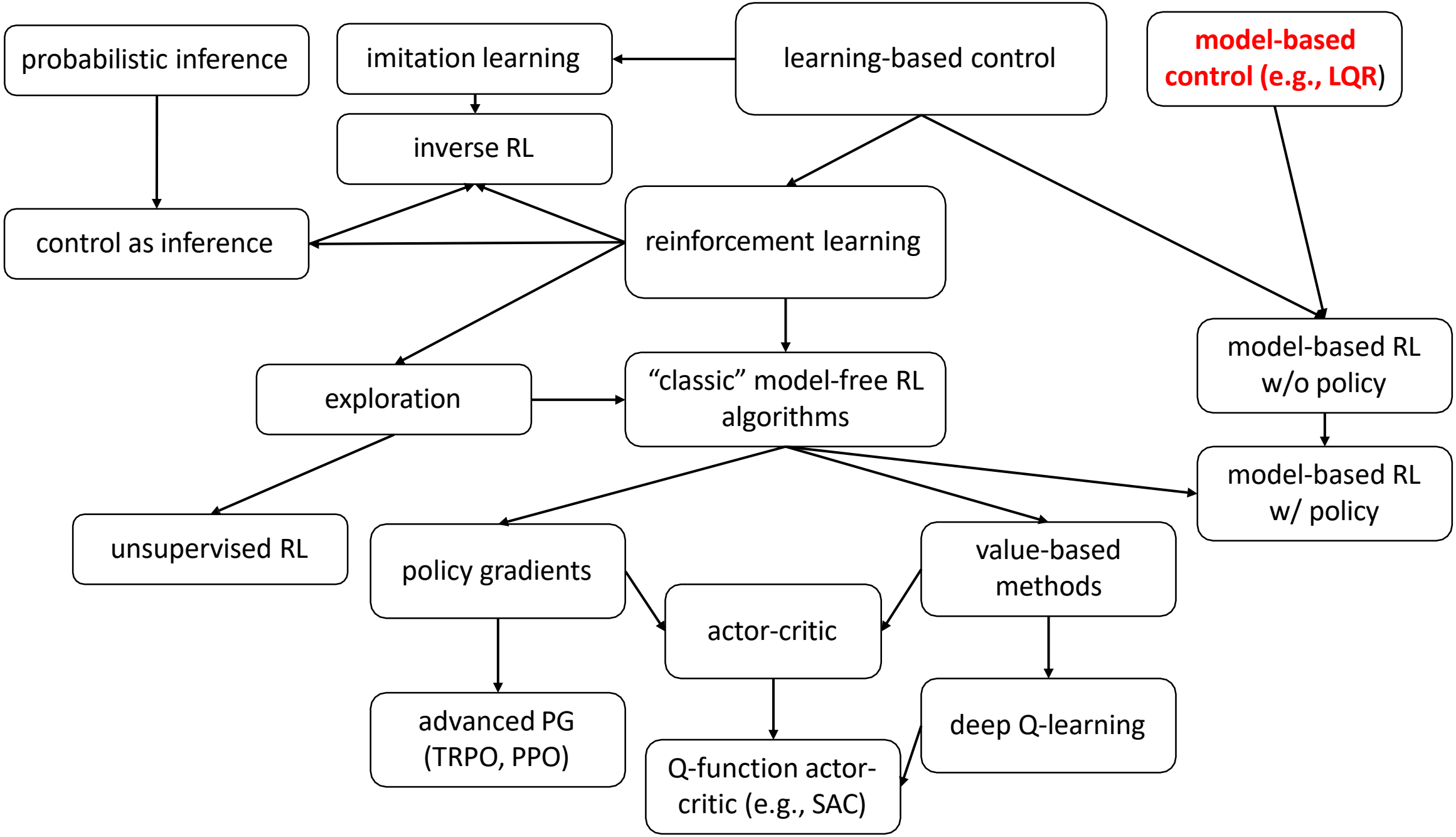
Optimal Control and Planning

CS 294-112: Deep Reinforcement Learning

Sergey Levine

https://www.youtube.com/playlist?list=PL_iWQOsE6TfURIIhCrIt-wj9ByIVpbfGc

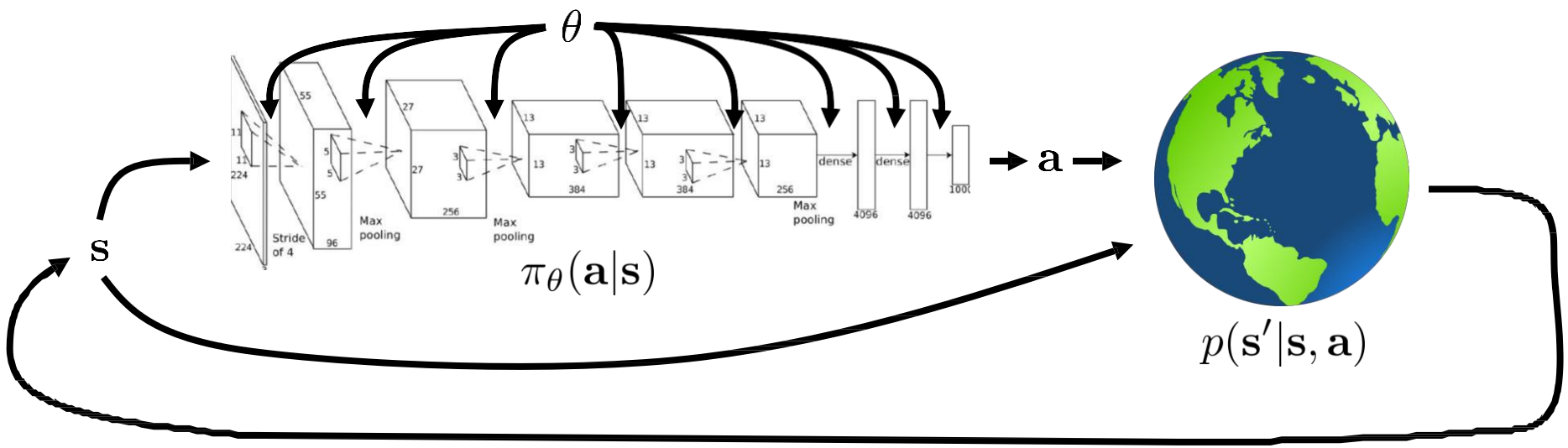

probabilistic inference
imitation learning
learning-based control
model-based control (e.g., LQR)
inverse RL
control as inference
reinforcement learning
model-based RL w/o policy
exploration
"classic" model-free RL algorithms
model-based RL w/ policy
unsupervised RL
policy gradients
value-based methods
actor-critic
advanced PG (TRPO, PPO)
deep Q-learning
Q-function actor-critic (e.g., SAC)

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

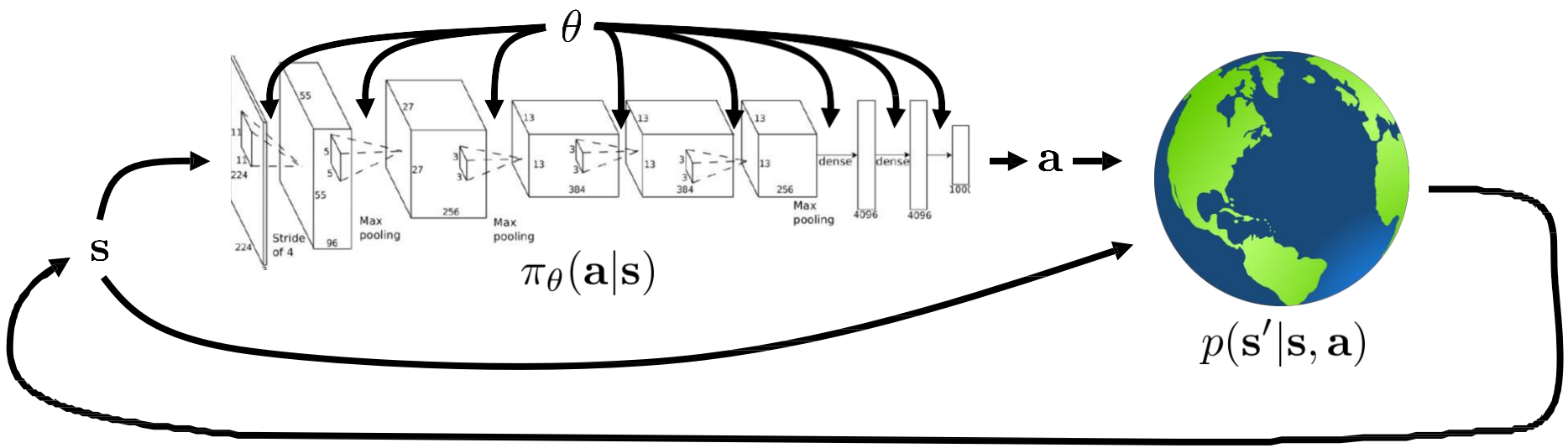
Recap: the reinforcement learning objective



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

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

Recap: model-free reinforcement learning



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

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)
     : rule
  1. Easily modeled systems (e.g., navigating a car)
     : **internal system (robot) dynamics model**
  1. Simulated environments (e.g., simulated robots, video games):
     : **external environment dynamics**

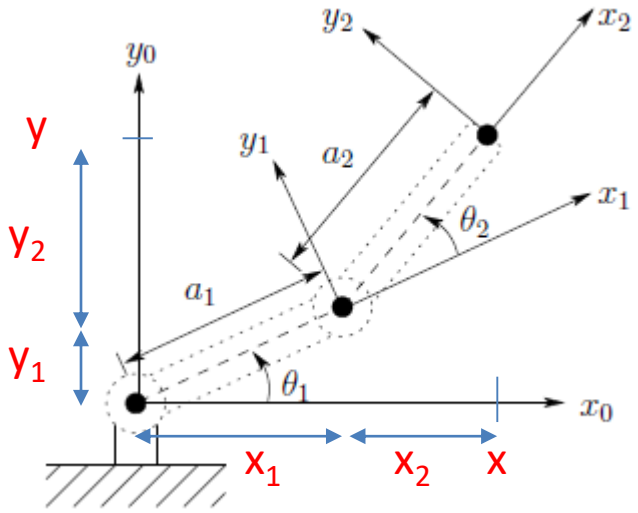
The rigid motions : kinematics: $\boldsymbol{H} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{d} \\ \boldsymbol{0} & \boldsymbol{1} \end{bmatrix}$

Dynamics: $D(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau$

- 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!

Forward Kinematics



Two-link planar manipulator

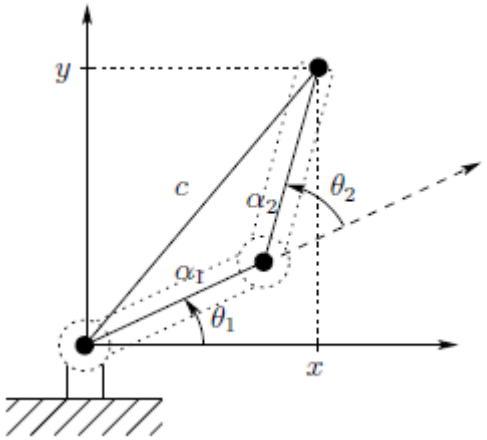
$FK(\cdot)$

Joint Space $(\theta_1, \theta_2) \implies$ Cartesian Space (x, y)

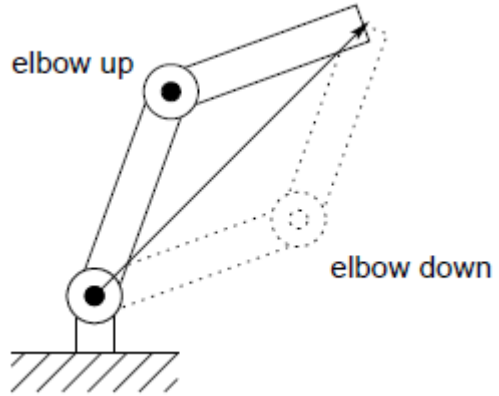
$$x = x_1 + x_2 = a_1 \cos\theta_1 + a_2 \cos(\theta_1 + \theta_2)$$

$$y = y_1 + y_2 = a_1 \sin\theta_1 + a_2 \sin(\theta_1 + \theta_2)$$

Inverse Kinematics



Two-link planar manipulator



Note: Multiple inverse kinematic solutions

$IK(\cdot)$

Joint Space (θ_1, θ_2) $\Leftarrow$ Cartesian Space (x, y)

$$\cos\theta_2 = \frac{x^2 + y^2 - a_1^2 - a_2^2}{2a_1a_2} := D \qquad \text{(law of cosines)} \qquad \left\{\Longrightarrow \sin\theta_2 = \pm\sqrt{1 - D^2}\right\}$$

$$\Longrightarrow \theta_2 = \tan^{-1}\left(\frac{\sin\theta_2}{\cos\theta_2}\right) = \tan^{-1}\left(\frac{\pm\sqrt{1 - D^2}}{D}\right)$$

$$\Longrightarrow \theta_1 = \tan^{-1}\left(\frac{y}{x}\right) - \tan^{-1}\left(\frac{a_2\sin\theta_2}{a_1 + a_2\cos\theta_2}\right)$$

Velocity Kinematics

From the forward kinematics of 2 DOF arm,

$$x = a_1 \cos\theta_1 + a_2 \cos(\theta_1 + \theta_2)$$
$$y = a_1 \sin\theta_1 + a_2 \, sin(\theta_1 + \theta_2)$$

the end-effector velocities are obtained as

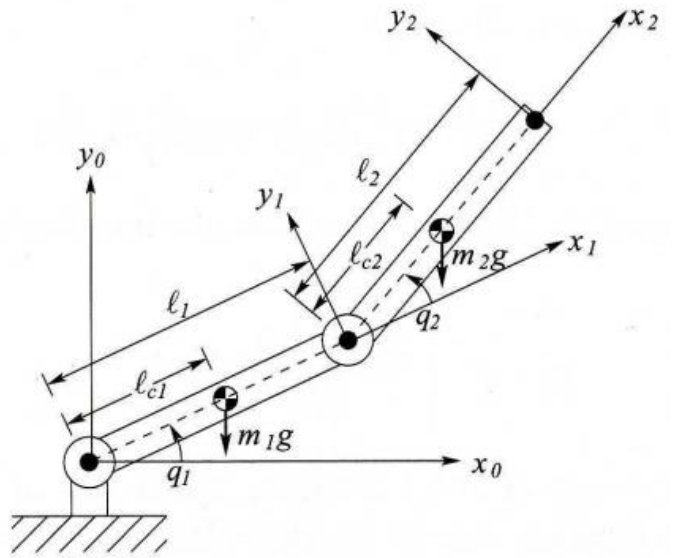
$$\dot{x} = \frac{dx}{dt} = -a_1 \sin\theta_1 \, \dot{\theta}_1 - a_2 \sin(\theta_1 + \theta_2)\left(\dot{\theta}_1 + \dot{\theta}_2\right)$$
$$\dot{y} = \frac{dy}{dt} = a_1 \cos\theta_1 \, \dot{\theta}_1 + a_2 \, cos(\theta_1 + \theta_2)\left(\dot{\theta}_1 + \dot{\theta}_2\right)$$

In compact form, it is

$$\dot{x} = \begin{bmatrix}\dot{x} \\ \dot{y}\end{bmatrix} = \begin{bmatrix}-a_1 \sin\theta_1 - a_2 \sin(\theta_1 + \theta_2) & -a_2 \sin(\theta_1 + \theta_2) \\ a_1 \cos\theta_1 + a_2 \, cos(\theta_1 + \theta_2) & +a_2 \, cos(\theta_1 + \theta_2)\end{bmatrix}\begin{bmatrix}\dot{\theta}_1 \\ \dot{\theta}_2\end{bmatrix}$$
$$= J\dot{\theta}$$

The matrix $J(\cdot)$ is called the Jacobian of the manipulator

Dynamics: Euler-Lagrange Equations



Dynamic equations of two-link revolute joint arm

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau$$

$$D(q) = \begin{bmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{bmatrix}$$

$$\begin{aligned} d_{11} &= m_1\ell_{c1}^2 + m_2(\ell_1^2 + \ell_{c2}^2 + 2\ell_1\ell_{c2}^2 + 2\ell_1\ell_{c2}\cos q_2) + I_1 + I_2 \\ d_{12} &= d_{21} = m_2(\ell_{c2}^2 + \ell_1\ell_{c2}\cos q_2) + I_2 \\ d_{22} &= m_2\ell_{c2}^2 + I_2 \end{aligned}$$

$$C = \begin{bmatrix} h\dot{q}_2 & h\dot{q}_2 + h\dot{q}_1 \\ -h\dot{q}_1 & 0 \end{bmatrix}$$

$$h = -m_2 l_1 l_{c2} \sin q_2$$

$$g(q) = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} = \begin{bmatrix} (m_1 l_{c1} + m_2 l_1) g \cos q_1 + m_2 l_{c2} g \cos(q_1 + q_2) \\ m_2 l_{c2} g \cos(q_1 + q_2) \end{bmatrix}$$

Model-based reinforcement learning

1. Model-based reinforcement learning
   : learn the transition dynamics, then figure out how to choose actions

1. Today: how can we make decisions if we *know* the dynamics?

   How can we choose actions **under perfect knowledge of the system dynamics**?

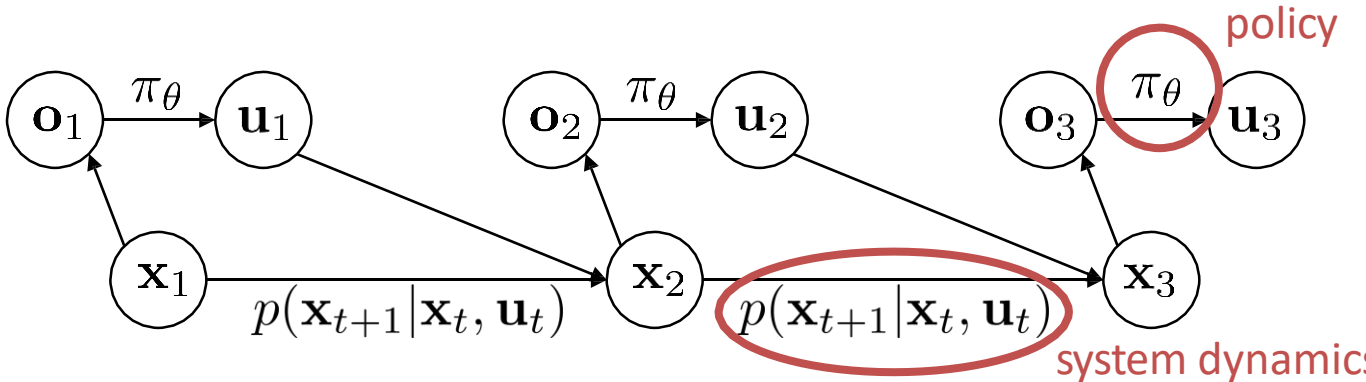
   (1) Optimal control, (2) **trajectory optimization**, (3) **planning**: similar terminology

(1) Selecting **controls (actions)** to minimize cost or maximize reward
(2) Selecting a **specific sequence of states and actions** that optimize some outcome – smooth gradient based
(3) Considering multiple discrete branches

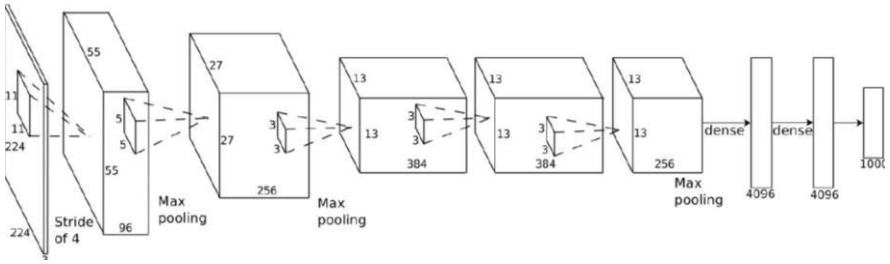
➔ **RL tackles optimal control from the perspective of learning !!!**

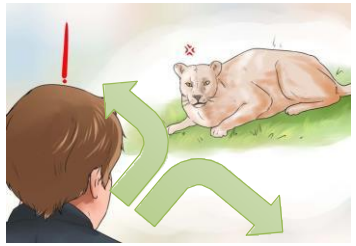
2. Next week: how can we learn *unknown* dynamics?

3. How can we then also **learn policies**? (*e.g.* ***by imitating optimal control***)



The objective





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

No policy, just states & actions

➔ planning (or trajectory optimization) problem

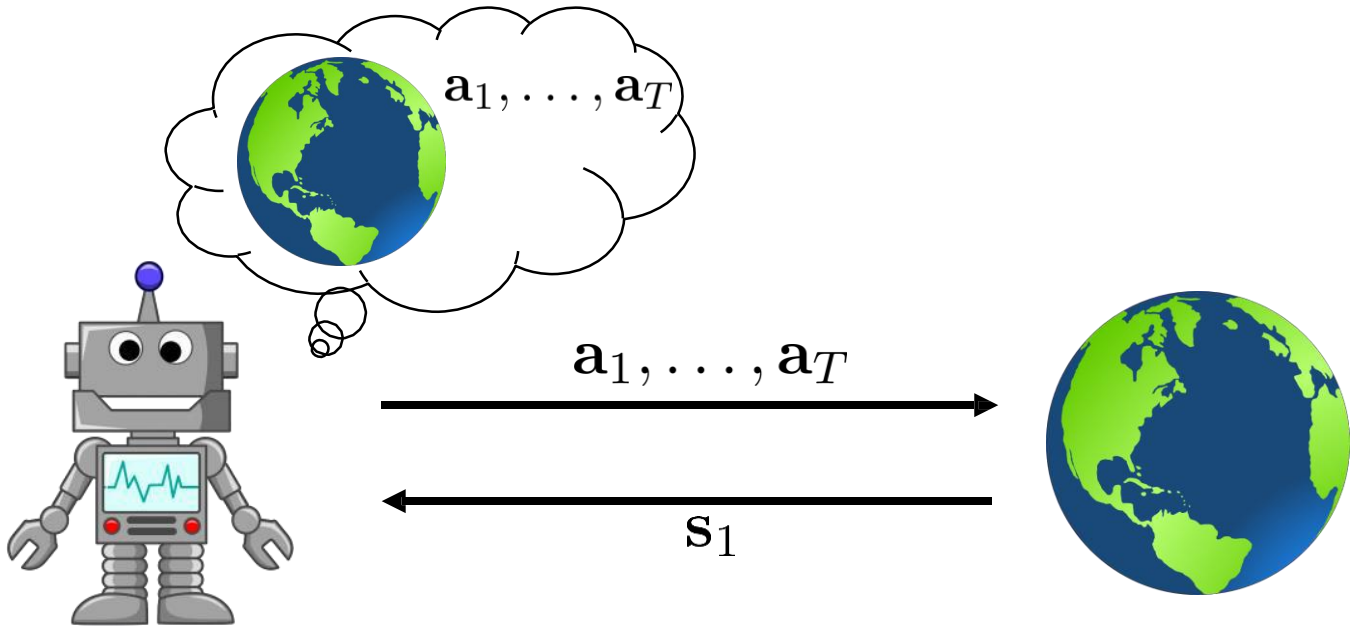
$$\min_{\mathbf{a}_1,\ldots,\mathbf{a}_T} \log p(\text{eaten by tiger}|\mathbf{a}_1,\ldots,\mathbf{a}_T)$$

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

Plan a **sequence of actions** that will minimize the probability of getting eaten by tiger

MBRL consists primarily of two aspects:
(1) learning a dynamics model
(2) using the learned dynamics models to plan and execute actions that
a) minimize a cost function (or b) maximize a reward function)

The deterministic case



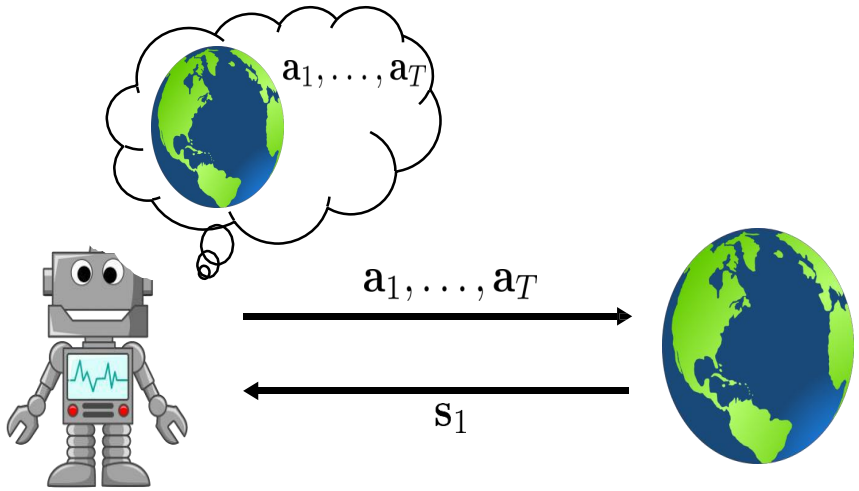
$$\mathbf{a}_1, \ldots, \mathbf{a}_T = \arg\max_{\mathbf{a}_1, \ldots, \mathbf{a}_T} \sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \text{ s.t. } \mathbf{a}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$$

s

Open-loop optimal control

Since deterministic policy ➔ next action is given right away: you don't know future states !!!

The stochastic open-loop case



why is this suboptimal (bad idea) in some cases?

: you don't know future states !!!

Impractical for two reasons:

(1) planning over an infinite sequence of actions is impossible

(2) the learned dynamics model is imperfect, so using it to plan in such an open-loop manner will lead to accumulating errors over time and planning far into the future will become very inaccurate

Stochastic dynamics: express things in terms of **distributions and expectations**

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

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

Stochastic !

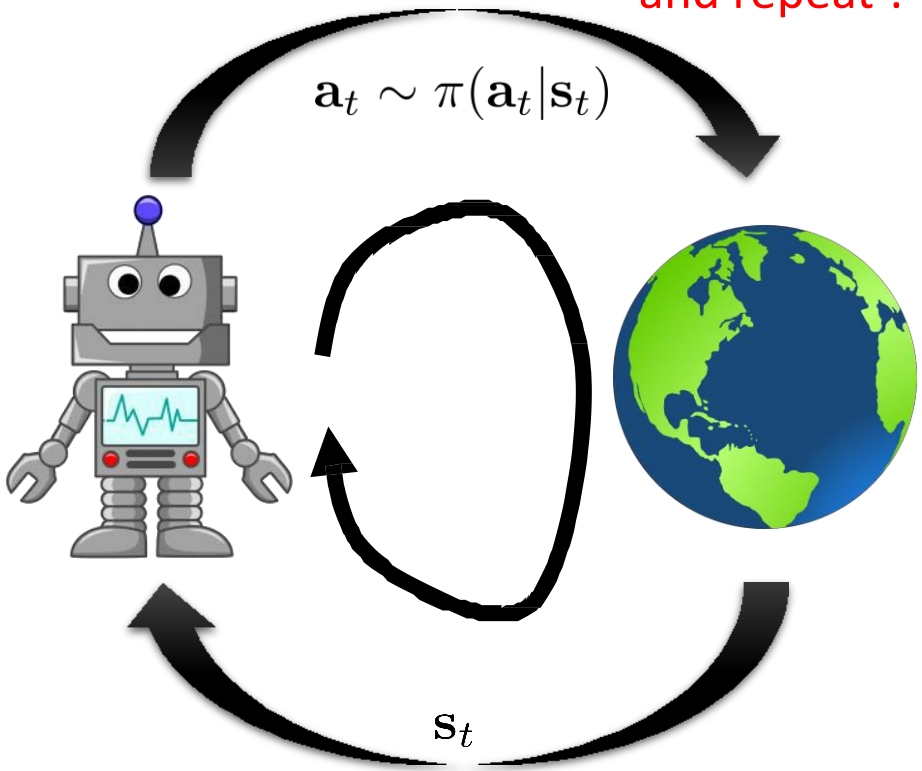
Exam Prob.!!! .Because a1➔ s2 may not be an optimal next state ➔ random action may be necessary from time to time ➔ **HW #4**

Aside: terminology

what is this “loop”? ➔ related to how a controller works

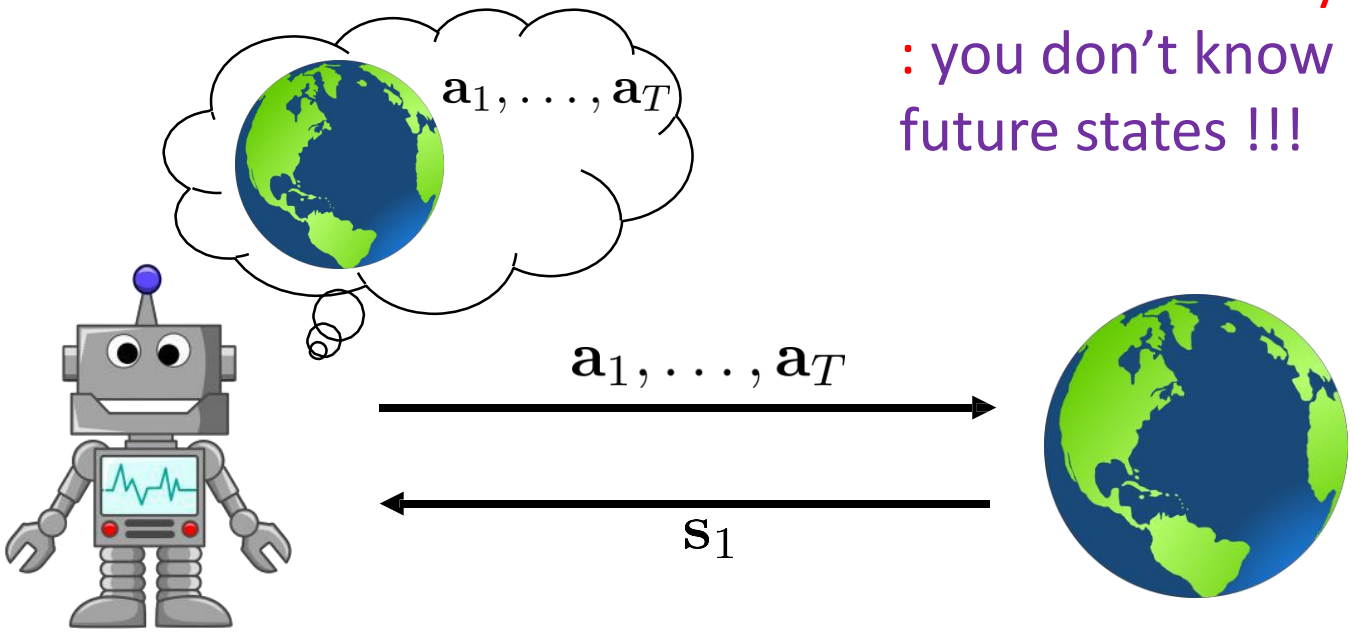
closed-loop

Take an action and see the next state and repeat !

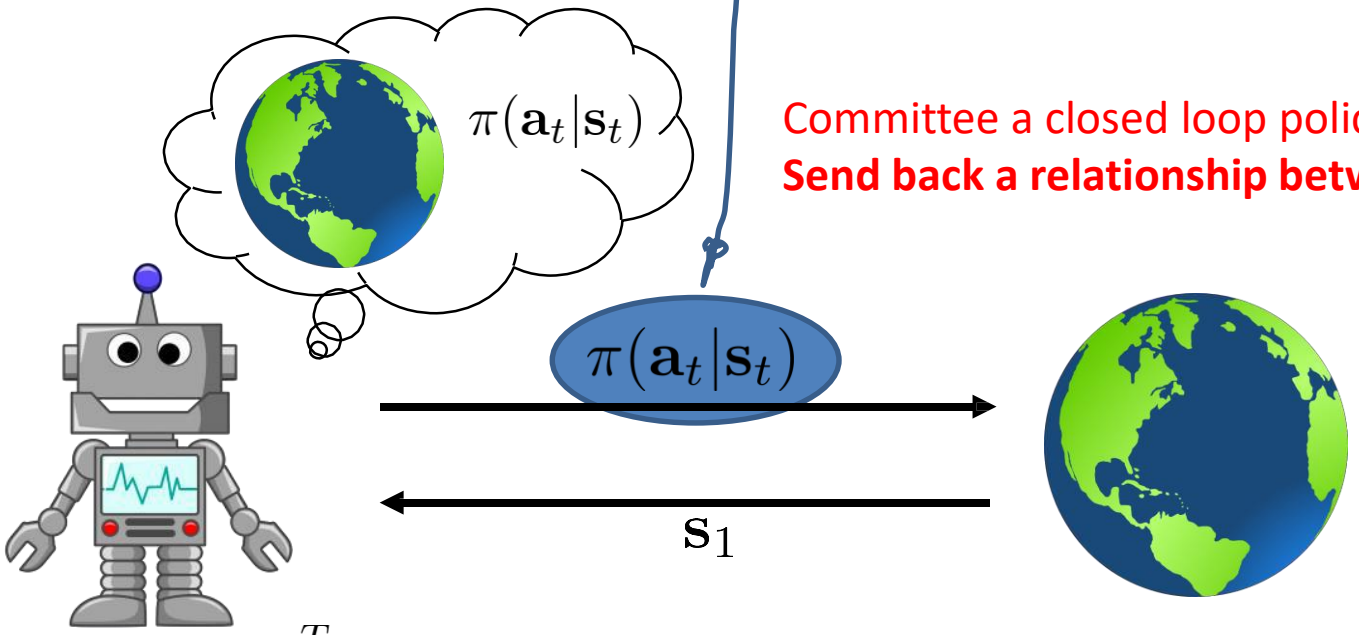


open-loop

only sent at t = 1, then it’s one-way! : you don’t know future states !!!



The stochastic closed-loop case = RL



Committee a closed loop policy

Send back a relationship between state and action

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

stochastic open-loop

form of π?

neural net

global

For all possible states !!!

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

stochastic closed-loop

time-varying linear

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

local

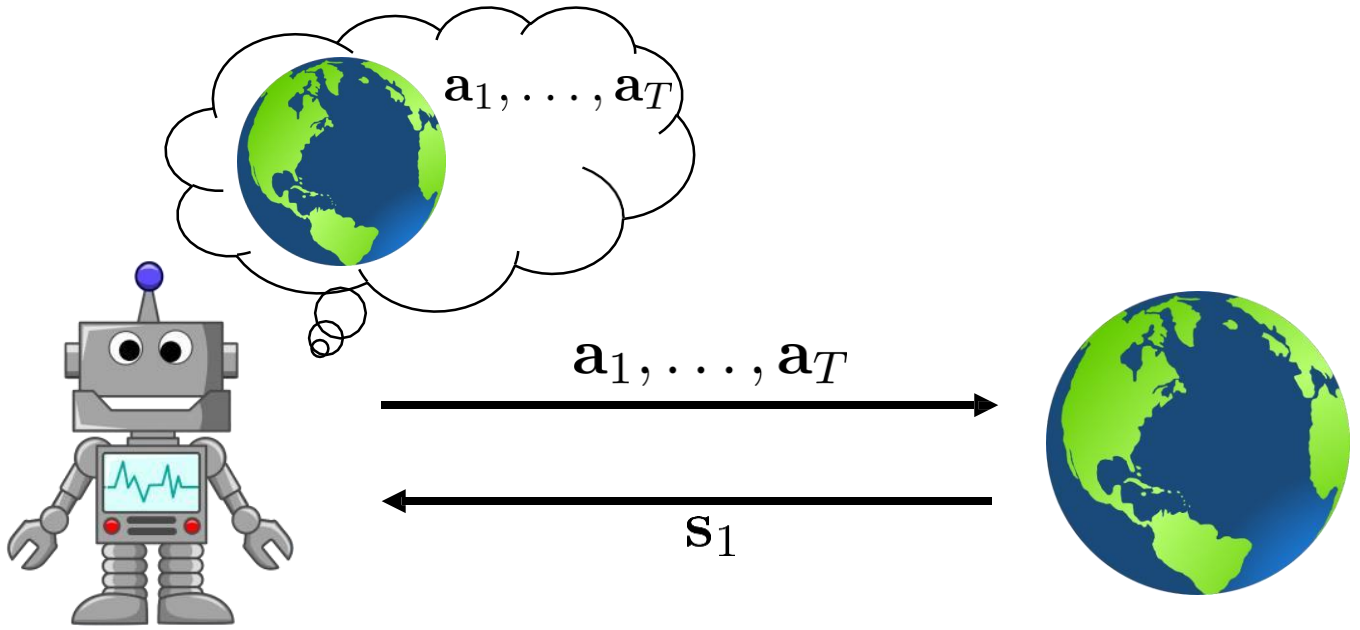
For local states around a nominal trajectory !!!

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

(more on this later)

Open-Loop Planning

But for now, open-loop planning



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

Stochastic optimization: (1) random shooting method

- arbitrary unconstrained optimization problem: Black box approach
- derivative free stochastic optimization
- Open loop planning

abstract away optimal control/planning:

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

$$\mathbf{A} = \arg\max_{\mathbf{A}} J(\mathbf{A})$$

$$\max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}) \approx \max\{Q(\mathbf{s}, \mathbf{a}_1), \dots, Q(\mathbf{s}, \mathbf{a}_N)\}$$

$(\mathbf{a}_1, \dots, \mathbf{a}_N)$ sampled from some distribution (e.g., uniform)

simplest method: guess & check

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

Algorithm: Consider N random action sequences (***Ai***) of length H (***Ai*** = (a1, a2, …aT)i), predict the result (i.e., future states) of taking each of these action sequences **using the learned dynamics model f_θ, evaluate the cost/reward (*J*)** associated with each candidate action sequence, and select the best action sequence.

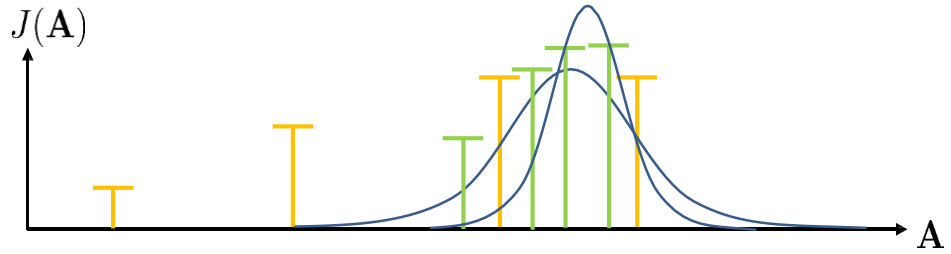
- Only plans H steps into the future:
  (1) desirable because it prevent accumulating model error,
  (2) limited because it may not be sufficient for solving long-horizon tasks.

Discussion:

- Very simple and efficient !!! parallelizable
- sometimes works very well for low dimensional action systems (& for short horizon) !!!
- **But depend on luck !!!**

(2) Cross-entropy method (CEM)

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



Algorithm: cross-entropy method with continuous-valued inputs:

1. sample $\mathbf{A}_1, \ldots, \mathbf{A}_N$ from $p(\mathbf{A})$ Initially from uniform distribution (random selection)
2. evaluate $J(\mathbf{A}_1), \ldots, J(\mathbf{A}_N)$
3. pick the *elites* $\mathbf{A}_{i_1}, \ldots, \mathbf{A}_{i_M}$ with the highest value, where $M < N$ M = (0.1 ~ 0.2)N, OK for N= 10 actions with 5 iteration steps, NG for more
4. refit $p(\mathbf{A})$ to the elites $\mathbf{A}_{i_1}, \ldots, \mathbf{A}_{i_M}$ Typically Gaussian distribution

Discussion:

- similar to random-shooting, but with iterative improvement of the distribution of actions that are sampled from
- good for low-to-mid dimensional space & for mid range horizon
- focus more on the good actions (with higher ***J***) : refit $p(\mathbf{A})$ and sample more from this region
- can find global optimum if large enough samples are taken
- see also: **(3) CMA-ES (Covariance Matrix Adaptation - Evolution Strategy)**
  (sort of like CEM with momentum ➔ so smaller no of samples)

What's the upside?

1. Very fast if parallelized
2. Extremely simple

What's the problem?

1. Very harsh dimensionality limit (**rule of thumb**: good for 30-60 dim,
10 dim w/15 time steps (150 dim) is still OK
because they are strongly correlated)
2. Only open-loop planning

Discrete case: Monte Carlo tree search (MCTS)

Closed loop feedback

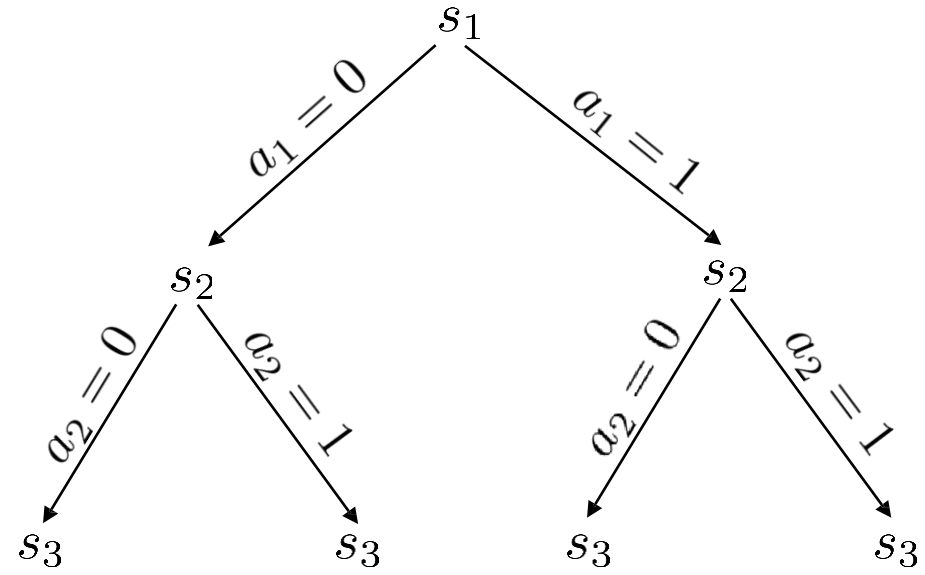
Good for Board game, AlphaGo, Poker (discrete stochastic setting)

s_t a_t

Sea Quest game: shooting torpedo (어뢰)

discrete planning as a search problem

Highly randomized



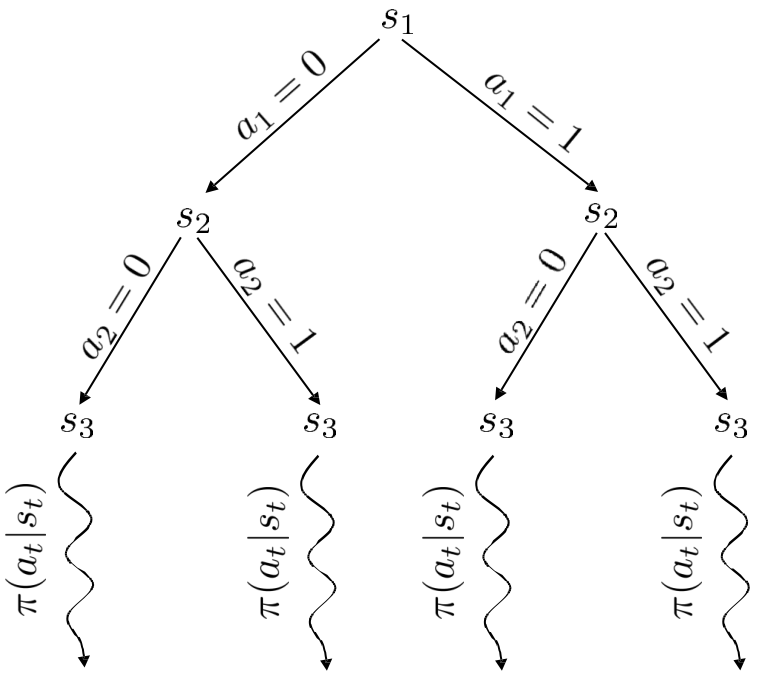
Exponentially increasing actions !

Discrete case: Monte Carlo tree search (MCTS)

s_t

a_t

how to approximate value without full tree?



Apply base policy, e.g., random policy

Algorithm:

1) Do tree search until depth = 3 (e.g. up to s3)
2) Approximate return with running a baseline policy (e.g. random policy with discount) **w/o full tree search**
   (this gives a reasonable idea of how good those states are: good state ➔ good value & bad state ➔ bad value)

Discussion:

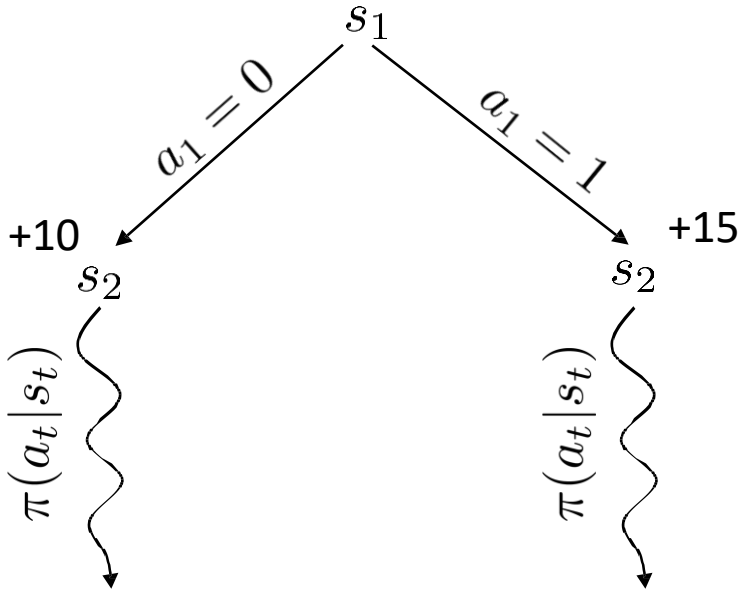
- Practically very good algorithm for discrete stochastic setting

Discrete case: Monte Carlo tree search (MCTS)

s_t

a_t

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



This total reward is obtained after a base policy.

This reward is **stochastic (sample-based estimate), i.e, may change for another use of action a_1 and use of random policy thereafter**

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

because stochastic

Discrete case: Monte Carlo tree search (MCTS)

generic MCTS sketch

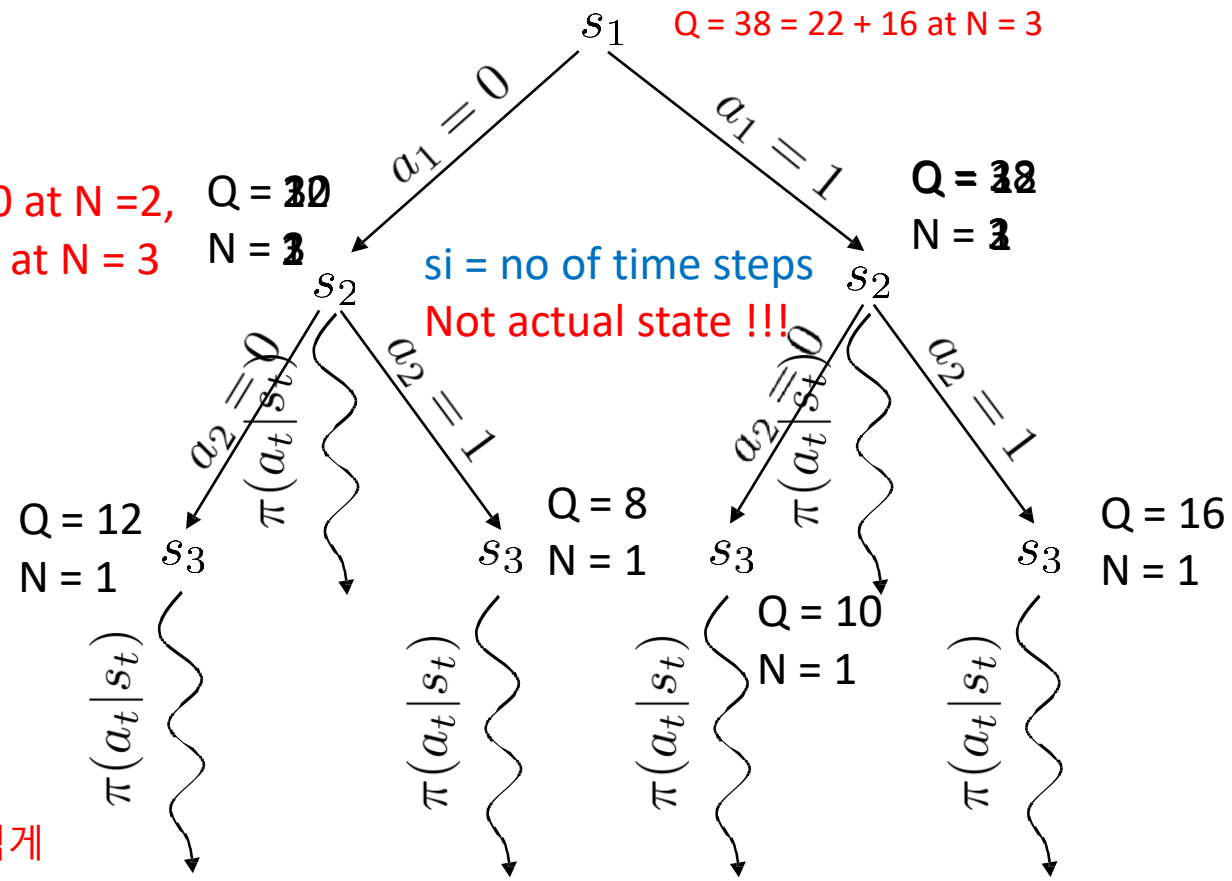
1. find a leaf s_l using TreePolicy(s_1) Stochastic !! ➔ different N (no. of visit to the state) ➔ different Q values !
2. evaluate the leaf using DefaultPolicy(s_l) e.g. random policy
3. update all values in tree between s_1 and s_l

   take best action from s_1

Iterate until computation resources are fully wasted

Q= 10 at N =1,
Q = 22 = 12 +10 at N =2,
Q = 30 = 22 + 8 at N = 3

Q= 12 at N =1,
Q = 22 = 12 +10 at N =2, Q-ave = 11
Q = 38 = 22 + 16 at N = 3



UCT (Upper Confidence Boundary of Tree)

UCT TreePolicy(s_t)

if s_t not fully expanded, choose new a_t

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

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

Average value Of node

Bonus for less visited node (더 적게 방문한 state의 action을 취하라 !!!)

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.

- MCTS is difficult to analyze
but works well in practice especially for chance game (like Poker),
- AlphaGo is a combination of RL

Case study: imitation learning from MCTS

Supervisor

Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning

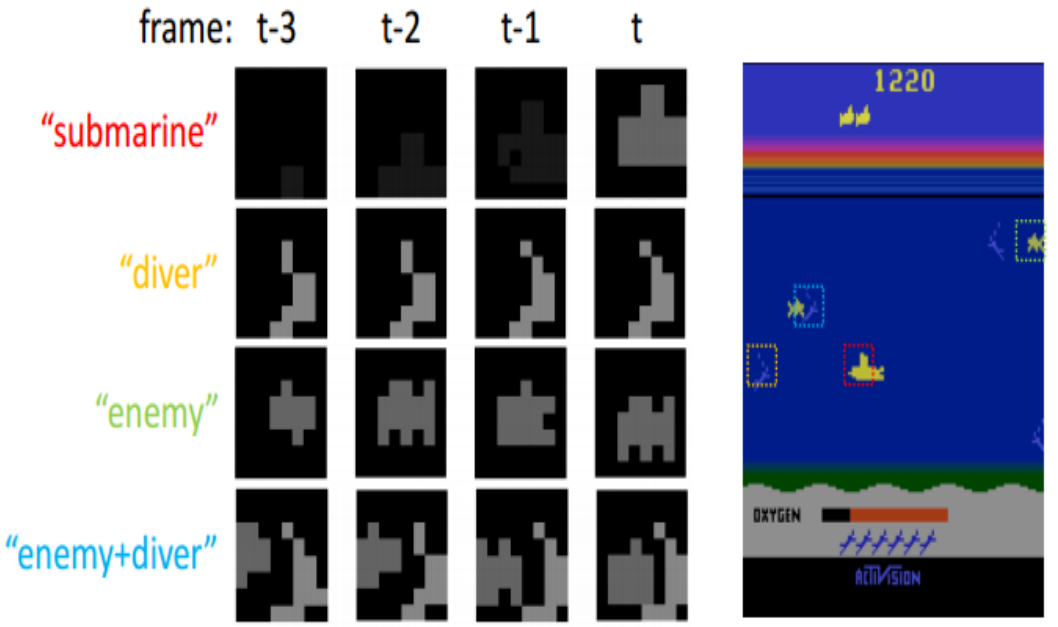
Xiaoxiao Guo
Computer Science and Eng.
University of Michigan
guoxiao@umich.edu

Satinder Singh
Computer Science and Eng.
University of Michigan
baveja@umich.edu

Honglak Lee
Computer Science and Eng.
University of Michigan
honglak@umich.edu

Richard Lewis
Department of Psychology
University of Michigan
rickl@umich.edu

Xiaoshi Wang
Computer Science and Eng
University of Michigan
xiaoshiw@umich.edu



Case study: imitation learning from MCTS

DAgger

1. train $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{u}_1, \ldots, \mathbf{o}_N, \mathbf{u}_N\}$
2. run $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \ldots, \mathbf{o}_M\}$
3. Ask human to label $\mathcal{D}_\pi$ with actions $\mathbf{u}_t$
3. Choose actions for states in $\mathcal{D}_\pi$ using MCTS
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

i.e. human ➔ MCTS

Why train a policy?

- In this case, MCTS is too slow for real-time play
- Other reasons – perception, generalization, etc.: more on this later

Trajectory Optimization with Derivatives

Can we use derivatives?

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

Constrained minimization problem w/ equality constraint

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

Unconstrained minimization problem
because constraints are put into the cost function

usual story: differentiate via backpropagation and optimize!

need $\frac{df}{d\mathbf{x}_t}, \frac{df}{d\mathbf{u}_t}, \frac{dc}{d\mathbf{x}_t}, \frac{dc}{d\mathbf{u}_t}$ Derivatives of function *f* and cost *c*

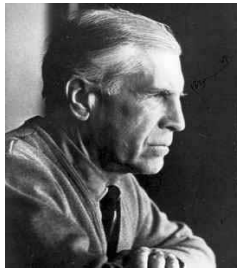
in practice, it really helps to use a 2nd order method!

1st order derivative method works poorly because of **vanishing/explosion**
➔ 2nd order derivative method like Newton's method

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

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

Richard Bellman



Lev Pontryagin

Shooting methods vs collocation

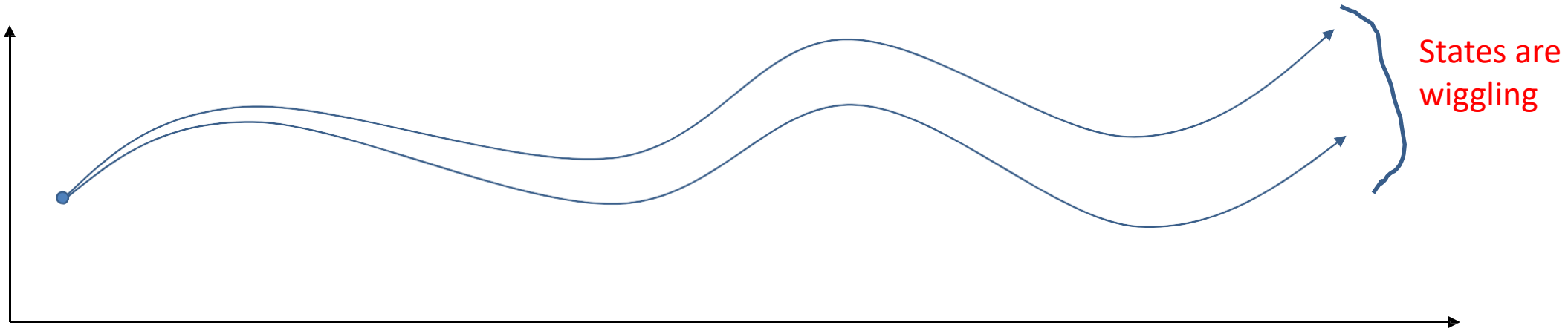
shooting method: **optimize over actions only**

because we know states from transition dynamics

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

Initial action contribute more on the future states (last action contribute little) !!!

➔ **Ill-conditioning** of 1st order numerical method: condition no. of Hessian matrix is very poor



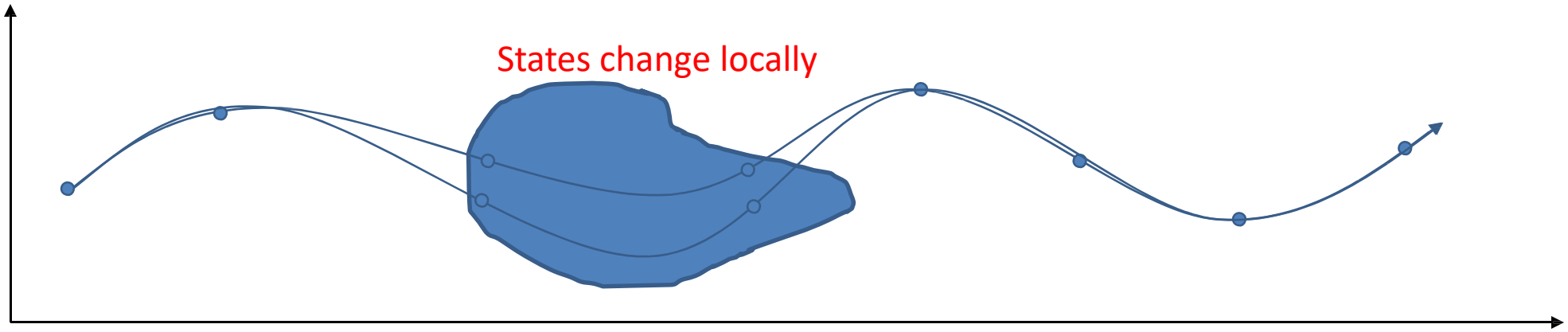
Shooting methods vs collocation

collocation method: **optimize over actions and states**, with constraints

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

Initial action contribute more on the future states **but constraints can control over the actions**

➔ States change locally ➔ **Well-conditioning** of 1st order numerical method



Linear case: LQR (Linear Quadratic Regulator)

: Shooting method w/2nd order method

Linear(ized) transition dynamics + Quadratic cost function

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

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

linear

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

quadratic

Deterministic LQR ! ➔ later stochastic LGQ

Linear cost case ➔ solution is not useful!!!

Regulating trajectory (s_t, a_t)!!!

Note that the linear dynamics and quadratic cost may be changing at each time step ➔ nonlinear case !!!

Linear case: LQR

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

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

only term that depends on $\mathbf{u}_T$

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

Backward recursion

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

Because last action $\mathbf{u}_T$ won't affect the cost (reward) of previous times!!!

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

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

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

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

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

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

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

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

Linear case: LQR

Q(x,u), V(x) vs Q(s,a), V(s) in RL !!!

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

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

State & action value function !

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

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

State value function !

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

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

Quadratic !!!

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

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

Linear case: LQR

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

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

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

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

$Q(\mathbf{x}_T, \mathbf{u}_T)$

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

Insert XT into V(XT) to get V(XT-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

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

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

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

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

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

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

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

Linear case: LQR

Backward recursion

for $t = T$ to 1:

$$\mathbf{Q}_t = \mathbf{C}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{F}_t$$

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

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

$$\mathbf{u}_t \leftarrow \arg\min_{\mathbf{u}_t} Q(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$$

Action ← minimization of Q : Grad(Q) = 0

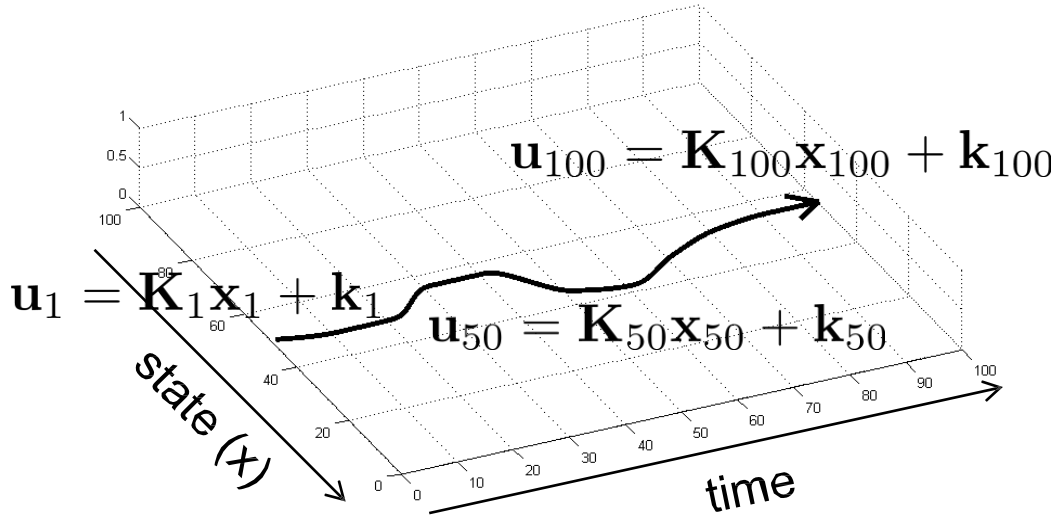
$$\mathbf{K}_t = -\mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t}^{-1} \mathbf{Q}_{\mathbf{u}_t, \mathbf{x}_t}$$

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

$$\mathbf{V}_t = \mathbf{Q}_{\mathbf{x}_t, \mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t, \mathbf{u}_t} \mathbf{K}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{x}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t} \mathbf{K}_t$$

$$\mathbf{v}_t = \mathbf{q}_{\mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t, \mathbf{u}_t} \mathbf{k}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t} \mathbf{k}_t$$

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



we know $\mathbf{x}_1$!

Forward recursion

For t = 1 to T

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

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$$

Zip to the end state (forward recursion)
and Unzip to the start state (backward recursion)

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 $\mathbf{x}_t$

Follow the optimal policy (optimal action *u)*

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 for Stochastic and Nonlinear Systems

Stochastic dynamics: **LQG** (Linear Quadratic Gaussian)

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

control gain by LQ optimal control & **state estimation by Kalman filter**
== **Linear Quadratic Gaussian(LQG)**. == Stochastic optimal control

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

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

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

Fixed constant covariance

➔ Optimal **closed loop** policy in the linear Gaussian case !!!

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

Optimal closed form control law (policy)
with control gain by LQ optimal control
LQR does produce closed-loop plans NOT open-loop plans

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)

- Adding Gaussian noise does not change the linear action $\boldsymbol{u_t}$ because of the symmetry of Gaussian distribution
- However, the visited states are different due to stochasticity
- Not getting a single sequence of states and actions
- Interesting to note that LQR does produce closed-loop plans NOT open-loop plans

In "http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-10.pdf", the instructor says that LQR works in stochastic dynamics in the same way as it works in deterministic dynamics, and mentions that the expectation of a quadratic under a Gaussian has an analytic solution.

What's the analytic result of the following expectation?

$$E_{X\sim\mathcal{N}(\mu,\Sigma)}\left[X^TAX+X^TB+C\right]$$

$$X^TAX+X^TB+C=\sum_{i,j}a_{ij}X_iX_j+\sum_i b_iX_i+C$$

Therefore

$$\begin{aligned}\mathbb{E}\left[X^TAX+X^TB+C\right]&=\mathbb{E}\left[\sum_{i,j}a_{ij}X_iX_j+\sum_i b_iX_i+C\right]\\&=\sum_{i,j}a_{ij}\mathbb{E}[X_iX_j]+\sum_i b_i\mathbb{E}[X_i]+C\end{aligned}$$

Here

$$\mathbb{E}[X_iX_j]=\sigma_{ij}+\mu_i\mu_j,\quad \mathbb{E}[X_i]=\mu_i.$$

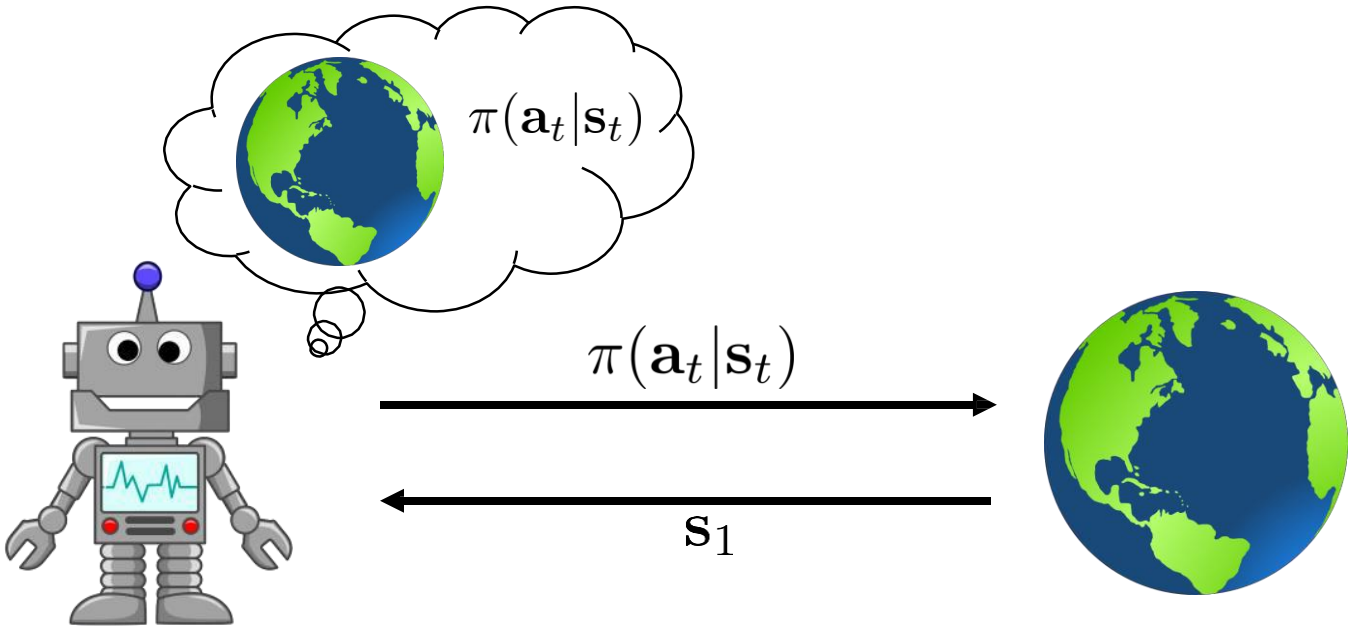
Finally,

$$\mathbb{E}\left[X^TAX+X^TB+C\right]=\sum_{i,j}a_{ij}\sigma_{ij}+\mu^TA\mu+\mu^TB+C$$

As Did noted in comment below, the answer can be rewritten as follows:

$$\mathbb{E}\left[X^TAX+X^TB+C\right]=\mathrm{tr}(A\Sigma)+\mu^TA\mu+\mu^TB+C.$$

The stochastic closed-loop case



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

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

form of π?

time-varying linear

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

Alternative of global NN policy !!!

Nonlinear case: DDP (Differential Dynamic Programming) /iterative LQR

Linear-quadratic assumptions:

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

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

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

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

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

$$\bar{f}(\delta\mathbf{x}_t, \delta\mathbf{u}_t) = \underbrace{\mathbf{F}_t}_{\nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)} \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix}$$

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

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

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

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)

Iterative !!!

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? ➔ Similar to Newton's method

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

until convergence:

$$\mathbf{g} = \nabla_{\mathbf{x}} g(\hat{\mathbf{x}})$$

$$\mathbf{H} = \nabla^2_{\mathbf{x}} g(\hat{\mathbf{x}})$$

Quadratic approximation of g(x) at x_hat

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

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

$C_t = H$; $c_t = g$

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

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

x = u;

g = c

Nonlinear case: DDP/iterative LQR

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

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

The main way in which the iLQR differs from the Newton's method is that it does'nt consider the 2nd derivatives of the dynamics

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

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

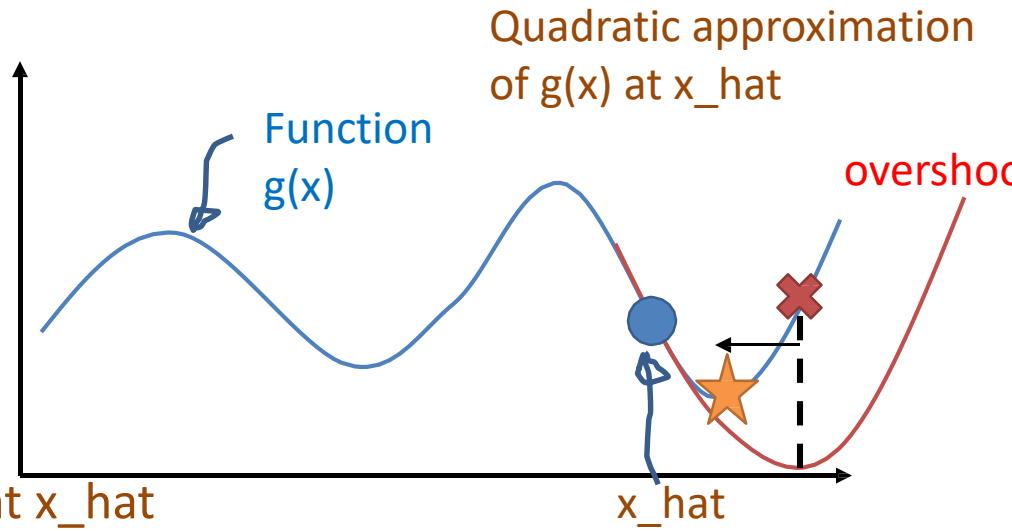
differential dynamic programming (DDP)

Nonlinear case: DDP/iterative LQR

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

Positive definite

why is this a bad idea? For very complicated nonlinear function



Current function value at x_hat

True local minimum near x_hat

Overshoot value due to approximated minimum of the quadratic approximation

search over α from 1 to 0 until improvement achieved

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 $\mathbf{u}_t = \mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t$

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

Case Study and Additional Readings

CS 285: Lecture 10, Part 5 - YouTube

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 $\mathbf{u}_t, \ldots, \mathbf{u}_T$ to minimize $\sum_{t'=t}^{t+T} c(\mathbf{x}_{t'}, \mathbf{u}_{t'})$

execute action $\mathbf{u}_t$, discard $\mathbf{u}_{t+1}, \ldots, \mathbf{u}_{t+T}$

➔ MPC (Model Predictive Control)

Re-plan at each time step

Video **Acrobat, Snake, Hopper, Humanoid**

Synthesis of Complex Behaviours with Online Trajectory Optimization

Yuval Tassa, Tom Erez & Emo Todorov

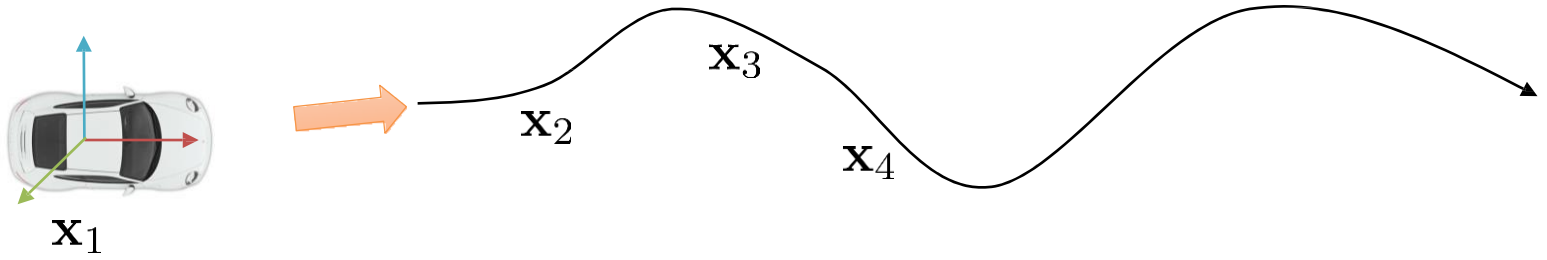
IEEE International Conference
on Intelligent Robots and Systems
2012

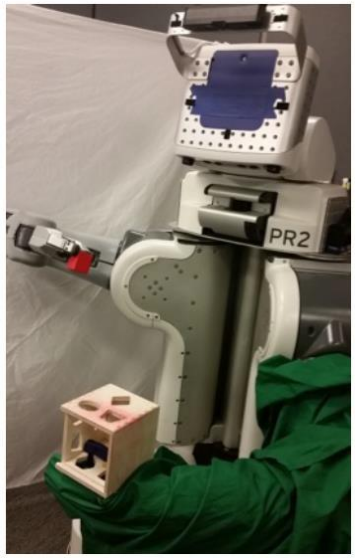
Real-time control, robust against perturbations (disturbances) with perfect system dynamics model (but imperfect model (like incorrect mass) ➔ NG

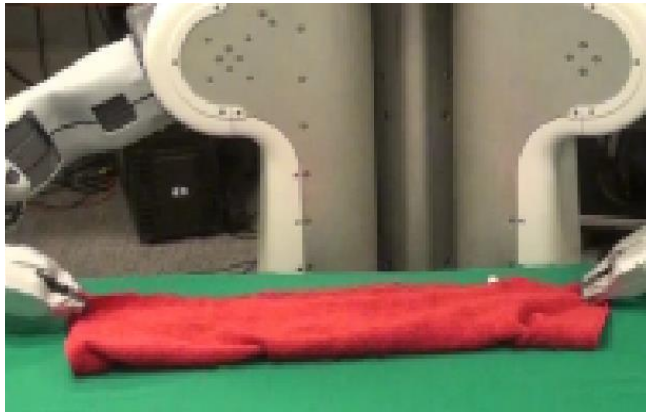
Additional reading

1. Mayne, Jacobson. (1970). Differential dynamic programming.
   - Original differential dynamic programming algorithm.
2. Tassa, Erez, **Todorov.** (2012). Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization.
   - Practical guide for implementing non-linear iterative LQR.
3. Levine, Abbeel. (2014). Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics.
   - Probabilistic formulation and trust region alternative to deterministic line search for LQR.

What's wrong with known dynamics?







Next time: learning the dynamics model for systems that cannot be modeled easily