mathbf{A}_1), \dots, J(\mathbf{A}_N)$
3. pick the *elites* $\mathbf{A}_{i_1}, \dots, \mathbf{A}_{i_M}$ with the highest value, where $M < N$
4. refit $p(\mathbf{A})$ to the elites $\mathbf{A}_{i_1}, \dots, \mathbf{A}_{i_M}$

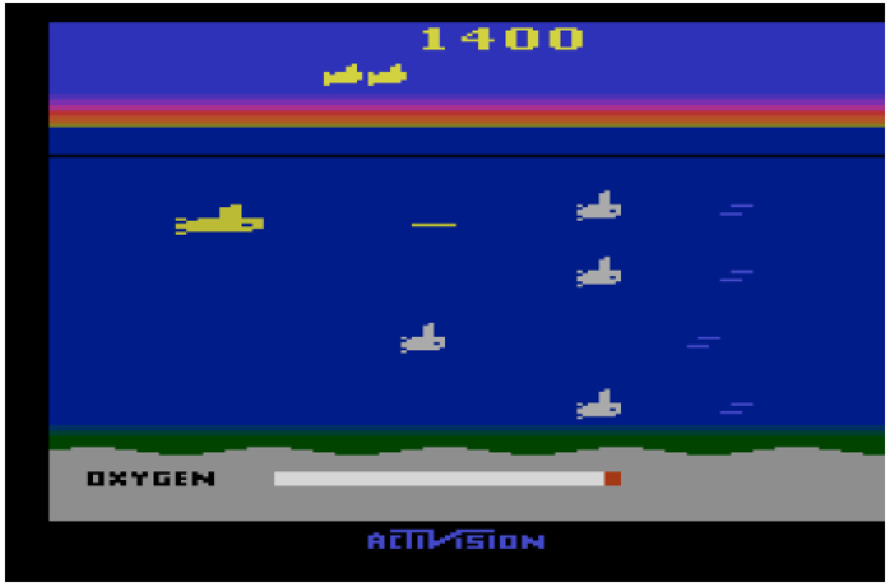
What’s the upside?

1. Very fast if parallelized
2. Extremely simple

What’s the problem?

1. Very harsh dimensionality limit
2. Only open-loop planning

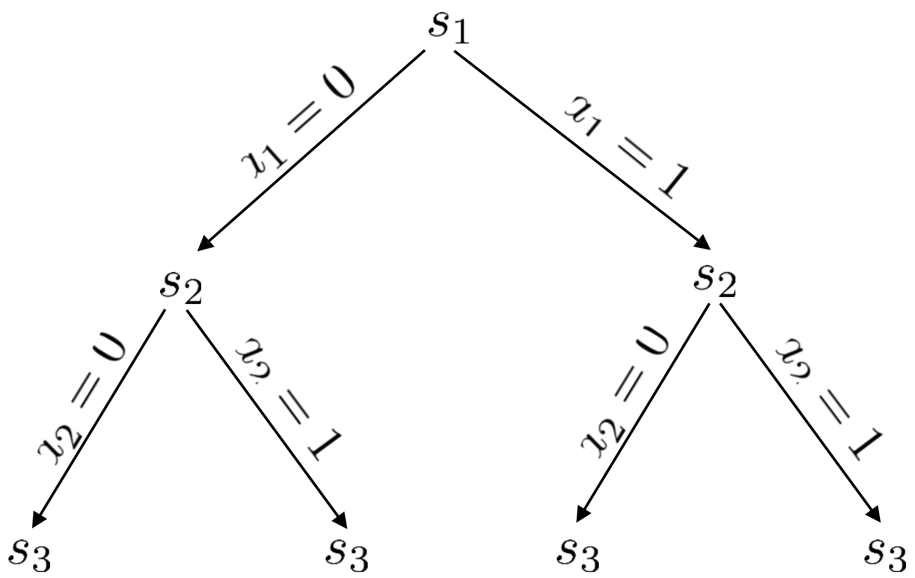
Discrete case: Monte Carlo tree search (MCTS)



s_t

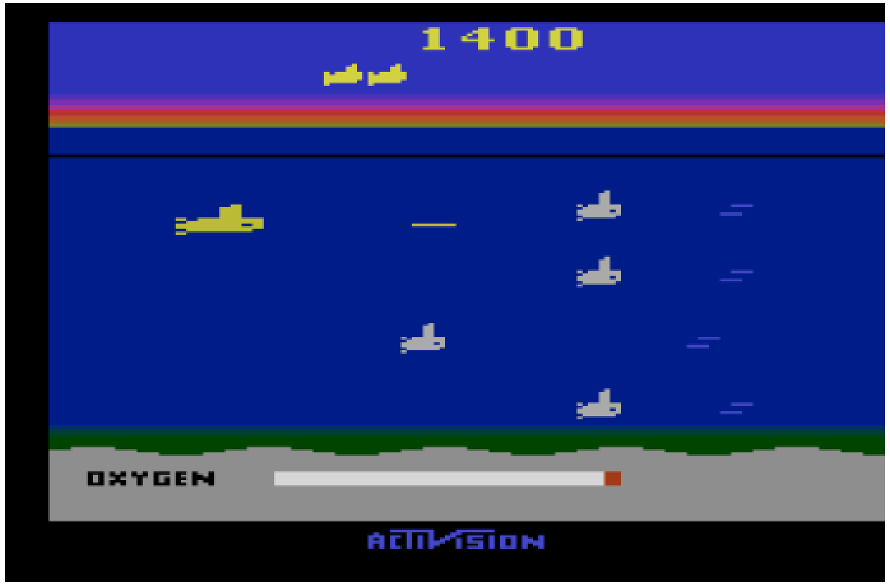
a_t

discrete planning as a search problem



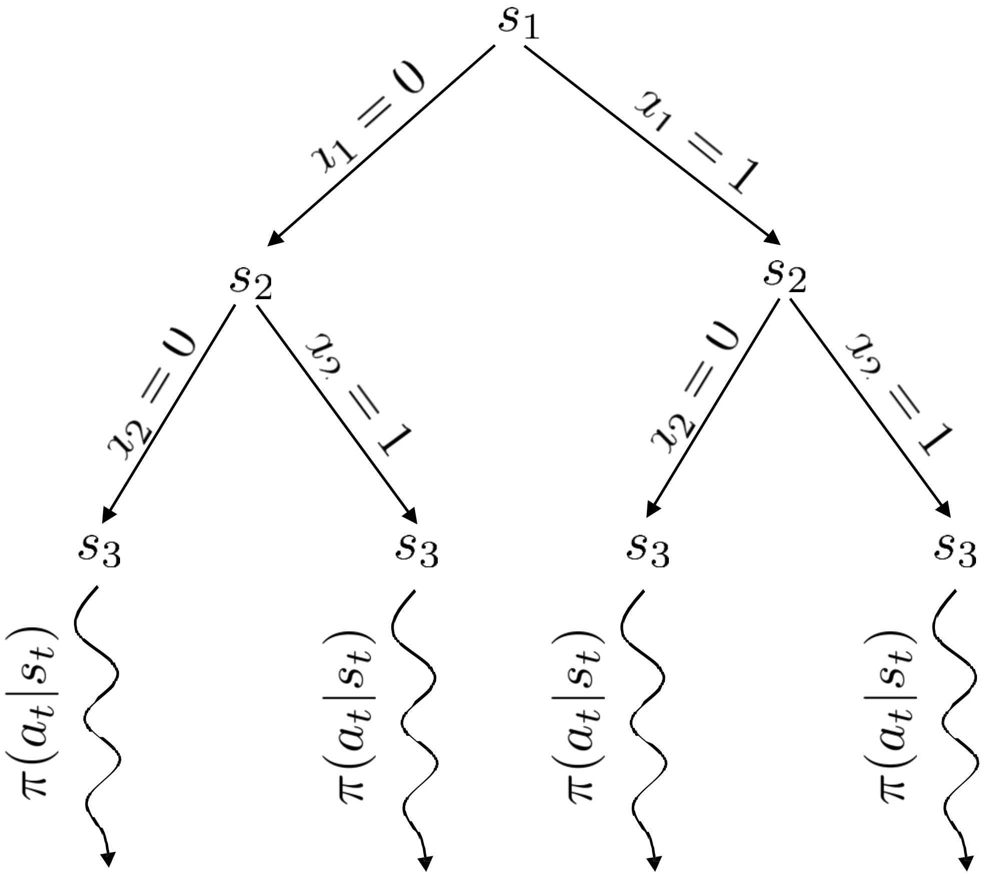
Discrete case: Monte Carlo tree search (MCTS)

how to approximate value without full tree?



s_t

a_t



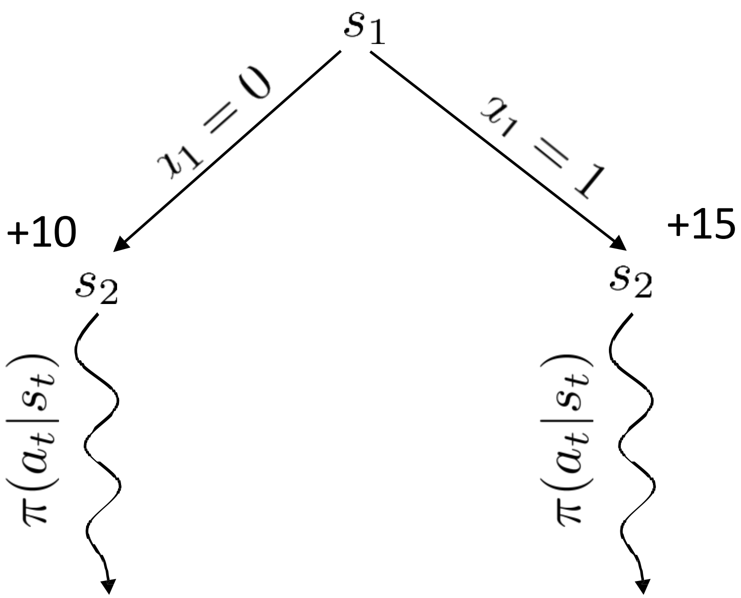
e.g., random policy

Discrete case: Monte Carlo tree search (MCTS)

can't search all paths – where to search first?

s_t

a_t

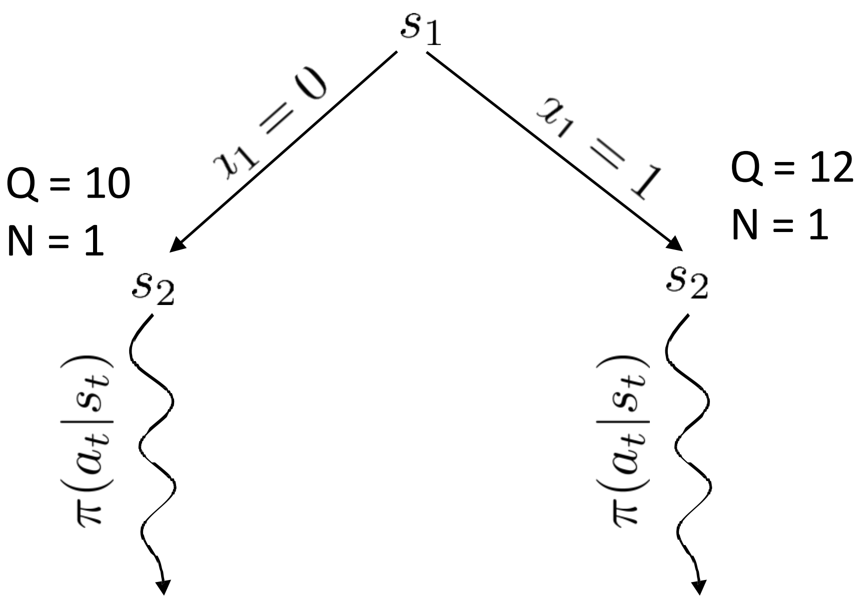


intuition: choose nodes with best reward, but also prefer rarely visited nodes

Discrete case: Monte Carlo tree search (MCTS)

```
generic MCTS sketch
    1. find a leaf $s_l$ using TreePolicy($s_1$)
    2. evaluate the leaf using DefaultPolicy($s_l$)
    3. update all values in tree between $s_1$ and $s_l$
take best action from $s_1$

UCT TreePolicy($s_t$)
    if $s_t$ not fully expanded, choose new $a_t$
    else choose child with best Score($s_{t+1}$)
```



Discrete case: Monte Carlo tree search (MCTS)

generic MCTS sketch

1. find a leaf s_l using TreePolicy(s_1)
2. evaluate the leaf using DefaultPolicy(s_l)
3. update all values in tree between s_1 and s_l

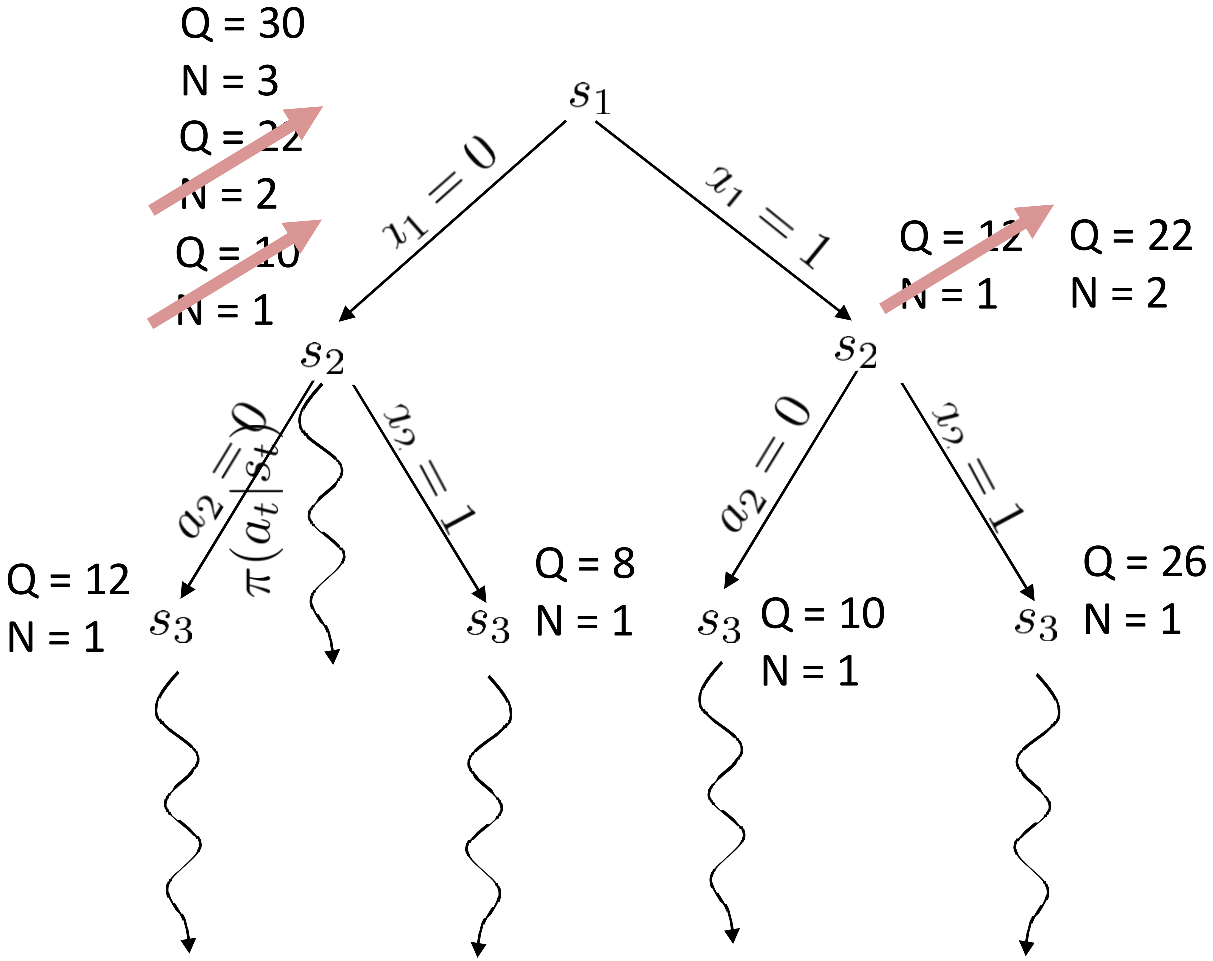
take best action from s_1

UCT TreePolicy(s_t)

if s_t not fully expanded, choose new a_t

else choose child with best Score(s_{t+1})

$$\text{Score}(s_t) = \frac{Q(s_t)}{N(s_t)} + 2C\sqrt{\frac{2\ln N(s_{t-1})}{N(s_t)}}$$



Additional reading

1. Browne, Powley, Whitehouse, Lucas, Cowling, Rohlfshagen, Tavener, Perez, Samothrakis, Colton. (2012). A Survey of Monte Carlo Tree Search Methods.
    - Survey of MCTS methods and basic summary.

Trajectory Optimization with Derivatives

Can we use derivatives?

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \text{ s.t. } \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \cdots + c(f(f(\ldots)\ldots), \mathbf{u}_T)$$

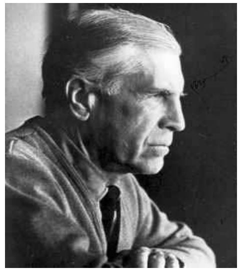
usual story: differentiate via backpropagation and optimize!

need $\frac{df}{d\mathbf{x}_t}, \frac{df}{d\mathbf{u}_t}, \frac{dc}{d\mathbf{x}_t}, \frac{dc}{d\mathbf{u}_t}$

in practice, it really helps to use a 2$^{\text{nd}}$ order method!

$\mathbf{s}_t$ – state
$\mathbf{a}_t$ – action

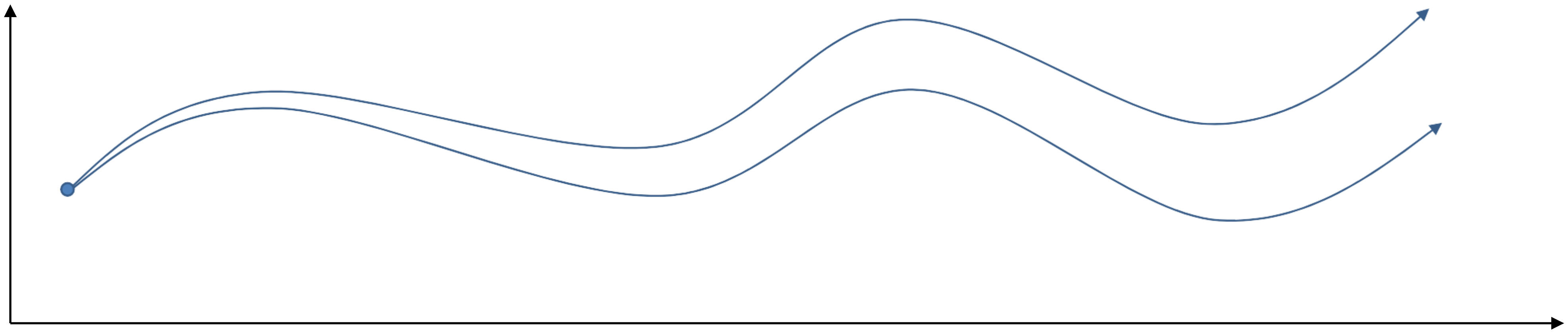
$\mathbf{x}_t$ – state
$\mathbf{u}_t$ – action



Shooting methods vs collocation

shooting method: optimize over actions only

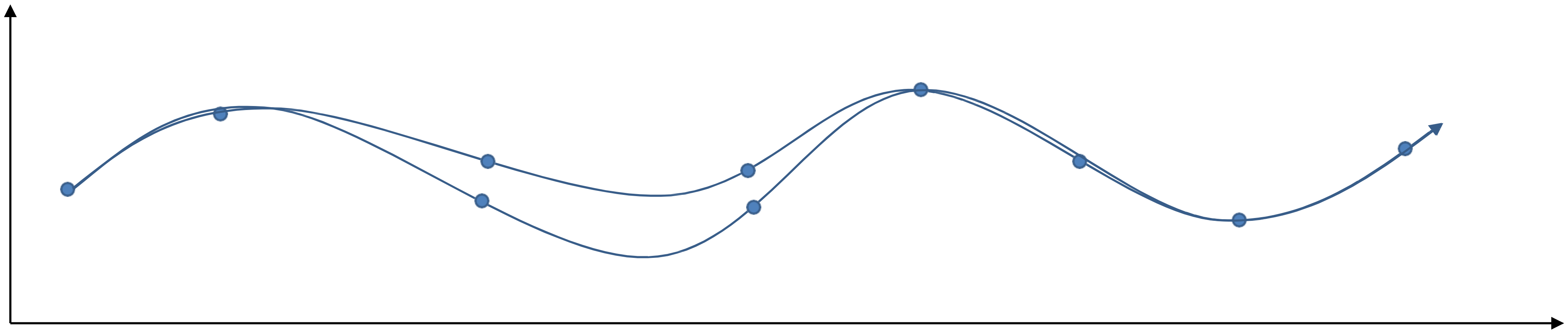
$$\min_{\mathbf{u}_1,\dots,\mathbf{u}_T} c(\mathbf{x}_1,\mathbf{u}_1) + c(f(\mathbf{x}_1,\mathbf{u}_1),\mathbf{u}_2) + \cdots + c(f(f(\dots)\dots),\mathbf{u}_T)$$



Shooting methods vs collocation

collocation method: optimize over actions and states, with constraints

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T,\mathbf{x}_1,\ldots,\mathbf{x}_T} \sum_{t=1}^{T} c(\mathbf{x}_t,\mathbf{u}_t) \text{ s.t. } \mathbf{x}_t = f(\mathbf{x}_{t-1},\mathbf{u}_{t-1})$$



Linear case: LQR

$$\min_{\mathbf{u}_1,\dots,\mathbf{u}_T} c(\mathbf{x}_1,\mathbf{u}_1) + c(f(\mathbf{x}_1,\mathbf{u}_1),\mathbf{u}_2) + \cdots + c(f(f(\dots)\dots),\mathbf{u}_T)$$

$$\underbrace{f(\mathbf{x}_t,\mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t}_{\text{linear}}$$

$$\underbrace{c(\mathbf{x}_t,\mathbf{u}_t) = \frac{1}{2}\begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{C}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{c}_t}_{\text{quadratic}}$$

Linear case: LQR

$$\min_{\mathbf{u}_1,\dots,\mathbf{u}_T} c(\mathbf{x}_1,\mathbf{u}_1) + c(f(\mathbf{x}_1,\mathbf{u}_1),\mathbf{u}_2) + \cdots + \underbrace{c(\overbrace{f(f(\dots)\dots)}^{\mathbf{x}_T \text{ (unknown)}},\mathbf{u}_T)}_{\text{only term that depends on } \mathbf{u}_T}$$

$$c(\mathbf{x}_t,\mathbf{u}_t) = \frac{1}{2}\begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix}^T \mathbf{C}_t \begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix} + \begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix}^T \mathbf{c}_t$$

$$f(\mathbf{x}_t,\mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix} + \mathbf{f}_t$$

$$\mathbf{C}_T = \begin{bmatrix}\mathbf{C}_{\mathbf{x}_T,\mathbf{x}_T} & \mathbf{C}_{\mathbf{x}_T,\mathbf{u}_T}\\ \mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T} & \mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}\end{bmatrix}$$

Base case: solve for $\mathbf{u}_T$ *only*

$$\mathbf{c}_T = \begin{bmatrix}\mathbf{c}_{\mathbf{x}_T}\\ \mathbf{c}_{\mathbf{u}_T}\end{bmatrix}$$

$$Q(\mathbf{x}_T,\mathbf{u}_T) = \text{const} + \frac{1}{2}\begin{bmatrix}\mathbf{x}_T\\ \mathbf{u}_T\end{bmatrix}^T \mathbf{C}_T \begin{bmatrix}\mathbf{x}_T\\ \mathbf{u}_T\end{bmatrix} + \begin{bmatrix}\mathbf{x}_T\\ \mathbf{u}_T\end{bmatrix}^T \mathbf{c}_T$$

$$\nabla_{\mathbf{u}_T} Q(\mathbf{x}_T,\mathbf{u}_T) = \mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T}\mathbf{x}_T + \mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}\mathbf{u}_T + \mathbf{c}_{\mathbf{u}_T}^T = 0$$

$$\mathbf{K}_T = -\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}^{-1}\mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T}$$

$$\mathbf{u}_T = -\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}^{-1}\left(\mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T}\mathbf{x}_T + \mathbf{c}_{\mathbf{u}_T}\right) \qquad \mathbf{u}_T = \mathbf{K}_T\mathbf{x}_T + \mathbf{k}_T \qquad \mathbf{k}_T = -\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}^{-1}\mathbf{c}_{\mathbf{u}_T}$$

Linear case: LQR

$$\mathbf{u}_T = \mathbf{K}_T\mathbf{x}_T + \mathbf{k}_T \qquad \mathbf{K}_T = -\mathbf{C}^{-1}_{\mathbf{u}_T,\mathbf{u}_T}\mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T} \qquad \mathbf{k}_T = -\mathbf{C}^{-1}_{\mathbf{u}_T,\mathbf{u}_T}\mathbf{c}_{\mathbf{u}_T}$$

$$Q(\mathbf{x}_T,\mathbf{u}_T) = \text{const} + \frac{1}{2}\begin{bmatrix}\mathbf{x}_T\\ \mathbf{u}_T\end{bmatrix}^T \mathbf{C}_T \begin{bmatrix}\mathbf{x}_T\\ \mathbf{u}_T\end{bmatrix} + \begin{bmatrix}\mathbf{x}_T\\ \mathbf{u}_T\end{bmatrix}^T \mathbf{c}_T$$

Since $\mathbf{u}_T$ is fully determined by $\mathbf{x}_T$, we can eliminate it via substitution!

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2}\begin{bmatrix}\mathbf{x}_T\\ \mathbf{K}_T\mathbf{x}_T + \mathbf{k}_T\end{bmatrix}^T \mathbf{C}_T \begin{bmatrix}\mathbf{x}_T\\ \mathbf{K}_T\mathbf{x}_T + \mathbf{k}_T\end{bmatrix} + \begin{bmatrix}\mathbf{x}_T\\ \mathbf{K}_T\mathbf{x}_T + \mathbf{k}_T\end{bmatrix}^T \mathbf{c}_T$$

$$\begin{aligned} V(\mathbf{x}_T) = &\frac{1}{2}\mathbf{x}_T^T\mathbf{C}_{\mathbf{x}_T,\mathbf{x}_T}\mathbf{x}_T + \frac{1}{2}\mathbf{x}_T^T\mathbf{C}_{\mathbf{x}_T,\mathbf{u}_T}\mathbf{K}_T\mathbf{x}_T + \frac{1}{2}\mathbf{x}_T^T\mathbf{K}_T^T\mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T}\mathbf{x}_T + \frac{1}{2}\mathbf{x}_T^T\mathbf{K}_T^T\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}\mathbf{K}_T\mathbf{x}_T + \\ &\mathbf{x}_T^T\mathbf{K}_T^T\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}\mathbf{k}_T + \frac{1}{2}\mathbf{x}_T^T\mathbf{C}_{\mathbf{x}_T,\mathbf{u}_T}\mathbf{k}_T + \mathbf{x}_T^T\mathbf{c}_{\mathbf{x}_T} + \mathbf{x}_T^T\mathbf{K}_T^T\mathbf{c}_{\mathbf{u}_T} + \text{const}\end{aligned}$$

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2}\mathbf{x}_T^T\mathbf{V}_T\mathbf{x}_T + \mathbf{x}_T^T\mathbf{v}_T$$

$$\mathbf{V}_T = \mathbf{C}_{\mathbf{x}_T,\mathbf{x}_T} + \mathbf{C}_{\mathbf{x}_T,\mathbf{u}_T}\mathbf{K}_T + \mathbf{K}_T^T\mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T} + \mathbf{K}_T^T\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}\mathbf{K}_T$$

$$\mathbf{v}_T = \mathbf{c}_{\mathbf{x}_T} + \mathbf{C}_{\mathbf{x}_T,\mathbf{u}_T}\mathbf{k}_T + \mathbf{K}_T^T\mathbf{C}_{\mathbf{u}_T} + \mathbf{K}_T^T\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}\mathbf{k}_T$$

Linear case: LQR

Solve for $\mathbf{u}_{T-1}$ in terms of $\mathbf{x}_{T-1}$

$\mathbf{u}_{T-1}$ affects $\mathbf{x}_T$!

$$f(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \mathbf{x}_T = \mathbf{F}_{T-1} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \mathbf{f}_{T-1}$$

$$Q(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{C}_{T-1} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{c}_{T-1} + \underbrace{V(f(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}))}_{V(\mathbf{x}_T) = \text{const} + \frac{1}{2}\mathbf{x}_T^T \mathbf{V}_T \mathbf{x}_T + \mathbf{x}_T^T \mathbf{v}_T}$$

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{F}_{T-1}}_{\text{quadratic}} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{f}_{T-1}}_{\text{linear}} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T \mathbf{v}_T}_{\text{linear}}$$

Linear case: LQR

$$Q(\mathbf{x}_{T-1},\mathbf{u}_{T-1}) = \text{const}+\frac{1}{2}\begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix}^T \mathbf{C}_{T-1}\begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix}+\begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix}^T \mathbf{c}_{T-1}+V(f(\mathbf{x}_{T-1},\mathbf{u}_{T-1}))$$

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2}\begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T\mathbf{V}_T\mathbf{F}_{T-1}}_{\text{quadratic}}\begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix} + \begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T\mathbf{V}_T\mathbf{f}_{T-1}}_{\text{linear}} + \begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T\mathbf{v}_T}_{\text{linear}}$$

$$Q(\mathbf{x}_{T-1},\mathbf{u}_{T-1}) = \text{const} + \frac{1}{2}\begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix}^T \mathbf{Q}_{T-1}\begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix} + \begin{bmatrix}\mathbf{x}_{T-1}\\ \mathbf{u}_{T-1}\end{bmatrix}^T \mathbf{q}_{T-1}$$

$$\mathbf{Q}_{T-1} = \mathbf{C}_{T-1} + \mathbf{F}_{T-1}^T\mathbf{V}_T\mathbf{F}_{T-1}$$

$$\mathbf{q}_{T-1} = \mathbf{c}_{T-1} + \mathbf{F}_{T-1}^T\mathbf{V}_T\mathbf{f}_{T-1} + \mathbf{F}_{T-1}^T\mathbf{v}_T$$

$$\nabla_{\mathbf{u}_{T-1}}Q(\mathbf{x}_{T-1},\mathbf{u}_{T-1}) = \mathbf{Q}_{\mathbf{u}_{T-1},\mathbf{x}_{T-1}}\mathbf{x}_{T-1} + \mathbf{Q}_{\mathbf{u}_{T-1},\mathbf{u}_{T-1}}\mathbf{u}_{T-1} + \mathbf{q}_{\mathbf{u}_{T-1}}^T = 0$$

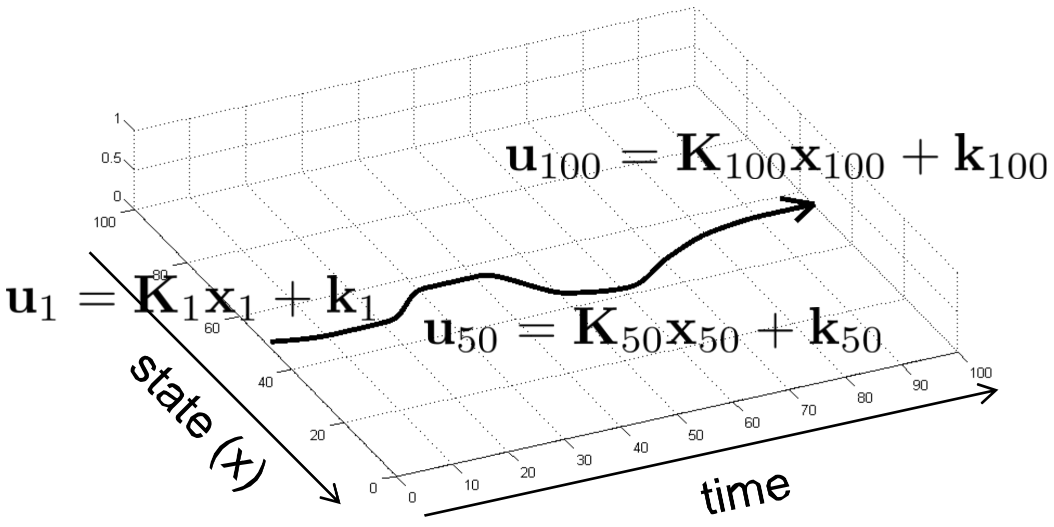
$$\mathbf{u}_{T-1} = \mathbf{K}_{T-1}\mathbf{x}_{T-1} + \mathbf{k}_{T-1} \qquad \mathbf{K}_{T-1} = -\mathbf{Q}_{\mathbf{u}_{T-1},\mathbf{u}_{T-1}}^{-1}\mathbf{Q}_{\mathbf{u}_{T-1},\mathbf{x}_{T-1}}$$

$$\mathbf{k}_{T-1} = -\mathbf{Q}_{\mathbf{u}_{T-1},\mathbf{u}_{T-1}}^{-1}\mathbf{q}_{\mathbf{u}_{T-1}}$$

Linear case: LQR

Backward recursion

```
for $t = T$ to 1:
    $\mathbf{Q}_t = \mathbf{C}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{F}_t$
    $\mathbf{q}_t = \mathbf{c}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{f}_t + \mathbf{F}_t^T \mathbf{v}_{t+1}$
    $Q(\mathbf{x}_t, \mathbf{u}_t) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{Q}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{q}_t$
    $\mathbf{u}_t \leftarrow \arg\min_{\mathbf{u}_t} Q(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$
    $\mathbf{K}_t = -\mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t}^{-1} \mathbf{Q}_{\mathbf{u}_t,\mathbf{x}_t}$
    $\mathbf{k}_t = -\mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t}^{-1} \mathbf{q}_{\mathbf{u}_t}$
    $\mathbf{V}_t = \mathbf{Q}_{\mathbf{x}_t,\mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t,\mathbf{u}_t} \mathbf{K}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t,\mathbf{x}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t} \mathbf{K}_t$
    $\mathbf{v}_t = \mathbf{q}_{\mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t,\mathbf{u}_t} \mathbf{k}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t} \mathbf{k}_t$
    $V(\mathbf{x}_t) = \text{const} + \frac{1}{2} \mathbf{x}_t^T \mathbf{V}_t \mathbf{x}_t + \mathbf{x}_t^T \mathbf{v}_t$
```



$\mathbf{u}_{100} = \mathbf{K}_{100} \mathbf{x}_{100} + \mathbf{k}_{100}$

$\mathbf{u}_1 = \mathbf{K}_1 \mathbf{x}_1 + \mathbf{k}_1$

$\mathbf{u}_{50} = \mathbf{K}_{50} \mathbf{x}_{50} + \mathbf{k}_{50}$

we know $\mathbf{x}_1$!

Forward recursion

```
for $t = 1$ to $T$:
    $\mathbf{u}_t = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$
    $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$
```

Linear case: LQR

Backward recursion

```
for $t = T$ to 1:
    $\mathbf{Q}_t = \mathbf{C}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{F}_t$
    $\mathbf{q}_t = \mathbf{c}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{f}_t + \mathbf{F}_t^T \mathbf{v}_{t+1}$
    $Q(\mathbf{x}_t, \mathbf{u}_t) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{Q}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{q}_t$
    $\mathbf{u}_t \leftarrow \arg\min_{\mathbf{u}_t} Q(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$
    $\mathbf{K}_t = -\mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t}^{-1} \mathbf{Q}_{\mathbf{u}_t,\mathbf{x}_t}$
    $\mathbf{k}_t = -\mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t}^{-1} \mathbf{q}_{\mathbf{u}_t}$
    $\mathbf{V}_t = \mathbf{Q}_{\mathbf{x}_t,\mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t,\mathbf{u}_t} \mathbf{K}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t,\mathbf{x}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t} \mathbf{K}_t$
    $\mathbf{v}_t = \mathbf{q}_{\mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t,\mathbf{u}_t} \mathbf{k}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t} \mathbf{k}_t$
    $V(\mathbf{x}_t) = \text{const} + \frac{1}{2} \mathbf{x}_t^T \mathbf{V}_t \mathbf{x}_t + \mathbf{x}_t^T \mathbf{v}_t$
```

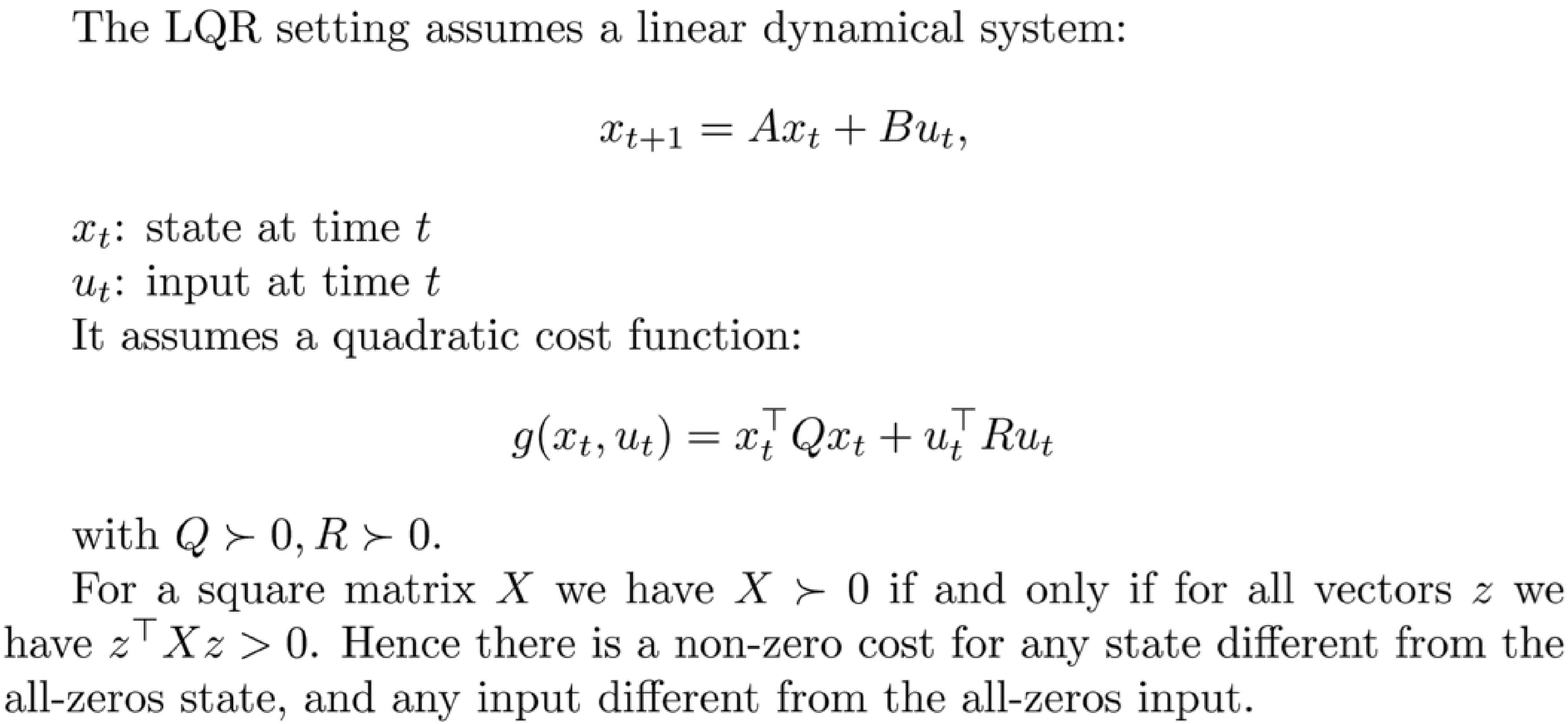
total cost from now until end if we take $\mathbf{u}_t$ from state $\mathbf{x}_t$

total cost from now until end from state $\mathbf{x}_t$

$$V(\mathbf{x}_t) = \min_{\mathbf{u}_t} Q(\mathbf{x}_t, \mathbf{u}_t)$$

LQR : Extensions

Linear Quadradic Regulator (LQR)



The LQR setting assumes a linear dynamical system:

$$x_{t+1} = Ax_t + Bu_t,$$

x_t: state at time t
u_t: input at time t
It assumes a quadratic cost function:

$$g(x_t, u_t) = x_t^\top Q x_t + u_t^\top R u_t$$

with $Q \succ 0, R \succ 0$.

For a square matrix X we have $X \succ 0$ if and only if for all vectors z we have $z^\top X z > 0$. Hence there is a non-zero cost for any state different from the all-zeros state, and any input different from the all-zeros input.

LQR vs Dynamic Programming

- Value Iteration
  Back-up step for i+1 steps to go:

$$J_{i+1}(s) = \min_u g(s,u) + \sum_{s'} P(s'|s,u) J_i(s')$$

- LQR:

$$J_{i+1}(x) = \min_u x^\top Q x + u^\top R u + \sum_{x'=Ax+Bu} J_i(x')$$

$$= \min_u \left[x^\top Q x + u^\top R u + J_i(Ax+Bu) \right]$$

LQR Value Iteration (DP)

$$J_{i+1}(x) \;\leftarrow\; \min_u \left[x^\top Q x + u^\top R u + J_i(Ax + Bu)\right]$$

Initialize $J_0(x) = x^\top P_0 x$.

$$\begin{aligned} J_1(x) &= \min_u \left[x^\top Q x + u^\top R u + J_0(Ax + Bu)\right] \\ &= \min_u \left[x^\top Q x + u^\top R u + (Ax + Bu)^\top P_0 (Ax + Bu)\right] \qquad (1) \end{aligned}$$

To find the minimum over u, we set the gradient w.r.t. u equal to zero:

$$\nabla_u [\ldots] = 2Ru + 2B^\top P_0(Ax + Bu) = 0,$$

$$\text{hence: } u = -(R + B^\top P_0 B)^{-1} B^\top P_0 A x \qquad (2)$$

$$\begin{aligned} \text{(2) into (1): } J_1(x) &= x^\top P_1 x \\ \text{for: } P_1 &= Q + K_1^\top R K_1 + (A + BK_1)^\top P_0 (A + BK_1) \\ K_1 &= -(R + B^\top P_0 B)^{-1} B^\top P_0 A. \end{aligned}$$

LQR Value Iteration (DP)

- In Summary:

$$\begin{aligned} J_0(x) &= x^\top P_0 x \\ x_{t+1} &= Ax_t + Bu_t \\ g(x,u) &= u^\top R u + x^\top Q x \end{aligned}$$

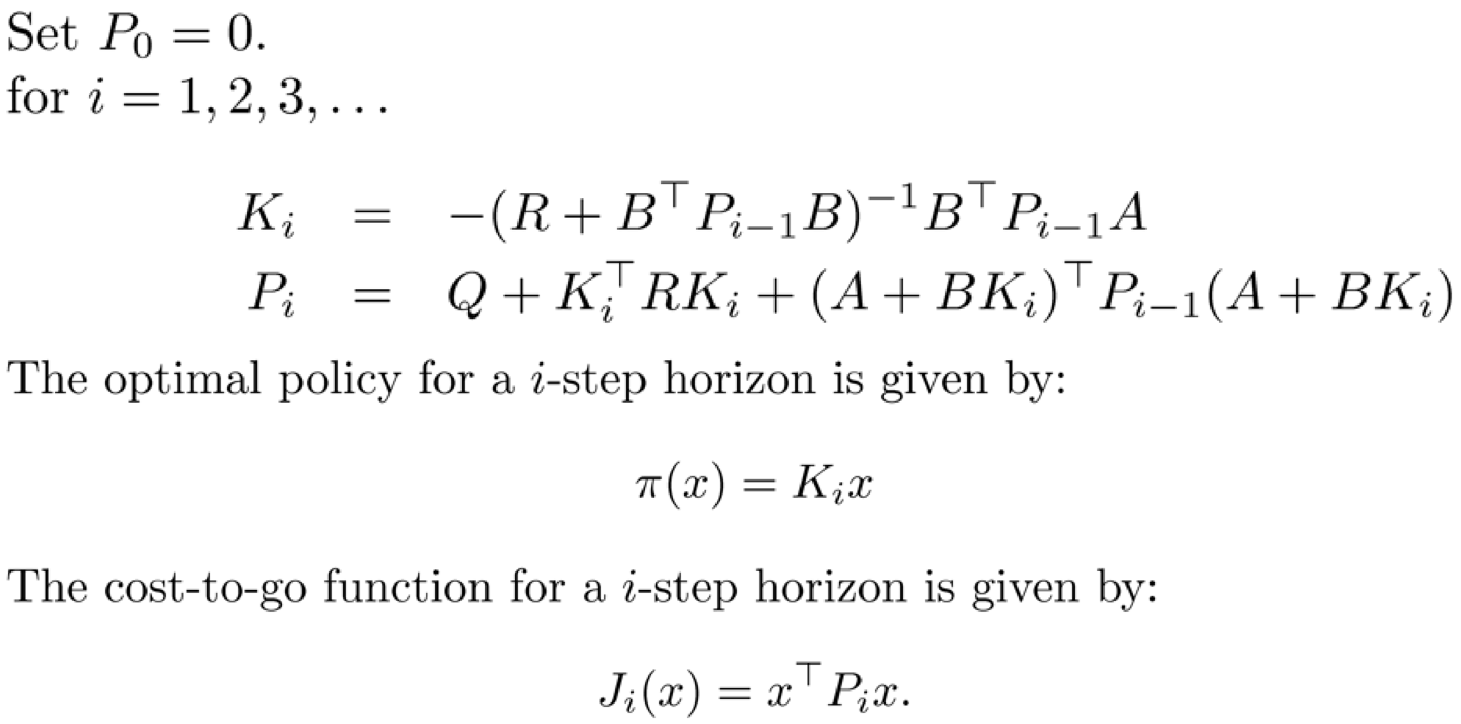
$$\boxed{\begin{array}{rcl} J_1(x) & = & x^\top P_1 x \\ \text{for: } P_1 & = & Q + K_1^\top R K_1 + (A + BK_1)^\top P_0 (A + BK_1) \\ K_1 & = & -(R + B^\top P_0 B)^{-1} B^\top P_0 A. \end{array}}$$

- $J_1(x)$ is quadratic, just like $J_0(x)$

Value iteration update is the same for all times and can be done in closed form for this particular continuous state-space system and cost!

$$\begin{array}{rcl} J_2(x) & = & x^\top P_2 x \\ \text{for: } P_2 & = & Q + K_2^\top R K_2 + (A + BK_2)^\top P_1 (A + BK_2) \\ K_2 & = & -(R + B^\top P_1 B)^{-1} B^\top P_1 A. \end{array}$$

Value Iteration Solution to LQR



Set $P_0 = 0$.
for $i = 1, 2, 3, \ldots$

$$\begin{aligned} K_i &= -(R + B^\top P_{i-1} B)^{-1} B^\top P_{i-1} A \\ P_i &= Q + K_i^\top R K_i + (A + B K_i)^\top P_{i-1} (A + B K_i) \end{aligned}$$

The optimal policy for a i-step horizon is given by:

$$\pi(x) = K_i x$$

The cost-to-go function for a i-step horizon is given by:

$$J_i(x) = x^\top P_i x.$$

- Guaranteed to converge to the infinite horizon optimal policy if and only if the dynamics (A, B) is such that there exists a policy that can drive the state to zero.
- Often most convenient to use the steady-state K for all times.

LQR: Assumptions

- Keep a linear system at the all-zeros state while preferring to keep the control input small.

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t \\ g(x_t, u_t) &= x_t^\top Q x_t + u_t^\top R u_t \end{aligned}$$

Extensions make it more generally applicable:

- Affine systems
- Systems with stochasticity
- Regulation around non-zero fixed point for non-linear systems
- Penalization for change in control inputs
- Linear time varying (LTV) systems
- Trajectory following for non-linear systems

LQR EXT: Affine Systems

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t + c \\ g(x_t, u_t) &= x_t^\top Q x_t + u_t^\top R u_t \end{aligned}$$

Optimal control policy remains linear, optimal cost-to-go function remains quadratic

Two avenues to do derivation:

1. Re-derive the update, which is very similar to what we did for standard setting

2. Re-define the state as: z_t = [x_t; 1], then we have:

$$z_{t+1} = \begin{bmatrix} x_{t+1} \\ 1 \end{bmatrix} = \begin{bmatrix} A & c \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ 1 \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u_t = A' z_t + B' u_t$$

LQR EXT: Stochastic Systems

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t + w_t \\ g(x_t, u_t) &= x_t^\top Q x_t + u_t^\top R u_t \\ & \quad w_t, t = 0, 1, \ldots \text{are zero mean and independent} \end{aligned}$$

Exercise: work through similar derivation as we did for the deterministic case, but which will now have expectations.

Results

- Same optimal control policy
- Cost-to-go function is almost identical: has one additional term which depends on the variance in the noise (and which cannot be influenced by the choice of control inputs)

LQR EXT: Non-Linear Systems

Non-Linear System: $x_{t+1} = f(x_t, u_t)$

We can keep the system at the state x* iff

$$\exists u^* \text{s.t.} \quad x^* = f(x^*, u^*)$$

Linearizing the dynamics around x* gives:

$$x_{t+1} \approx f(x^*, u^*) + \underbrace{\frac{\partial f}{\partial x}(x^*, u^*)}_{A}(x_t - x^*) + \underbrace{\frac{\partial f}{\partial u}(x^*, u^*)}_{B}(u_t - u^*)$$

$$x_{t+1} - x^* \approx A(x_t - x^*) + B(u_t - u^*)$$

Let $z_t = x_t - x^*$, let $v_t = u_t - u^*$, then:

$$z_{t+1} = Az_t + Bv_t, \qquad \text{cost} = z_t^\top Q z_t + v_t^\top R v_t$$

$$v_t = Kz_t \Rightarrow u_t - u^* = K(x_t - x^*) \Rightarrow u_t = u^* + K(x_t - x^*)$$

LQR Ext: Penalize for Change in Control Inputs

- Standard

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t \\ g(x_t, u_t) &= x_t^\top Q x_t + u_t^\top R u_t \end{aligned}$$

- When run in this format on real systems: often high frequency control inputs get generated. Typically highly undesirable and results in poor control performance.
- Why?
- Solution: frequency shaping of the cost function. Can be done by augmenting the system with a filter and then the filter output can be used in the quadratic cost function. (See, e.g., Anderson and Moore.)
- Simple special case which works well in practice: penalize for change in control inputs.
- How ??

LQR Ext: Penalize for Change in Control Inputs

- Standard

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t \\ g(x_t, u_t) &= x_t^\top Q x_t + u_t^\top R u_t \end{aligned}$$

How to incorporate the change in controls into the cost/reward function?

- Soln. method A: explicitly incorporate into the state by augmenting the state with the past control input vector, and the difference between the last two control input vectors.
- Soln. method B: change of control input variables.

LQR Ext: Penalize for Change in Control Inputs

- Standard

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t \\ g(x_t, u_t) &= x_t^\top Q x_t + u_t^\top R u_t \end{aligned}$$

Introducing change in controls Δu

$$\underbrace{\begin{bmatrix} x_{t+1} \\ u_t \end{bmatrix}}_{x'_{t+1}} = \underbrace{\begin{bmatrix} A & B \\ 0 & I \end{bmatrix}}_{A'} \underbrace{\begin{bmatrix} x_t \\ u_{t-1} \end{bmatrix}}_{x'_t} + \underbrace{\begin{bmatrix} B \\ I \end{bmatrix}}_{B'} \underbrace{\Delta u_t}_{u'_t}$$

$$x'_{t+1} = A' \quad x'_t + B' \quad u'_t$$

$$\text{cost} = -(x'^\top Q' x' + \Delta u^\top R' \Delta u) \qquad Q' = \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix}$$

$$R' = \text{penalty for change in controls}$$

[If R’=0, then “equivalent” to standard LQR.]

LQR Ext: Linear Time Varying Systems (LTV)

$$\begin{aligned} x_{t+1} &= A_t x_t + B_t u_t \\ g(x_t, u_t) &= x_t^\top Q_t x_t + u_t^\top R_t u_t \end{aligned}$$

Set $P_0 = 0$.
for $i = 1, 2, 3, \ldots$

$$\begin{aligned} K_i &= -(R_{H-i} + B_{H-i}^\top P_{i-1} B_{H-i})^{-1} B_{H-i}^\top P_{i-1} A_{H-i} \\ P_i &= Q_{H-i} + K_i^\top R_{H-i} K_i + (A_{H-i} + B_{H-i} K_i)^\top P_{i-1} (A_{H-i} + B_{H-i} K_i) \end{aligned}$$

The optimal policy for a i-step horizon is given by:

$$\pi(x) = K_i x$$

The cost-to-go function for a i-step horizon is given by:

$$J_i(x) = x^\top P_i x.$$

LQR Ext:
Trajectory Following for Non-Linear Systems

- A state sequence x_0^*, x_1^*, …, x_H^* is a feasible target trajectory if and only if

$$\exists u_0^*, u_1^*, \ldots, u_{H-1}^* \;:\; \forall t \in \{0, 1, \ldots, H-1\} \;:\; x_{t+1}^* = f(x_t^*, u_t^*)$$

- Problem Statement

$$\min_{u_0, u_1, \ldots, u_{H-1}} \textstyle\sum_{t=0}^{H-1} (x_t - x_t^*)^\top Q (x_t - x_t^*) + (u_t - u_t^*)^\top R (u_t - u_t^*)$$

$$\text{s.t.} \quad x_{t+1} = f(x_t, u_t)$$

- Transform into Linear Time Varying System

$$x_{t+1} \approx f(x_t^*, u_t^*) + \underbrace{\frac{\partial f}{\partial x}(x_t^*, u_t^*)}_{A_t}(x_t - x_t^*) + \underbrace{\frac{\partial f}{\partial u}(x_t^*, u_t^*)}_{B_t}(u_t - u_t^*)$$

$$x_{t+1} - x_{t+1}^* \approx A_t (x_t - x_t^*) + B_t (u_t - u_t^*)$$

LQR Ext: Trajectory Following for Non-Linear Systems

- Transform into Linear Time Varying System

$$x_{t+1} \approx f(x_t^*, u_t^*) + \underbrace{\frac{\partial f}{\partial x}(x_t^*, u_t^*)}_{A_t}(x_t - x_t^*) + \underbrace{\frac{\partial f}{\partial u}(x_t^*, u_t^*)}_{B_t}(u_t - u_t^*)$$

$$x_{t+1} - x_{t+1}^* \approx A_t(x_t - x_t^*) + B_t(u_t - u_t^*)$$

Now we can run the standard LQR back-up iterations.

Resulting policy at i time-steps from the end:

$$u_{H-i} - u_{H-i}^* = K_i(x_{H-i} - x_{H-i}^*)$$

The target trajectory need not be feasible to apply this technique, however, if it is infeasible then there will an offset term in the dynamics:

$$x_{t+1} - x_{t+1}^* = f(x_t, u_t) - x_{t+1}^* + A_t(x_t - x_t^*) + B_t(u_t - u_t^*)$$

LQR for Stochastic and Nonlinear Systems

General Optimal Control

- What about

$$\min_{u_0,\ldots,u_H} \sum_{t=0}^{H} g(x_t, u_t)$$

Stochastic dynamics

$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t$$

$$\mathbf{x}_{t+1} \sim p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$$

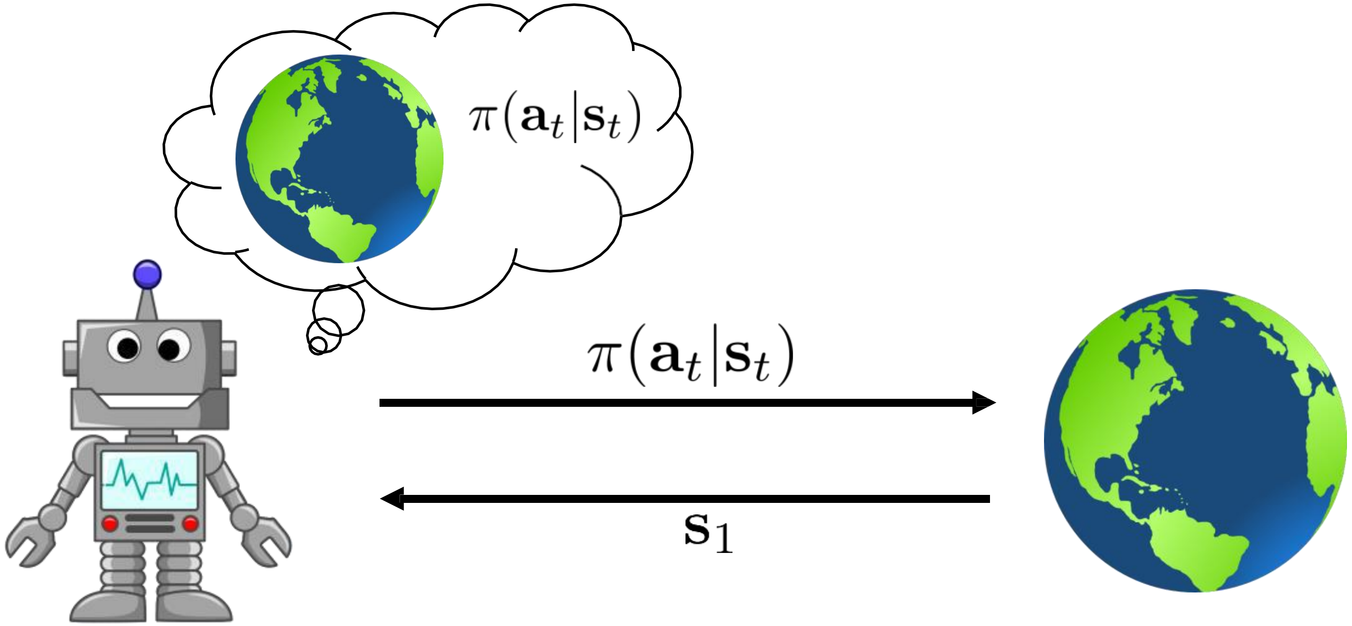
$$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}\left(\mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t, \Sigma_t\right)$$

Solution: choose actions according to $\mathbf{u}_t = \mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t$

$\mathbf{x}_t \sim p(\mathbf{x}_t)$, no longer deterministic, but $p(\mathbf{x}_t)$ is Gaussian

no change to algorithm! can ignore Σ_t due to symmetry of Gaussians (checking this is left as an exercise; hint: the expectation of a quadratic under a Gaussian has an analytic solution)

The stochastic closed-loop case


$\pi(\mathbf{a}_t|\mathbf{s}_t)$
$\pi(\mathbf{a}_t|\mathbf{s}_t)$
$\mathbf{s}_1$

form of π?

$$p(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

time-varying linear

$\mathbf{K}_t\mathbf{s}_t + \mathbf{k}_t$

$$\pi = \arg\max_{\pi} E_{\tau \sim p(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

Nonlinear case: DDP/iterative LQR

Linear-quadratic assumptions:

$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t \qquad c(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{C}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{c}_t$$

Can we *approximate* a nonlinear system as a linear-quadratic system?

$$f(\mathbf{x}_t, \mathbf{u}_t) \approx f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}$$

$$c(\mathbf{x}_t, \mathbf{u}_t) \approx c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}^T \nabla^2_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}$$

Nonlinear case: DDP/iterative LQR

$$f(\mathbf{x}_t, \mathbf{u}_t) \approx f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t,\mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \left[\begin{array}{c} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{array} \right]$$

$$c(\mathbf{x}_t, \mathbf{u}_t) \approx c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \left[\begin{array}{c} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{array} \right] + \frac{1}{2} \left[\begin{array}{c} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{array} \right]^T \nabla^2_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \left[\begin{array}{c} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{array} \right]$$

$$\bar{f}(\delta\mathbf{x}_t, \delta\mathbf{u}_t) = \underbrace{\mathbf{F}_t}_{\nabla_{\mathbf{x}_t,\mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)} \left[\begin{array}{c} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{array} \right]$$

$$\bar{c}(\delta\mathbf{x}_t, \delta\mathbf{u}_t) = \frac{1}{2} \left[\begin{array}{c} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{array} \right]^T \underbrace{\mathbf{C}_t}_{\nabla^2_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)} \left[\begin{array}{c} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{array} \right] + \left[\begin{array}{c} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{array} \right]^T \underbrace{\mathbf{c}_t}_{\nabla_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)}$$

$$\delta\mathbf{x}_t = \mathbf{x}_t - \hat{\mathbf{x}}_t$$

$$\delta\mathbf{u}_t = \mathbf{u}_t - \hat{\mathbf{u}}_t$$

Now we can run LQR with dynamics $\bar{f}$, cost $\bar{c}$, state $\delta\mathbf{x}_t$, and action $\delta\mathbf{u}_t$

Nonlinear case: DDP/iterative LQR

Iterative LQR (simplified pseudocode)

```
until convergence:
    $\mathbf{F}_t = \nabla_{\mathbf{x}_t,\mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$
    $\mathbf{c}_t = \nabla_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$
    $\mathbf{C}_t = \nabla^2_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$
    Run LQR backward pass on state $\delta\mathbf{x}_t = \mathbf{x}_t - \hat{\mathbf{x}}_t$ and action $\delta\mathbf{u}_t = \mathbf{u}_t - \hat{\mathbf{u}}_t$
    Run forward pass with real nonlinear dynamics and $\mathbf{u}_t = \mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t$
    Update $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{u}}_t$ based on states and actions in forward pass
```

Nonlinear case: DDP/iterative LQR

Why does this work?

Compare to Newton's method for computing $\min_{\mathbf{x}} g(\mathbf{x})$:

```
until convergence:
    $\mathbf{g} = \nabla_{\mathbf{x}} g(\hat{\mathbf{x}})$
    $\mathbf{H} = \nabla^2_{\mathbf{x}} g(\hat{\mathbf{x}})$
    $\hat{\mathbf{x}} \leftarrow \arg\min_{\mathbf{x}} \frac{1}{2}(\mathbf{x} - \hat{\mathbf{x}})^T \mathbf{H} (\mathbf{x} - \hat{\mathbf{x}}) + \mathbf{g}^T (\mathbf{x} - \hat{\mathbf{x}})$
```

Iterative LQR (iLQR) is the same idea: locally approximate a complex nonlinear function via Taylor expansion

In fact, iLQR is an approximation of Newton's method for solving

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \cdots + c(f(f(\ldots)\ldots), \mathbf{u}_T)$$

Nonlinear case: DDP/iterative LQR

In fact, iLQR is an approximation of Newton's method for solving

$$\min_{\mathbf{u}_1,\dots,\mathbf{u}_T} c(\mathbf{x}_1,\mathbf{u}_1)+c(f(\mathbf{x}_1,\mathbf{u}_1),\mathbf{u}_2)+\dots+c(f(f(\dots)\dots),\mathbf{u}_T)$$

To get Newton's method, need to use *second order* dynamics approximation:

$$f(\mathbf{x}_t,\mathbf{u}_t)\approx f(\hat{\mathbf{x}}_t,\hat{\mathbf{u}}_t)+\nabla_{\mathbf{x}_t,\mathbf{u}_t} f(\hat{\mathbf{x}}_t,\hat{\mathbf{u}}_t)\left[\begin{array}{c}\delta\mathbf{x}_t\\ \delta\mathbf{u}_t\end{array}\right]+\frac{1}{2}\left(\nabla^2_{\mathbf{x}_t,\mathbf{u}_t} f(\hat{\mathbf{x}}_t,\hat{\mathbf{u}}_t)\cdot\left[\begin{array}{c}\delta\mathbf{x}_t\\ \delta\mathbf{u}_t\end{array}\right]\right)\left[\begin{array}{c}\delta\mathbf{x}_t\\ \delta\mathbf{u}_t\end{array}\right]$$

differential dynamic programming (DDP)

General Optimal Control
Iterative LQR: Practical Concerns

- F is non-linear hence this is a Non-Convex Optimization Problem.
  - Can get Stuck in local minima.
  - Good initialization matters

- If g is non-convex, then LQ could fail to have Positive-Definite Cost Matrices.
  - Practical Fix: if Q_t and R_t are not PD, then
    - Increase penalty of deviating from the current state and control (x_t, u_t), until resulting Q_t and R_t are Positive definite.

General Optimal Control
Differential Dynamic Programming

Often loosely referred to as Iterative LQR Procedure

Details:

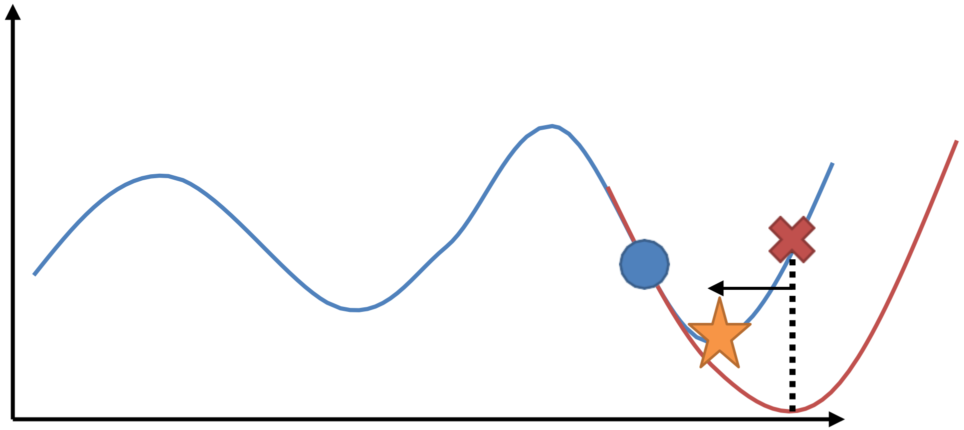
- Don't: Linearize dynamics and 2nd order Taylor expansion of Costs
- Do's: Directly perform 2nd order Taylor expansion from the Bellman backup equation

- This retains a term which is otherwise discarded in Iterative LQR approach.
    - It's a quadratic term in the Dynamics Model
    - So even if the cost is convex, resulting LQ problem can be non-convex.

Nonlinear case: DDP/iterative LQR

$$\hat{\mathbf{x}} \leftarrow \arg\min_{\mathbf{x}} \frac{1}{2}(\mathbf{x}-\hat{\mathbf{x}})^T\mathbf{H}(\mathbf{x}-\hat{\mathbf{x}}) + \mathbf{g}^T(\mathbf{x}-\hat{\mathbf{x}})$$

why is this a bad idea?



until convergence:

$$\mathbf{F}_t = \nabla_{\mathbf{x}_t,\mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

$$\mathbf{c}_t = \nabla_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

$$\mathbf{C}_t = \nabla^2_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

Run LQR backward pass on state $\delta\mathbf{x}_t = \mathbf{x}_t - \hat{\mathbf{x}}_t$ and action $\delta\mathbf{u}_t = \mathbf{u}_t - \hat{\mathbf{u}}_t$

~~Run forward pass with $\mathbf{u}_t = \mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t$~~

search over α
until improvement achieved

Run forward pass with $\mathbf{u}_t = \mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \alpha\mathbf{k}_t + \hat{\mathbf{u}}_t$

Update $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{u}}_t$ based on states and actions in forward pass

General Optimal Control

- Does it work?
  Need not converge as formulated!
- Reason: the optimal policy for the LQ approximation might end up not staying close to the sequence of points around which the LQ approximation was computed by Taylor expansion
- Solution: in each iteration, adjust the cost function so this is the case, i.e., use the cost function

$$(1-\alpha)g(x_t, u_t) + \alpha(\|x_t - x_t^{(i)}\|_2^2 + \|u_t - u_t^{(i)}\|_2^2)$$

- Assuming g is bounded, for α close enough to one, the 2nd term will dominate and ensure the linearizations are good approximations around the solution trajectory found by LQR.
  I.e., the extra term acts like a trust region

General Optimal Control
Differential Dynamic Programming

At convergence, in both iLQR and DDP, we end up with the linearizations around the (state, input) trajectory

In Practice: the system could not be in this trajectory due to perturbations or initial state deviations or incorrect dynamics model or some other noise factor.

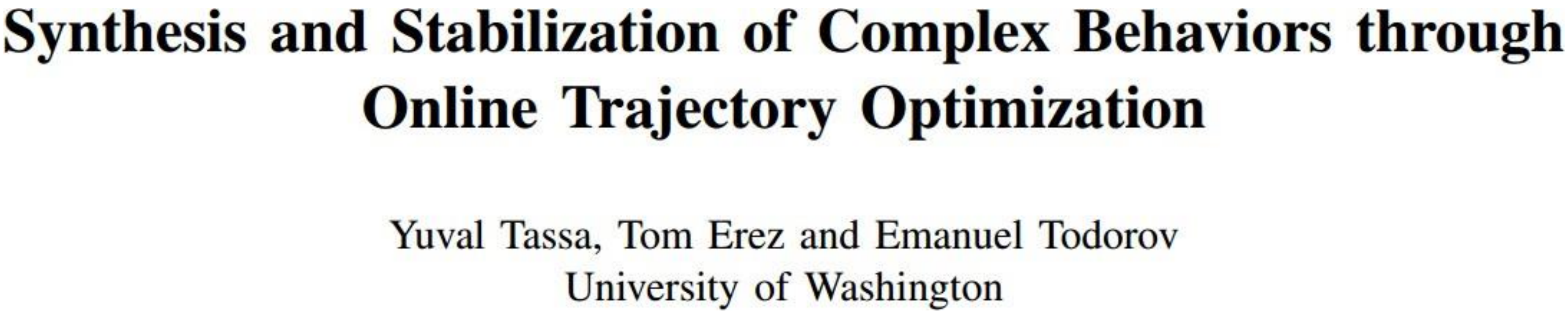
Solution: When asked to generate control input u_t we could re-solve the control problem using iLQR or DDP over the time steps *t* through *H.*

Replanning entire trajectory is often computationally impractical.
Hence replan over horizon *H* – also known as Receding Horizon Control

- This requires a cost-to-go J^{t+h} which accounts for all future costs.
- This can be used from a previous offline iLQR or DDP run.

Case Study and Additional Readings

Case study: nonlinear model-predictive control

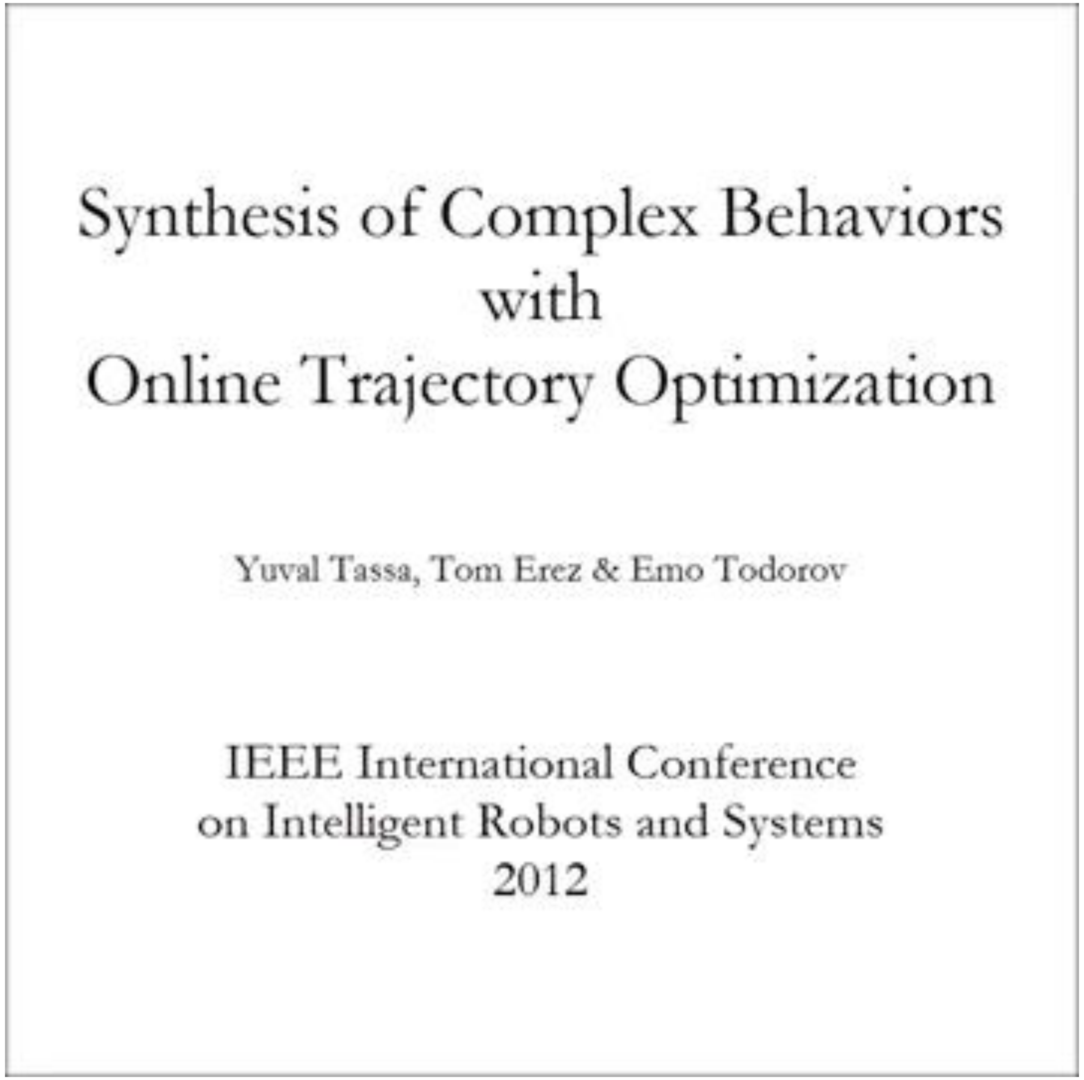


Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization

Yuval Tassa, Tom Erez and Emanuel Todorov
University of Washington

```
every time step:
    observe the state $\mathbf{x}_t$
    use iLQR to plan $\mathbf{u}_t, \dots, \mathbf{u}_T$ to minimize $\sum_{t'=t}^{t+T} c(\mathbf{x}_{t'}, \mathbf{u}_{t'})$
    execute action $\mathbf{u}_t$, discard $\mathbf{u}_{t+1}, \dots, \mathbf{u}_{t+T}$
```

Case study: nonlinear model-predictive control



Synthesis of Complex Behaviors
with
Online Trajectory Optimization

Yuval Tassa, Tom Erez & Emo Todorov

IEEE International Conference
on Intelligent Robots and Systems
2012

Additional reading

Mayne, Jacobson. (1970). Differential dynamic programming.

Original differential dynamic programming algorithm.

Tassa, Erez, Todorov. (2012). Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization.

Practical guide for implementing non-linear iterative LQR.

Levine, Abbeel. (2014). Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics.

Probabilistic formulation and trust region alternative to deterministic line search.

Additional Reading Material:
Videos: Steve Burton:

- Linear Quadratic Regulator (LQR) Control for the Inverted Pendulum on a Cart
- Linear Quadratic Gaussian

Other Notes:

- Underactuated Robotics by Russ Tedrake
  - http://underactuated.mit.edu/acrobot.html#section1
- LQR Note/Cheatsheet by Somil Bansal

Acknowledgements

Slides adapted from

CS 287 UC Berkeley
Pieter Abbeel

CS 285 UC Berkeley
Sergey Levine

CSC 498 Univ of Toronto
Animesh Garg