Trustworthy AI Autonomy

Lecture 8: Model-based decision making

Ding Zhao

Assistant Professor

Carnegie Mellon University

Carnegie Mellon University

Safe AI Lab @CMU

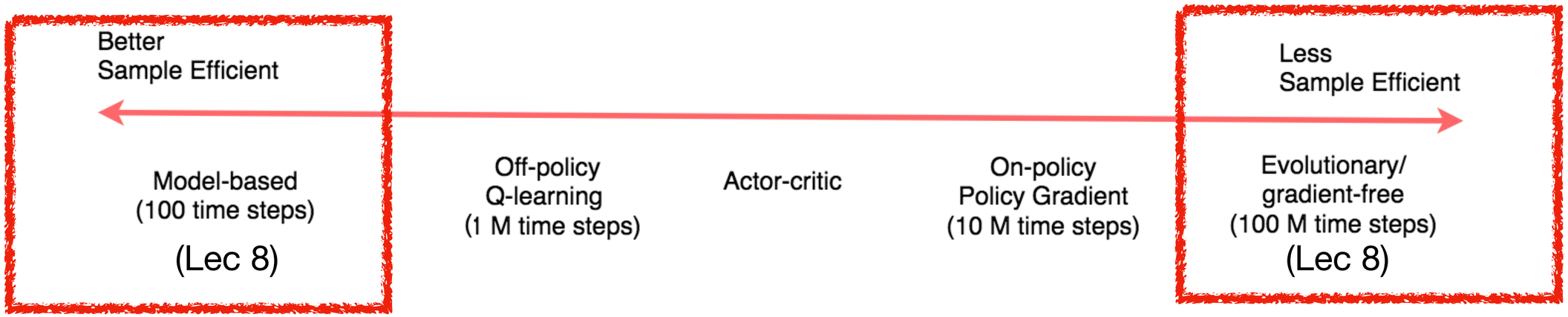
Plan for today

- Model-based control
  - LQR, iLQR, MPC
- Model-based reinforcement learning
  - Neural network based method
  - Local (linearized) model
  - Planning: Cross Entropy Method
  - Gaussian processed-based Reinforcement learning (next lecture)

Recap: On-policy vs off-policy

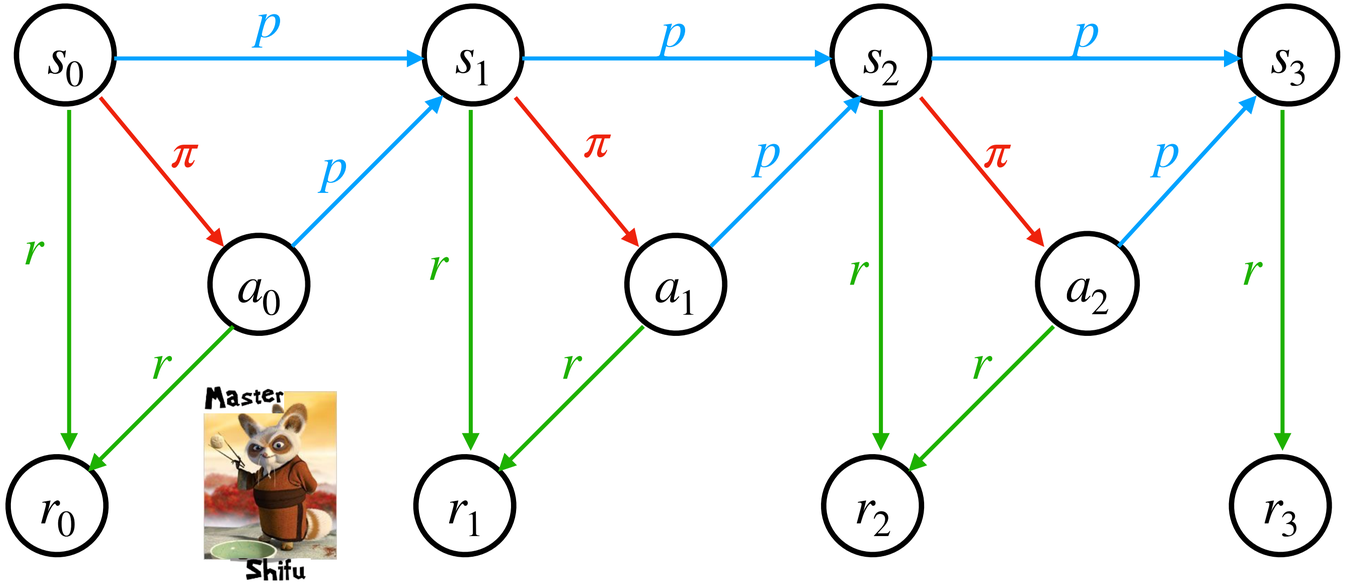
- Policy optimization is almost always performed **on-policy**, which means that each update only uses data collected while acting according to the most recent version of the **policy**. The historical data collected with very old policy is not used. They can be used with both continuous and discrete states. Using gradient, they converge to a local minima of $J(\theta)$

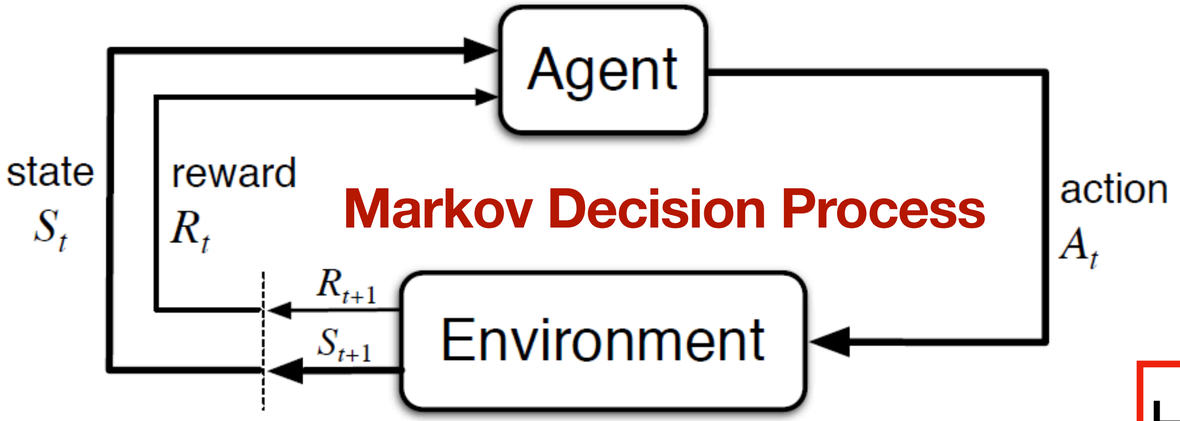
- Q-learning, e.g., DQN, is almost always performed off-policy, which means that each update can use data collected during the whole training history, regardless of what policy the agent was choosing to explore the environment. Therefore, it is more sampling efficient. No guarantee of convergence.



https://jonathan-hui.medium.com/rl-model-based-reinforcement-learning-3c2b6f0aa323

Recap: MDP/Reinforcement Learning





- Instead of asking for demos, we only request a single digit number r_t to indicate the level of happiness - reward.

$$s_{t+1} \sim p\left(\cdot \mid s_t, a_t\right)$$

$$a_t \sim \pi\left(\cdot \mid s_t\right)$$

$$r_t \sim r\left(\cdot \mid s_t, a_t\right)$$

Here $p(s_{t+1} \mid s_t, a_t)$ is called the ***model***

How to get the model?

- **Often we do know the dynamics**
  - Well-studied systems, e.g., automotive
  - Optimal control

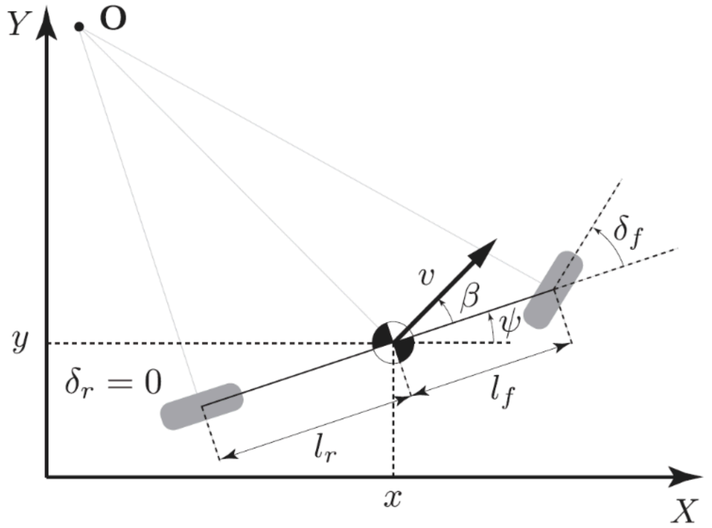


Figure 1: Bicycle model[2]

$$\dot{x} = f(x, u, t) \qquad \dot{x} = Ax + Bu$$

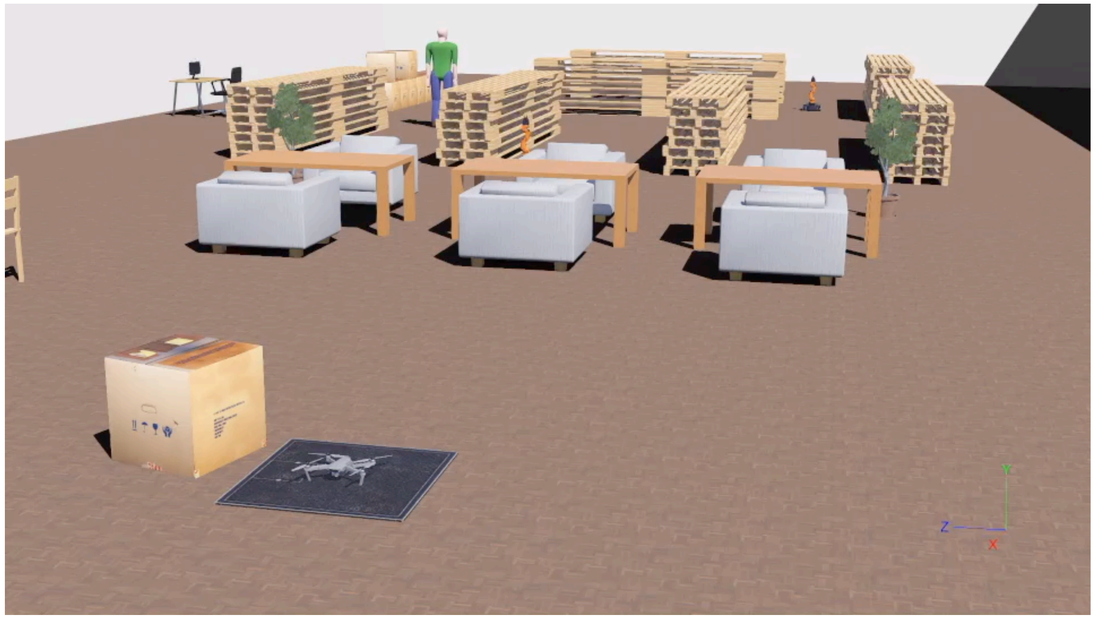
$$y = g(x, u, t) \qquad y = Cx + Du$$

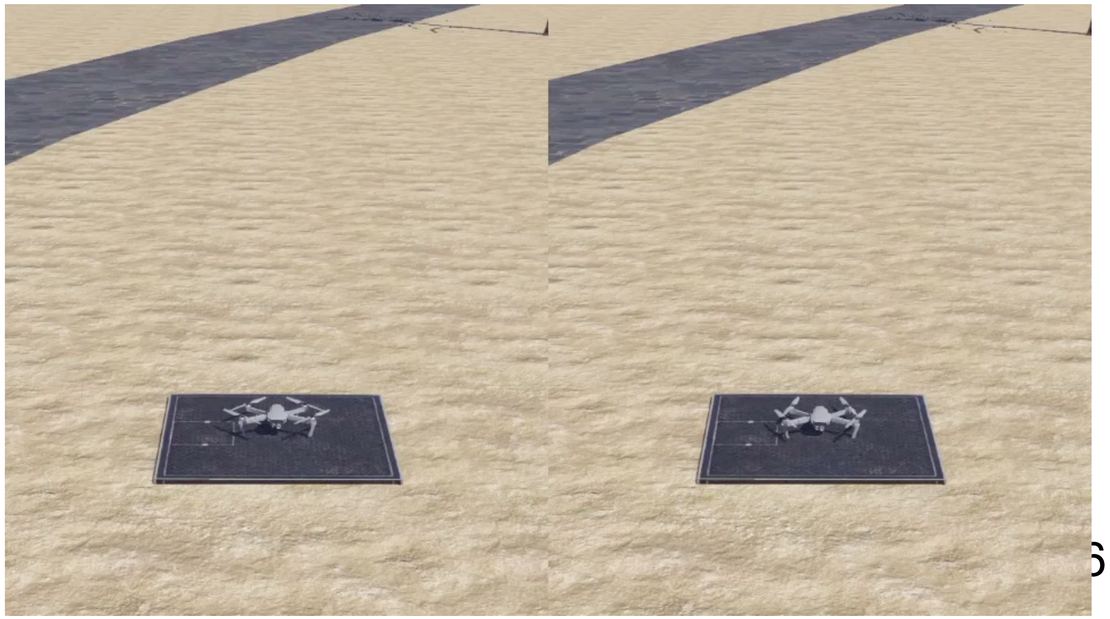
$$\frac{d}{dt}s_1 = \frac{d}{dt}\begin{bmatrix} y \\ \dot{y} \\ \psi \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-4C_\alpha}{m\dot{x}} & 0 & -\dot{x} + \frac{2C_\alpha(l_r - l_f)}{m\dot{x}} \\ 0 & 0 & 0 & 1 \\ 0 & \frac{2(l_r - l_f)C_\alpha}{I_z\dot{x}} & 0 & -\frac{2\left(l_f^2 + l_r^2\right)C_\alpha}{I_z\dot{x}} \end{bmatrix} \begin{bmatrix} y \\ \dot{y} \\ \psi \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \frac{2C_\alpha}{m} & 0 \\ 0 & 0 \\ \frac{2l_fC_\alpha}{I_z} & 0 \end{bmatrix} \begin{bmatrix} \delta \\ F \end{bmatrix}$$

$$\frac{d}{dt}s_2 = \frac{d}{dt}\begin{bmatrix} x \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & \frac{1}{m} \end{bmatrix} \begin{bmatrix} \delta \\ F \end{bmatrix} + \begin{bmatrix} 0 \\ \dot{\psi}\dot{y} - fg \end{bmatrix}$$

Where to get the model?

- **Often we do know the dynamics**
  - Well-studied systems, e.g., automotive
  - Optimal control
- **We know the structure of the dynamics but need to fit some parameters**
  - System identification: fit unknown parameters of a known model structure, e.g., estimation of the road friction, abrupt changes
  - Adaptive control: the model may not be accurately estimated but the control error vanishes





Adaptive control (MRAC) vs Optimal Control (baseline)

$$\ddot{x} = (\cos\phi \sin\theta \cos\psi + \sin\phi \sin\psi)\frac{U_1}{m}$$

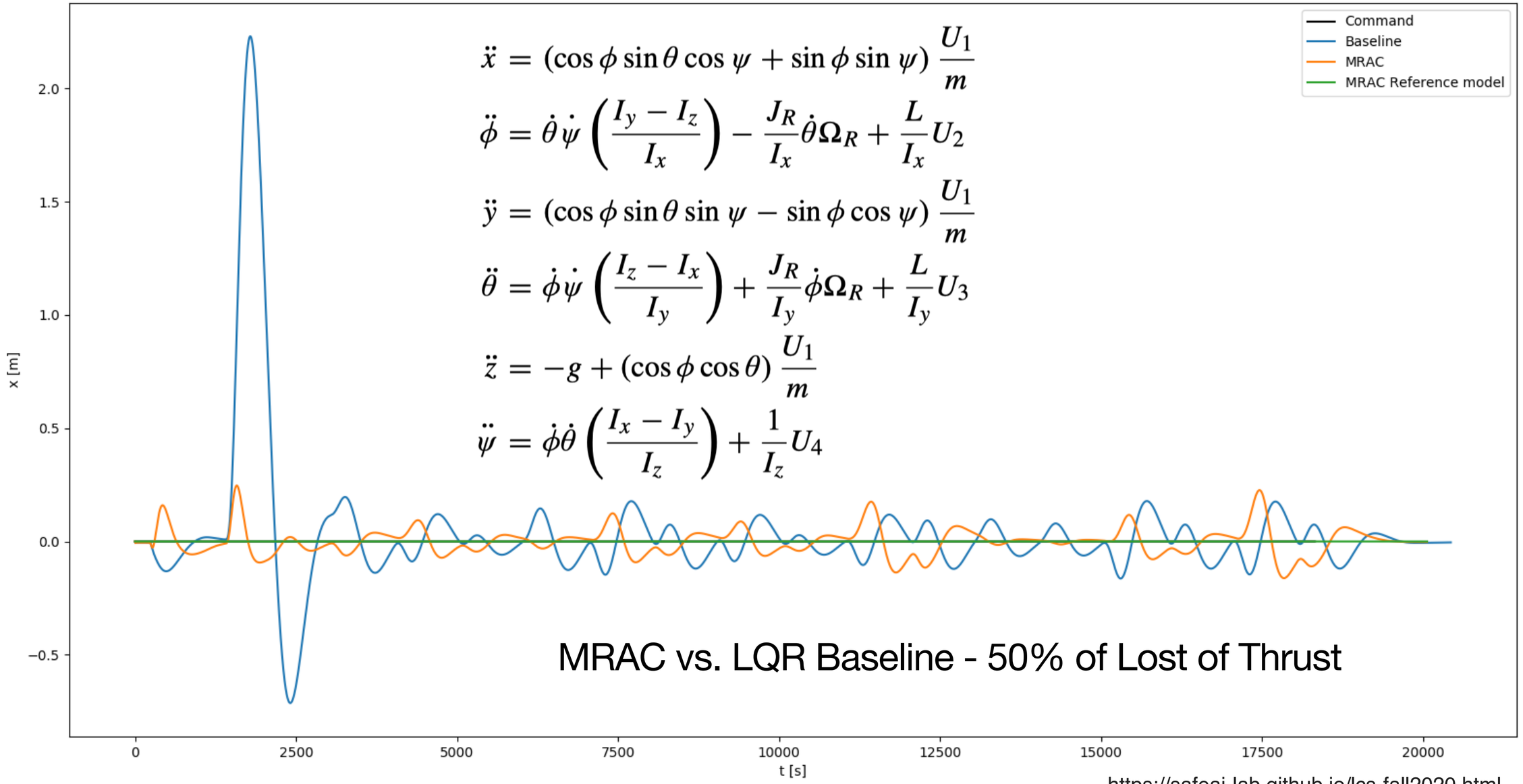
$$\ddot{\phi} = \dot{\theta}\dot{\psi}\left(\frac{I_y - I_z}{I_x}\right) - \frac{J_R}{I_x}\dot{\theta}\Omega_R + \frac{L}{I_x}U_2$$

$$\ddot{y} = (\cos\phi \sin\theta \sin\psi - \sin\phi \cos\psi)\frac{U_1}{m}$$

$$\ddot{\theta} = \dot{\phi}\dot{\psi}\left(\frac{I_z - I_x}{I_y}\right) + \frac{J_R}{I_y}\dot{\phi}\Omega_R + \frac{L}{I_y}U_3$$

$$\ddot{z} = -g + (\cos\phi \cos\theta)\frac{U_1}{m}$$

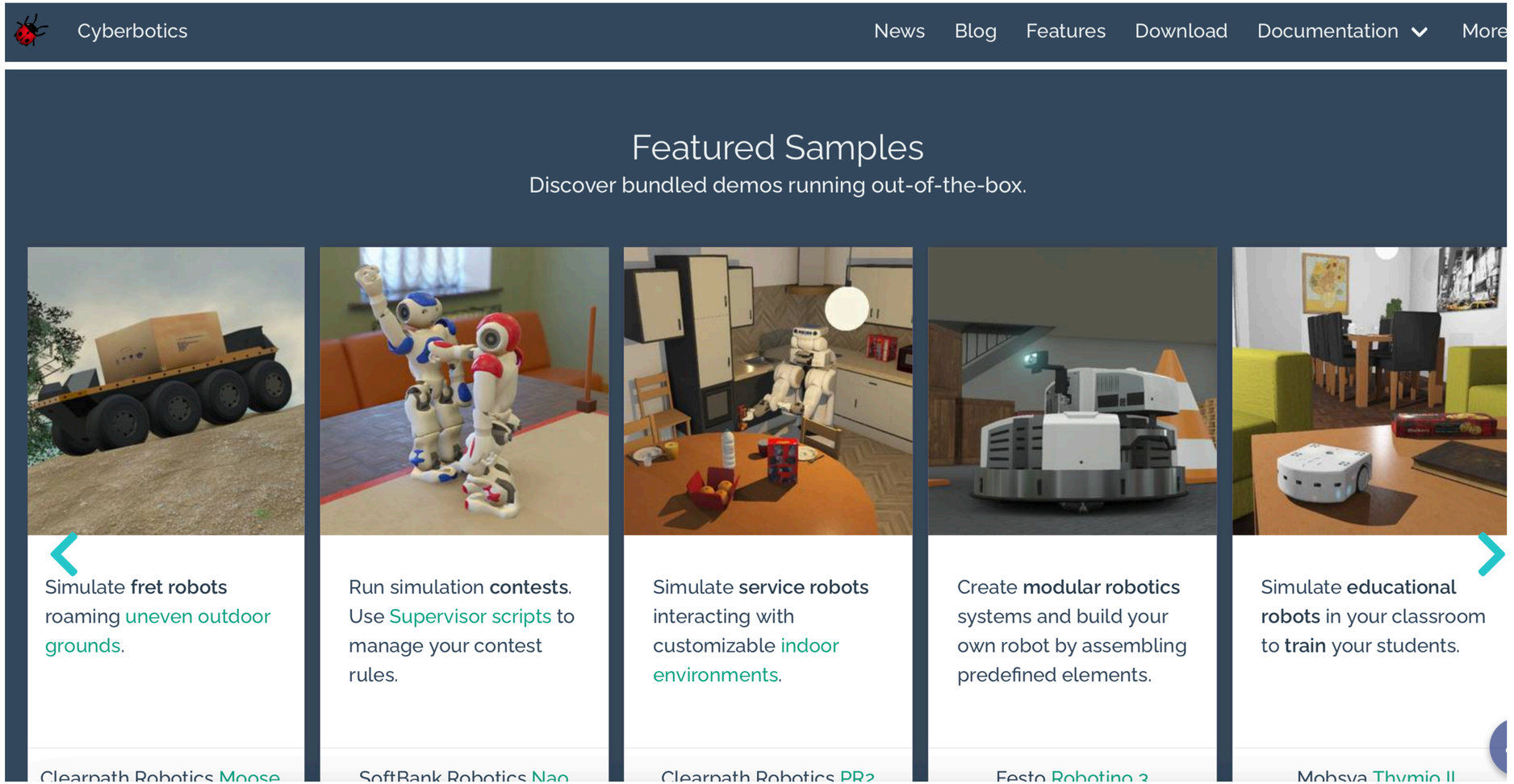
$$\ddot{\psi} = \dot{\phi}\dot{\theta}\left(\frac{I_x - I_y}{I_z}\right) + \frac{1}{I_z}U_4$$



MRAC vs. LQR Baseline - 50% of Lost of Thrust

https://safeai-lab.github.io/lcs-fall2020.html

Webots platform



https://www.cyberbotics.com

Where to get the model?

- **We do know the dynamics**
  - Well-studied systems, e.g., automotive
  - Optimal control
- **We know the structure of the dynamics but need to fit some parameters**
  - System identification – fit unknown parameters of a known model, e.g. estimation of the road friction, abrupt changes
  - Adaptive control: the model may not be accurate but the control error vanishes
- **We can learn the dynamics**
  - Fit a general-purpose model to observed transition data, e.g. a comfortable driving distance with surrounding cars
  - Model-based reinforcement learning: similar to adaptive control, the method might focus on the reward rather than the model accuracy

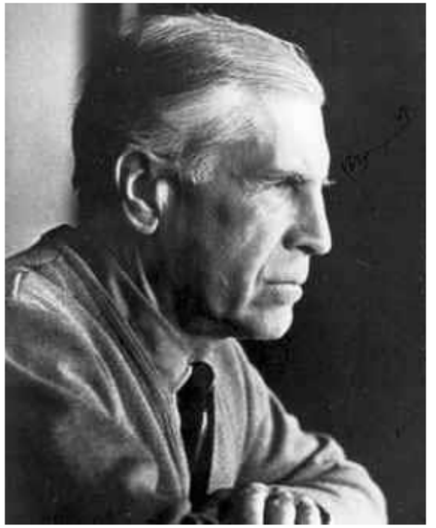
Aside: notation

$\mathbf{s}_t$ – state
$\mathbf{a}_t$ – action
$r(\mathbf{s}, \mathbf{a})$ – reward function

$\mathbf{x}_t$ – state
$\mathbf{u}_t$ – action
$c(\mathbf{x}, \mathbf{u})$ – cost function

$$r(\mathbf{s}, \mathbf{a}) = -c(\mathbf{x}, \mathbf{u})$$

Richard Bellman



Lev Pontryagin

Finite Horizon Discrete-Time Linear Quadratic Regulator

- Design control policy to minimize the cost function.

$$J_{0,N} = \frac{1}{2}x(N)^T S x(N) + \frac{1}{2}\sum_{k=0}^{N-1}\left(x(k)^T Q x(k) + u(k)^T R u(k)\right)$$

where $S, Q, R \geq 0$, subject to the system dynamics

$$x(k+1) = Ax(k) + Bu(k)$$

- It is found that the optimal control solution follows an elegant format

$$u^*(k) = K_k x(k) \qquad \min J_{k,N} = J^*_{k,N} = \frac{1}{2}x(k)^T S_k x(k)$$

- where K_k is a constant only dependent on A, B, S, Q, R,
  $S_N = S \Rightarrow K_{N-1} \Rightarrow S_{N-1} \Rightarrow K_{N-2} \Rightarrow S(N-2) \Rightarrow \ldots \Rightarrow S(0)(= J^*_{0,N})$

$$K_k = -(R + B^T S_{k+1} B)^{-1} B^T S_{k+1} A, S_k = (A + BK_k)^T S_{k+1}(A + BK_k) + Q + K_k^T R K_k$$

Infinite Horizon Discrete-Time Linear Quadratic Regulator

- Three ingredients to make IH-DT-LQR much computationally lighter compared to FH-DT-LQR:
  - $N \to \infty$, constant K, Discrete-time Algebraic Riccati Equation (DARE)

$$\min_{K} J = \sum_{k=0}^{\infty} x(k)^T Q x(k) + u(k)^T R u(k)$$

$$\text{with} \begin{cases} x(k+1) = Ax(k) + Bu(k) \\ u(k) = Kx(k) \end{cases}$$

Optimal solution:

$$K^* = -(R + B^T S B)^{-1} B^T S A$$

Where S can be solved by a DARE

$$S = A^T S A - A^T S B (R + B^T S B)^{-1} B^T S A + Q$$

Example

$$x(k+1) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(k)$$

$$y(k) = [1 \quad 0] x(k)$$

Let $Q = C^T C = I, R = 0.3$. Infinite horizon. Solve the optimal control.

Solve DARE

$$S = A^T S A - A^T S B (R + B^T S B)^{-1} B^T S A + Q,\ K = -(R + B^T S B)^{-1} B^T S A$$

$$\Rightarrow S = \begin{bmatrix} 2.751.91 \\ 1.913.34 \end{bmatrix}$$

$$K = [-0.524, -1.44]$$

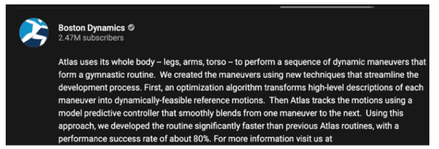
```python
from __future__ import division, print_function
import numpy as np
import scipy.linalg
def dlqr(A,B,Q,R):
    """Solve the discrete time lqr controller.
    x[k+1] = A x[k] +S B u[k]
    cost = sum x[k].T*Q*x[k] + u[k].T*R*u[k]
    """
    #ref Bertsekas, p.151
    #first, try to solve the ricatti equation
    S = np.matrix(scipy.linalg.solve_discrete_are(A, B, Q, R))
    #compute the LQR gain
    K = -np.matrix(scipy.linalg.inv(B.T*S*B+R)*(B.T*S*A))
    eigVals, eigVecs = scipy.linalg.eig(A+B*K)
    return K, S, eigVals
```

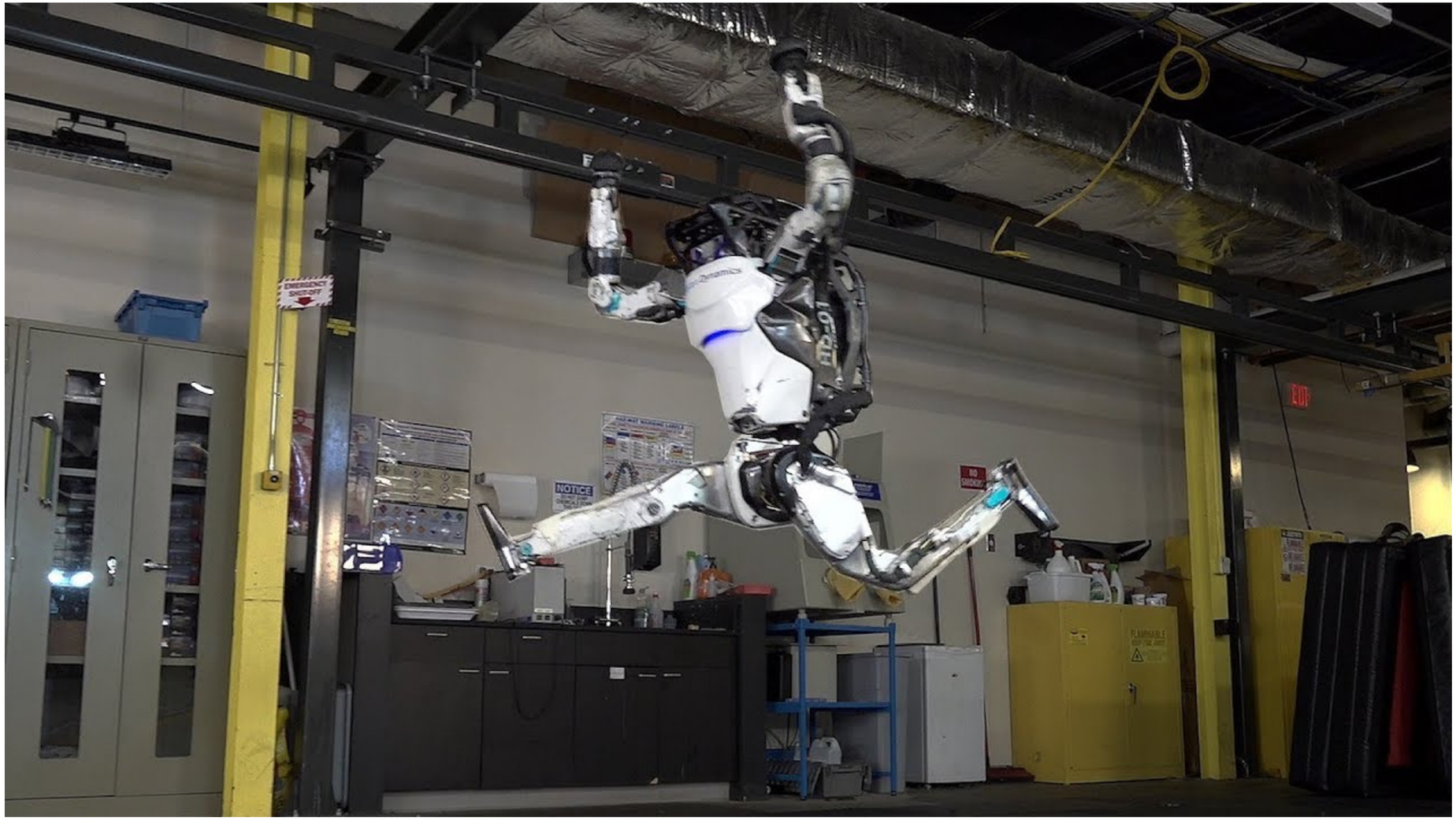
```python
A = np.array([[1,1],[0,1]])
B = np.array([[0],[1]])
Q = np.eye(2)
R = 0.3

K,S,_ = dlqr(A,B,Q,R)
print("S:", S)
print("K:", K)
```

```
S: [[2.75078485 1.90801622]
 [1.90801622 3.34052588]]
K: [[-0.52410456 -1.44169888]]
```

Model Predictive Control

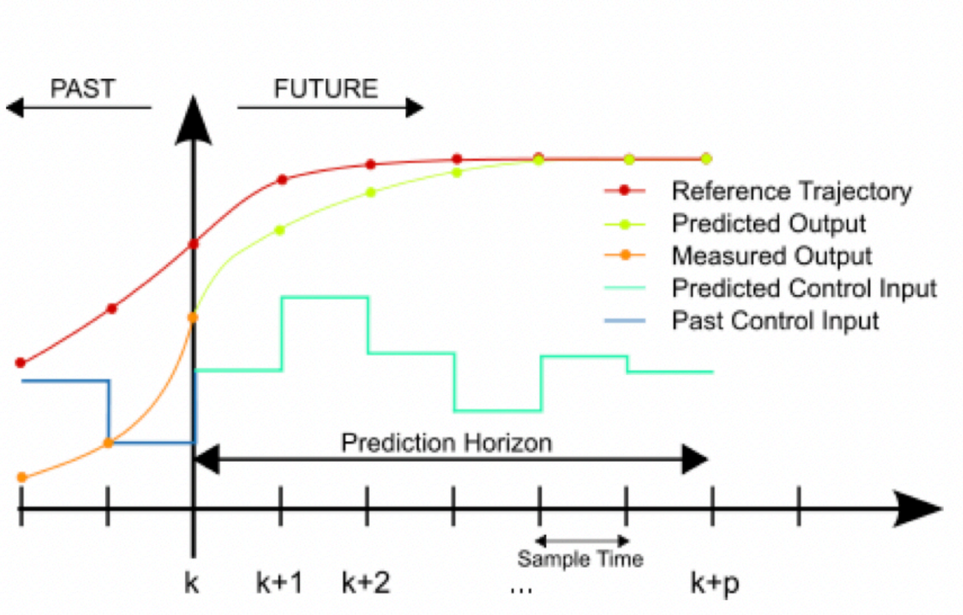




Atlas uses its whole body -- legs, arms, torso -- to perform a sequence of dynamic maneuvers that form a gymnastic routine. We created the maneuvers using new techniques that streamline the development process. First, an optimization algorithm transforms high-level descriptions of each maneuver into dynamically-feasible reference motions. Then Atlas tracks the motions using a model predictive controller that smoothly blends from one maneuver to the next. Using this approach, we developed the routine significantly faster than previous Atlas routines, with a performance success rate of about 80%.

https://www.youtube.com/watch?v=_sBBaNYex3E

FH-DT-LQR, Constrained LQR, & MPC



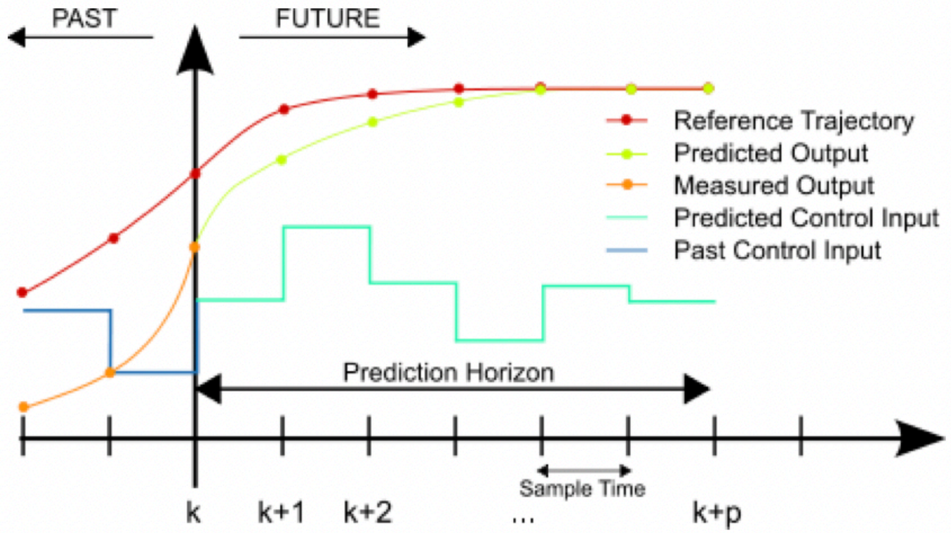
- **Finite-Horizon Discrete-Time Linear Quadratic Regulator**
- Design control policy to minimize the cost function

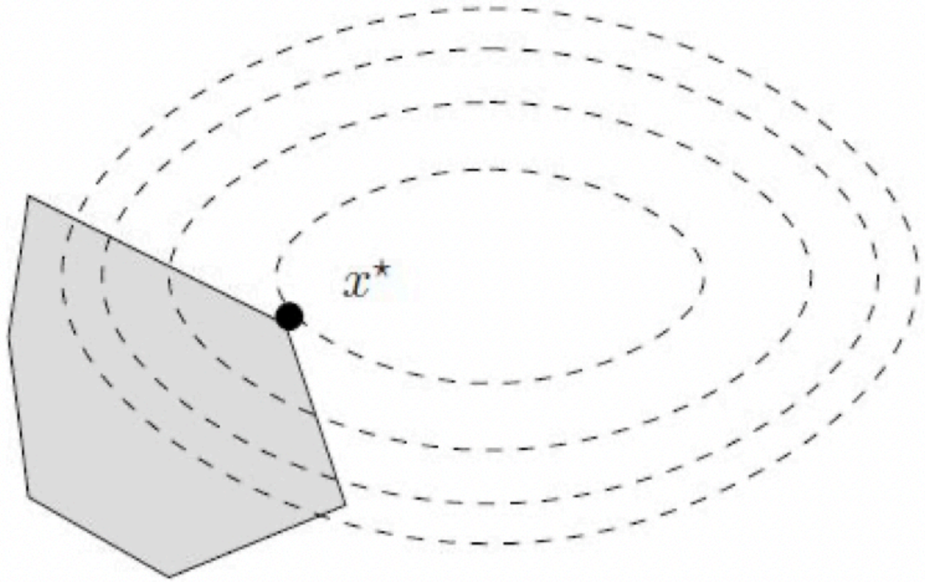
$$J_{0,N} = \frac{1}{2}x(N)^T S_N x(N) + \frac{1}{2}\sum_{k=0}^{N-1}(x(k)^T Q x(k) + u(k)^T R u(k))$$

where $S_N, Q, R \geq 0$, subject to the system dynamics

$$x(k+1) = Ax(k) + Bu(k)$$

FH-DT-LQR, Constrained LQR, & MPC





- Design control policy to minimize the cost function

$$J_{0,N} = \frac{1}{2}x(N)^T S_N x(N) + \frac{1}{2}\sum_{k=0}^{N-1}\left(x(k)^T Q x(k) + u(k)^T R u(k)\right)$$

where $S_N, Q, R \geq 0$, subject to the system dynamics

$$x(k+1) = Ax(k) + Bu(k)$$

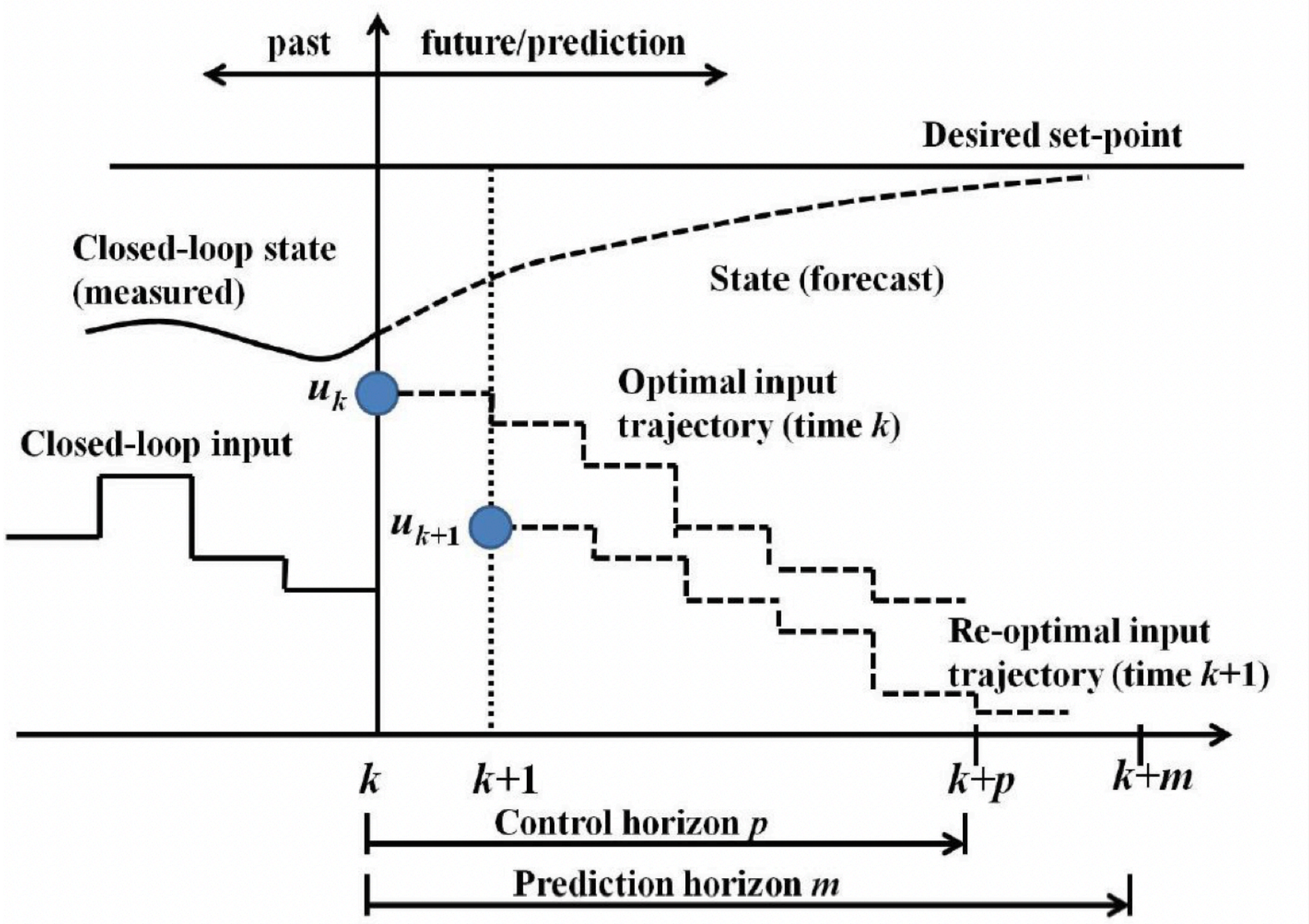
and constraints:

$$Hx(k) \leq h$$

$$Fu(k) \leq f$$

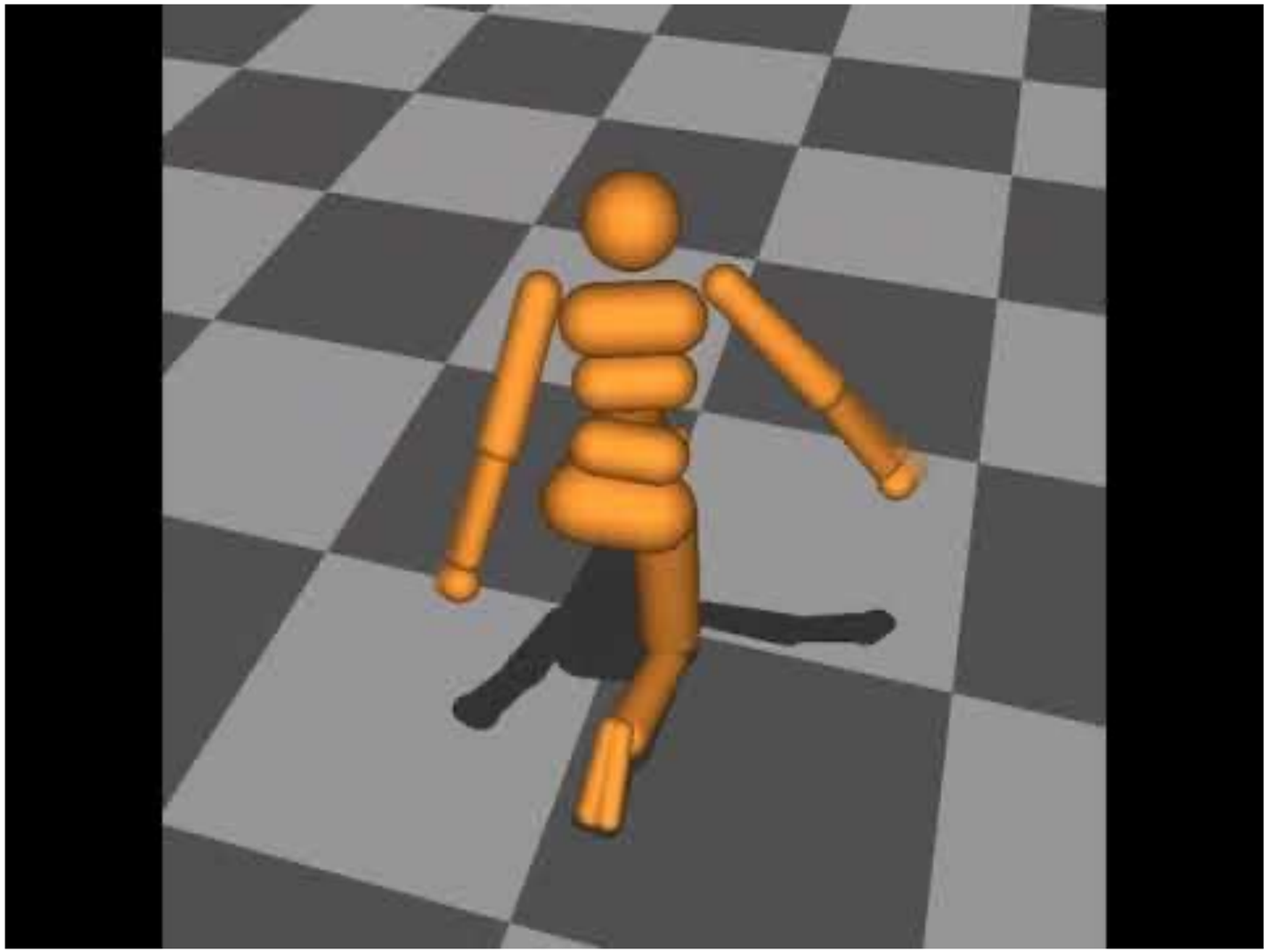
- Can be solved by Quadratic Programming (QP)
  - Many efficient and reliable algorithms available

FH-DT-LQR, Constrained LQR, & MPC



- (Linear) Modal Predictive Control or "Receding Horizon Control
- Calculate $u^*(k:k+N)$, but only use $u^*(k)$and recalculate $u^*(k+1:k+N+1)$in the next step. Essentially, it is a closed loop version of Constrained LQR, therefore, it could be more robust by increasing computation budget.

iterative LQR



https://www.youtube.com/watch?v=anIsw2-Lbco

iterative LQR (iLQR)

- Approximate a nonlinear system as a linear-quadratic system at $\tilde{x}_t$, $\tilde{u}_t$ with Taylor expansion

$$x_{t+1} = f\left(x_t, u_t\right) \approx f\left(\tilde{x}_t, \tilde{u}_t\right) + \nabla_{x_t, u_t} f\left(\tilde{x}_t, \tilde{u}_t\right) \begin{bmatrix} x_t - \tilde{x}_t \\ u_t - \tilde{u}_t \end{bmatrix}$$

$$c\left(x_t, u_t\right) \approx c\left(\tilde{x}_t, \tilde{u}_t\right) + \nabla_{x_t u_t} c\left(\tilde{x}_t, \tilde{u}_t\right) \begin{bmatrix} x_t - \tilde{x}_t \\ u_t - \tilde{u}_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} x_t - \tilde{x}_t \\ u_t - \tilde{u}_t \end{bmatrix}^T \nabla^2_{x, u_t} c\left(\tilde{x}_t, \tilde{u}_t\right) \begin{bmatrix} x_t - \tilde{x}_t \\ u_t - \tilde{u}_t \end{bmatrix}$$

$$\delta x_t = x_t - \tilde{x}_t\,, \quad \delta x_{t+1} = f\left(x_t, u_t\right) - f\left(\tilde{x}_t, \tilde{u}_t\right)$$

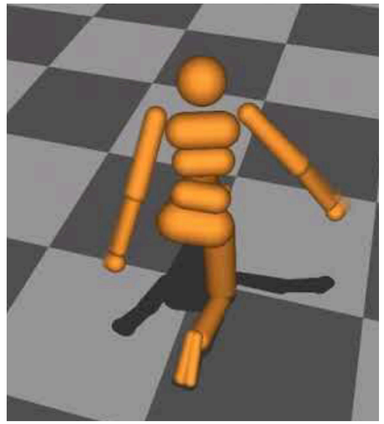
$$\delta u_t = u_t - \tilde{u}_t$$

- Run LQR with state δx_t and action δu_t. Then rerun the linearization to update the model.

Case study: nonlinear model-predictive control with iLQR

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 $\sum_{t'=t}^{t+T} c(\mathbf{x}_{t'}, \mathbf{u}_{t'})$

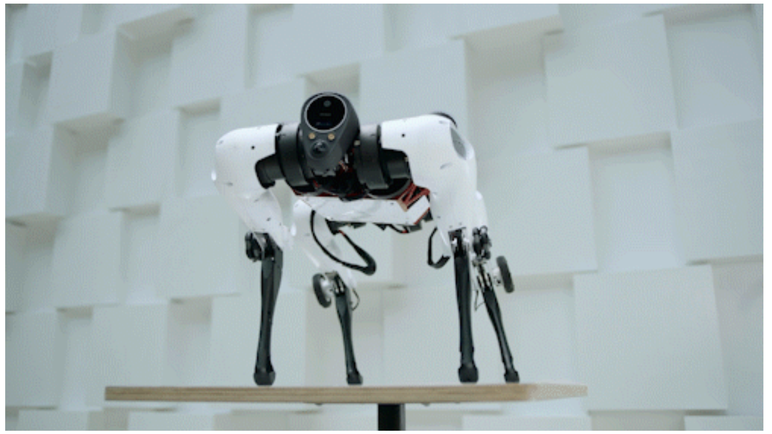
execute action $\mathbf{u}_t$, discard $\mathbf{u}_{t+1}, \ldots, \mathbf{u}_{t+T}$

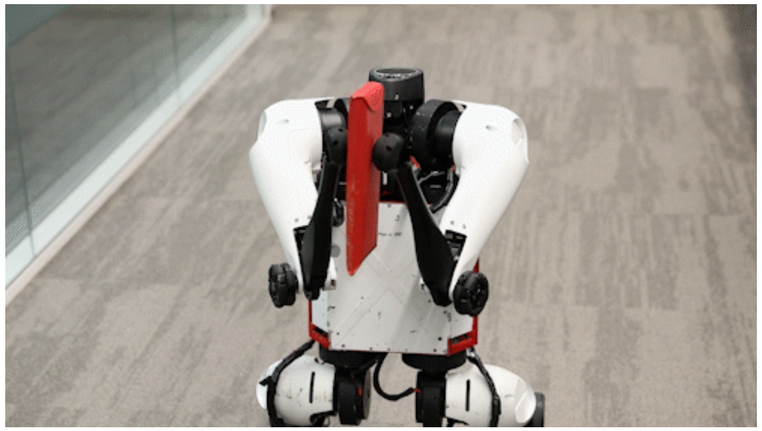
Additional reading on iLQR

- Tassa, Erez, Todorov. (2012). Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization.
- Levine, Abbeel. (2014). Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics.
- a github repo https://github.com/anassinator/ilqr

Tencent’s new dog robot

- Released on March 2, 2021, Nonlinear MPC



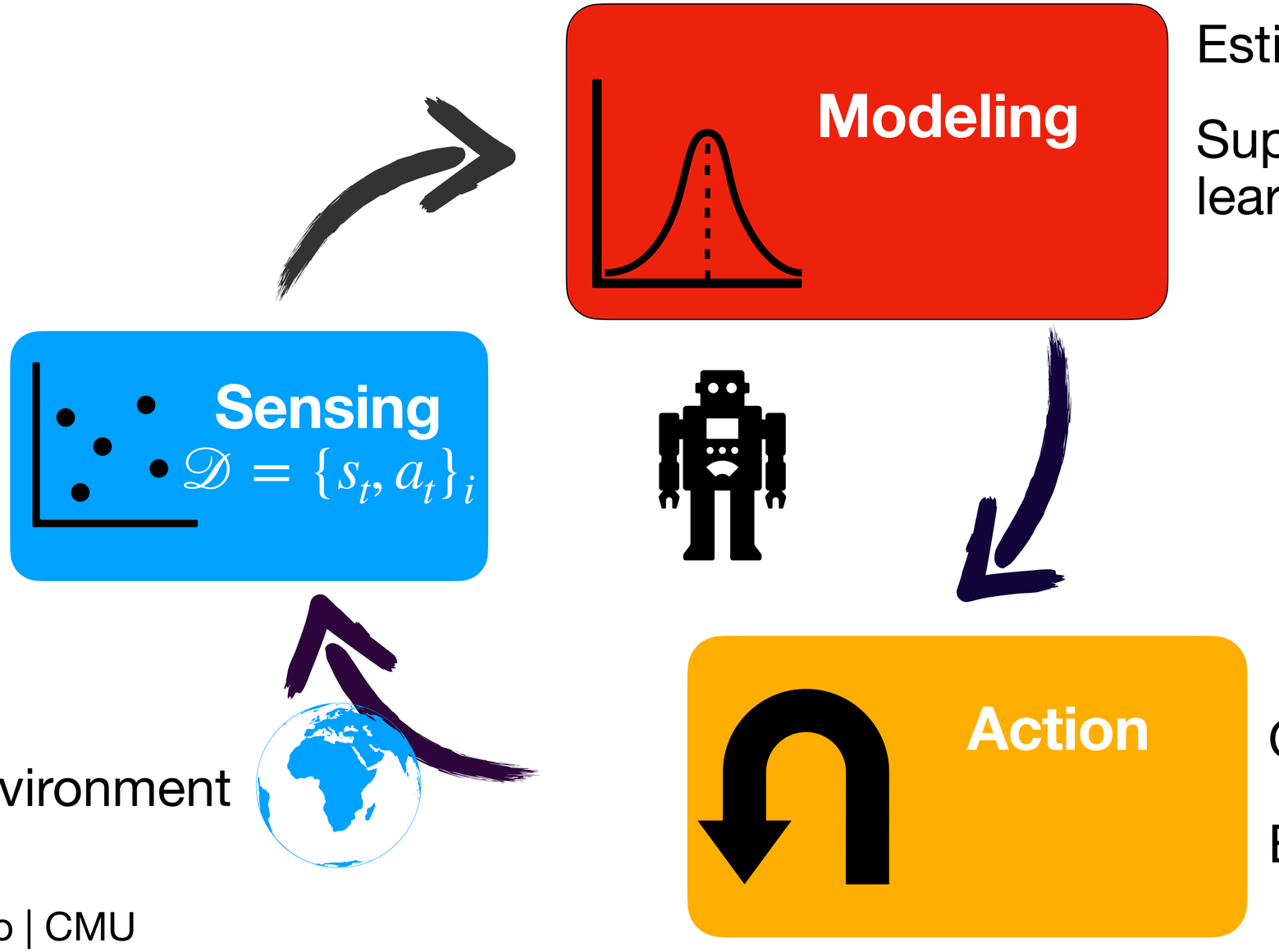


https://www.yicaiglobal.com/news/tencent-unveils-its-first-multi-modal-robot-dog

Where to get the model?

- **We do know the dynamics**
  - Well-studied systems, e.g., automotive
  - Optimal control
- **We know the structure of the dynamics but need to fit some parameters**
  - System identification – fit unknown parameters of a known model, e.g. estimation of the road friction, abrupt changes
  - Adaptive control: the model may not be accurate but the control error vanishes
- **We will learn the dynamics**
  - Fit a general-purpose model to observed transition data, e.g. a comfortable driving distance with surrounding cars
  - Model-based reinforcement learning: similar to adaptive control, the method might focus on the reward rather than the model accuracy

Model-based Reinforcement Learning



Estimate $p(s_{t+1} \mid s_t, a_t)$

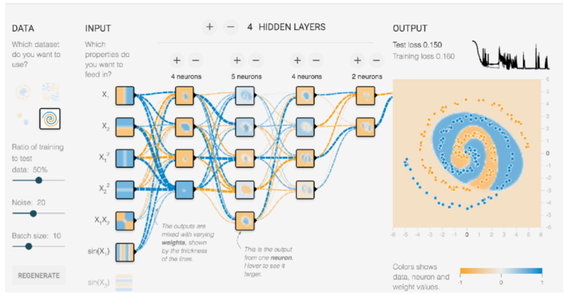
Supervised reinforcement learning

Optimize $a_t \sim \pi_\theta(\cdot \mid s_t)$

Backpropogation with p

What kind of models can we learn?

Neural networks

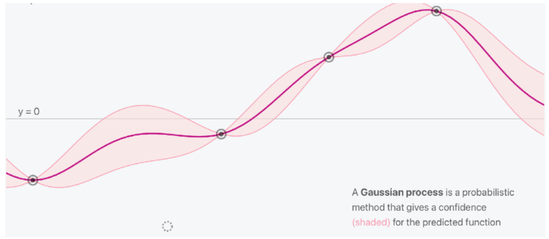


$$s_{t+1} = f_\phi(s_t, a_t)$$

Pro: very expressive, can take the advantage of rich data

Con: not so good in low data regimes/rare events, lack of interpretation

Stochastic functions (Gaussian Processes)



$$s_{t+1} \sim \mathcal{N}(\cdot \mid s_t, a_t, \mathcal{D})$$

Pro: data efficient

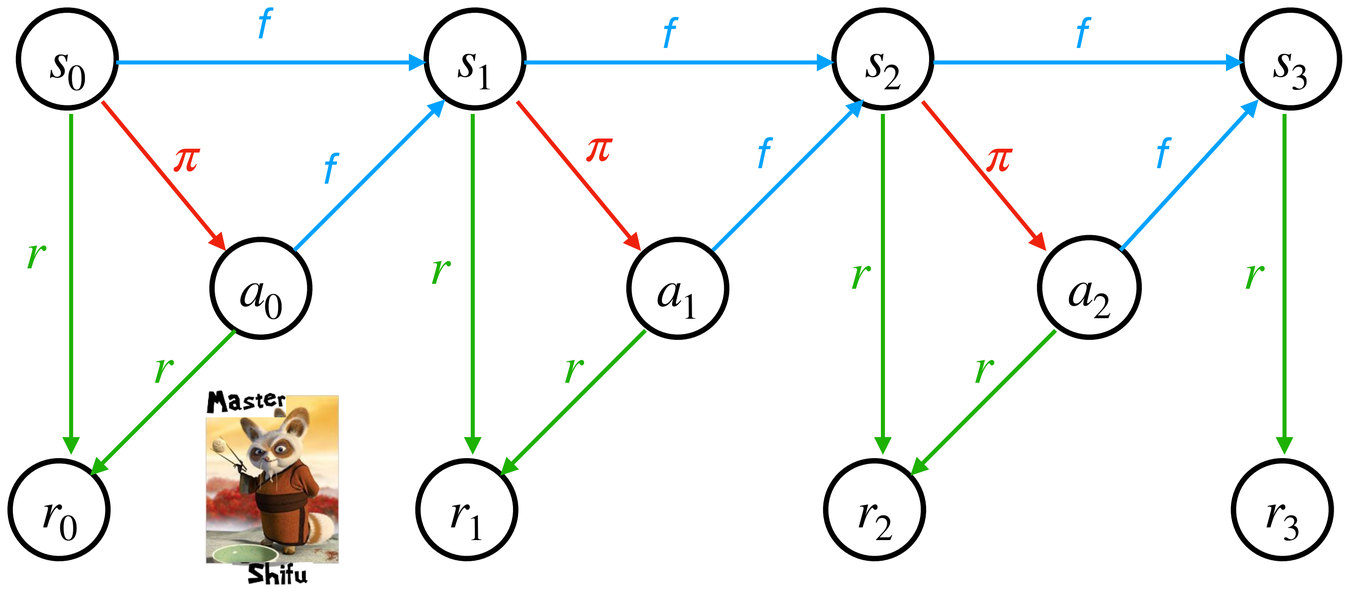
Con: hard to model non-smooth dynamics, slower than NN when dataset is big

Hierarchical /modular structures

Pro: good interpretation, data efficient

Con: hard to train

Reinforcement Learning - NN model-based



MB-NN-RL-1.0

Issue: we may over-reply on the model, which could have safety issues.

Planning helps to make the model more trustworthy

1. Run base policy $\pi_{\theta^{(0)}}\left(a_t \mid s_t\right)$ (e.g., random policy) to collect $\mathcal{D} = \{(s_t, a_t, s_{t+1})_{t=1:D}\}$
2. Learn model $f_{\phi^{(i)}}$ by minimizing $\sum_t \|f_{\phi^{(i)}}(s_t, a_t) - s_{t+1}\|^2$
3. Backpropagate to optimize $\pi_{\theta^{(k)}}\left(a_t \mid s_t\right)$ through $f_{\phi^{(i)}}$
4. Execute with policy $\pi_{\theta^{(k)}}\left(a_t \mid s_t\right)$, append new data to $\mathcal{D}$

Reinforcement Learning - NN model-based

MB-NN-RL-2.0

1. Run base policy $\pi_{\theta^{(0)}}\left(a_t \mid s_t\right)$ (e.g., random policy) to collect $\mathcal{D} = \{(s_t, a_t, s_{t+1})_{t=1:D}\}$
2. Learn model $f_{\phi^{(i)}}$ by minimizing $\sum_t \|f_{\phi^{(i)}}(s_t, a_t) - s_{t+1}\|^2$
3. Plan through $f_{\phi^{(i)}}(s_t, a_t)$ to choose actions
4. Execute the first planned action, observe results states s_{t+1} (e.g. MPC)
5. Append (s_t, a_t, s_{t+1}) to $\mathcal{D}$

How to do planning?

- Planning with linearized models (local model)
  - i-LQR
- Planning with sampling based methods
  - CEM, PETS

Case study: local models and iLQR

Learning Contact-Rich Manipulation Skills with Guided Policy Search

Sergey Levine, Nolan Wagener, Pieter Abbeel

***Abstract*— Autonomous learning of object manipulation skills can enable robots to acquire rich behavioral repertoires that scale to the variety of objects found in the real world. However, current motion skill learning methods typically restrict the behavior to a compact, low-dimensional representation, limiting its expressiveness and generality. In this paper, we extend a recently developed policy search method [1] and use it to learn a range of dynamic manipulation behaviors with highly general policy representations, without using known models or example demonstrations. Our approach learns a set of trajectories for the desired motion skill by using iteratively refitted time-varying linear models, and then unifies these trajectories into a single control policy that can generalize to new situations. To enable this method to run on a real robot, we introduce several improvements that reduce the sample count and automate parameter selection. We show that our method can acquire fast, fluent behaviors after only minutes of interaction time, and can learn robust controllers for complex tasks, including putting together a toy airplane, stacking tight-fitting lego blocks, placing wooden rings onto tight-fitting pegs, inserting a shoe tree into a shoe, and screwing bottle caps onto bottles.**

I. Introduction

Autonomous acquisition of manipulation skills has the potential to dramatically improve both the ease of deployment of robotic platforms, in domains ranging from manufacturing to household robotics, and the fluency and speed of the robot's motion. It is often much easier to specify *what* a robot should do, by means of a compact cost function, than

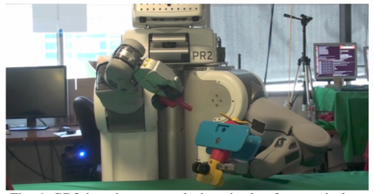


Fig. 1: PR2 learning to attach the wheels of a toy airplane.

In this paper, we show that a range of motion skills can be learned using only general-purpose policy representations. We use our recently developed policy search algorithm [1], which combines a sample-efficient method for learning linear-Gaussian controllers with the framework of guided policy search, which allows multiple linear-Gaussian controllers (trained, for example, from several initial states, or under different conditions) to be used to train a single nonlinear policy with any parameterization, including complex, high-dimensional policies represented by large neural networks. This policy can then generalize to a wider range of conditions than the individual linear-Gaussian controllers.

We present several modifications to this method that make it practical for deployment on a robotic platform.

https://safeai-lab.github.io/lcs-fall2020.html

Cross Entropy Method (Random Shooting)

Optimal planning:

$$a_1, \ldots, a_T = \arg\max J(a_1, \ldots, a_T), \mathbf{A} = \arg\max J(\mathbf{A})$$

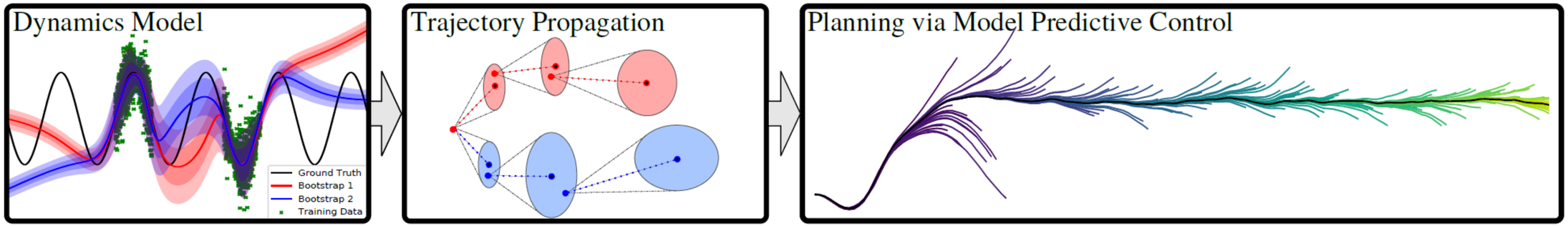
Simplest method: guess and check

1. Pick $\mathbf{A_1}, \ldots, \mathbf{A_N}$ from some distribution (e.g., uniform)
2. Choose $\mathbf{A_i}$ based on $\arg\max J(\mathbf{A})$

Case study: CEM with MPC

Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models

Kurtland Chua **Roberto Calandra** **Rowan McAllister** **Sergey Levine**
Berkeley Artificial Intelligence Research
University of California, Berkeley
{kchua, roberto.calandra, rmcallister, svlevine}@berkeley.edu

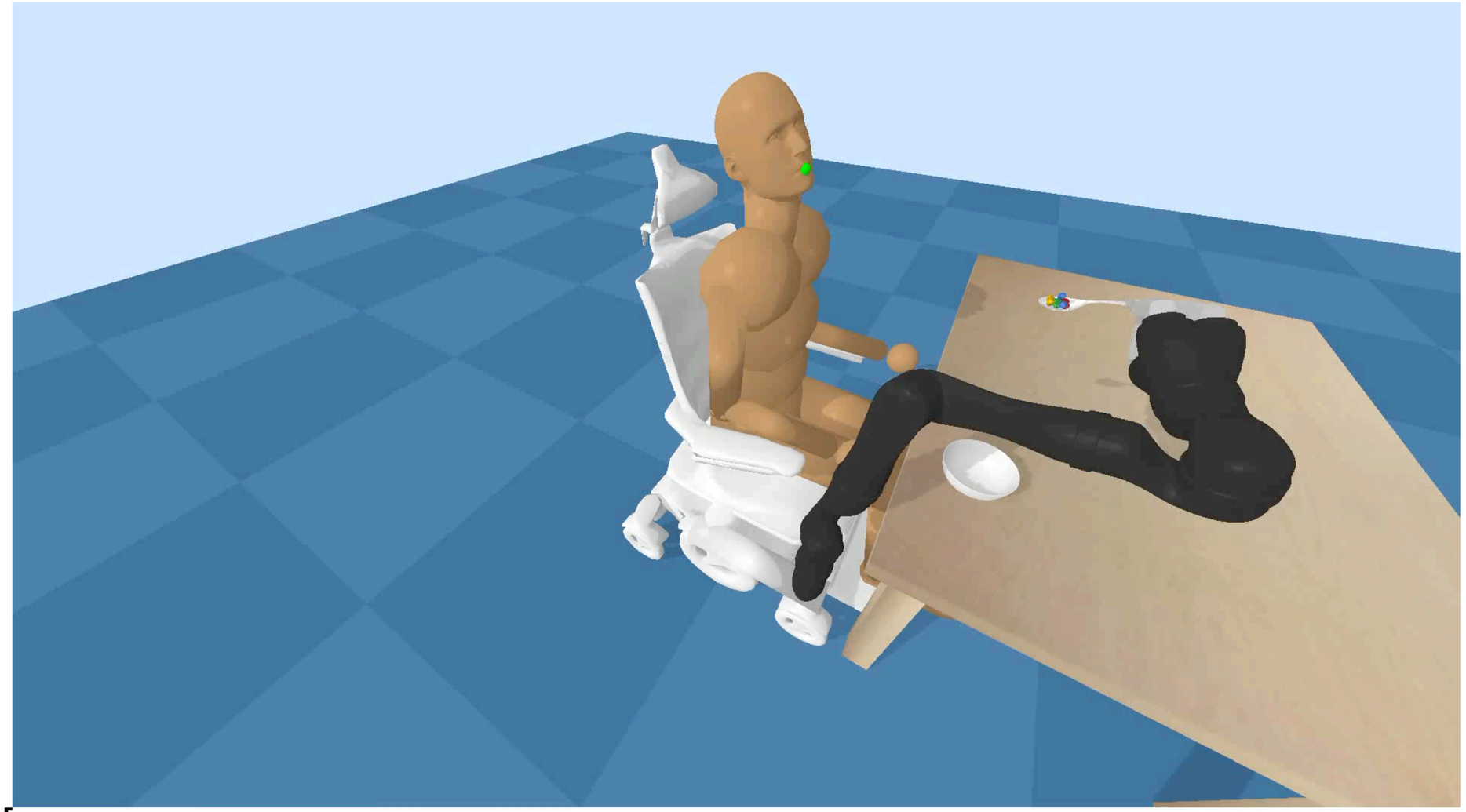


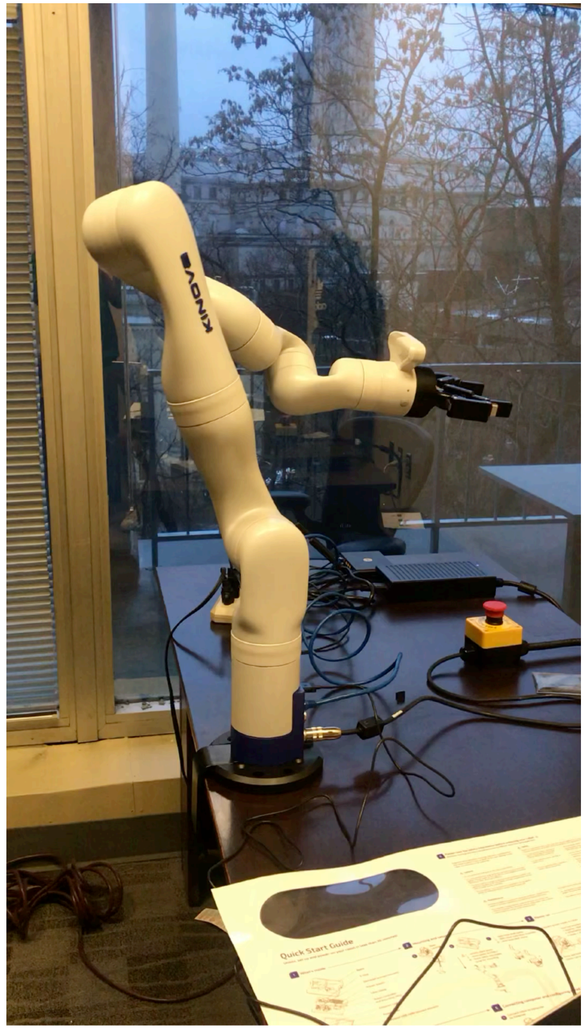
Algorithm 1 Our model-based MPC algorithm '*PETS*':

Initialize data $\mathbb{D}$ with a random controller for one trial.
for Trial $k = 1$ to K **do**
  Train a *PE* dynamics model $\widetilde{f}$ given $\mathbb{D}$.
  for Time $t = 0$ to TaskHorizon **do**
    for Actions sampled $\boldsymbol{a}_{t:t+T} \sim \text{CEM}(\cdot)$, 1 to NSamples **do**
      Propagate state particles $\boldsymbol{s}_\tau^p$ using *TS* and $\widetilde{f}|\{\mathbb{D}, \boldsymbol{a}_{t:t+T}\}$.
      Evaluate actions as $\sum_{\tau=t}^{t+T} \frac{1}{P} \sum_{p=1}^{P} r(\boldsymbol{s}_\tau^p, \boldsymbol{a}_\tau)$
      Update CEM$(\cdot)$ distribution.
    Execute first action $\boldsymbol{a}_t^*$ (only) from optimal actions $\boldsymbol{a}_{t:t+T}^*$.
    Record outcome: $\mathbb{D} \leftarrow \mathbb{D} \cup \{\boldsymbol{s}_t, \boldsymbol{a}_t^*, \boldsymbol{s}_{t+1}\}$.

Case study: planning with CEM

Safe RL with non-stationary environment (a shaking head)





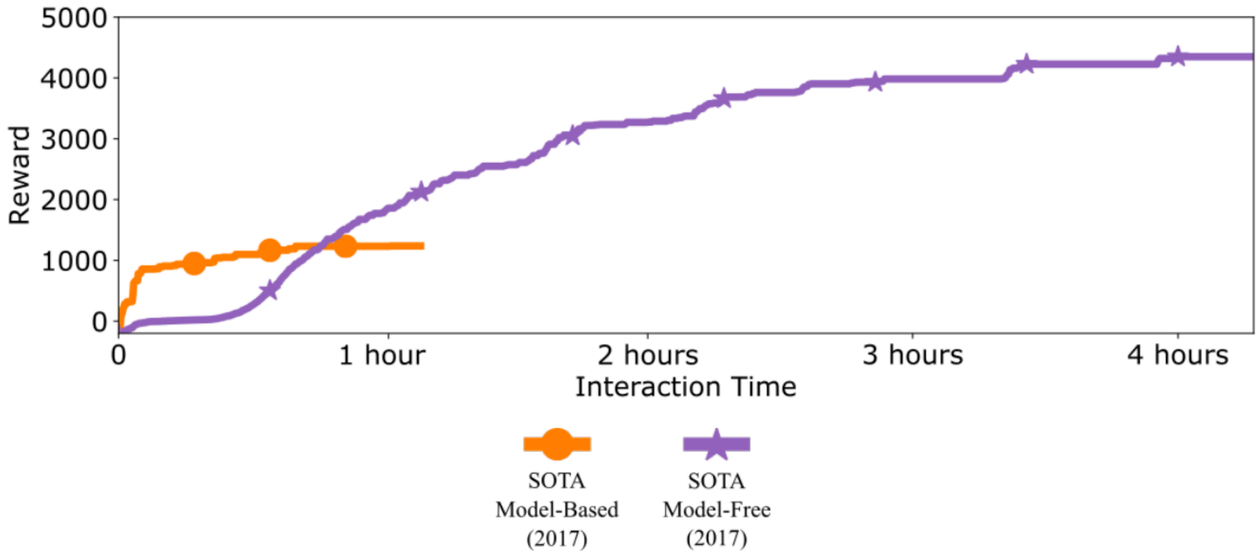
Challenges in model learning

- Under-modelling: If the model class is restricted (e.g., linear function or gaussian process) we have under-modeling: we cannot represent complex dynamics, e.g., contact dynamics that are not smooth. As a result, though we learn faster than model free in the beginning, MBRL ends up having worse asymptotic performance than model-free methods, that do not suffer from model bias.

- Over-fitting: If the model class is very expressive (e.g., neural networks) the model will overfit, especially in the beginning of training, where we have very few samples

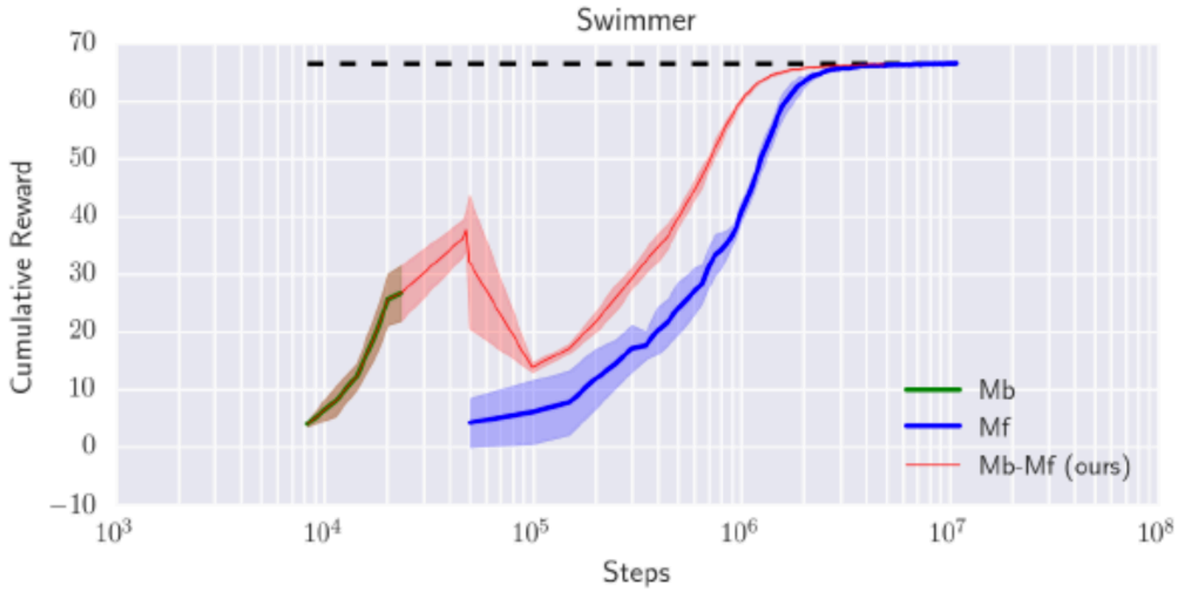
- Uncertainty/errors propagated and amplified through planning

Comparative Performance on HalfCheetah



Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, Sergey Levine
University of California, Berkeley



Model-based vs Model free

- Model-based

  + data efficient in training

  + Possibility to transfer across tasks

  + Increase interoperability

  - Do not optimize directly over performance

  - Usually need domain knowledge (overfitting/under-fitting)

  - Maybe hard to learn policy

- Model-free

  + Need little assumption

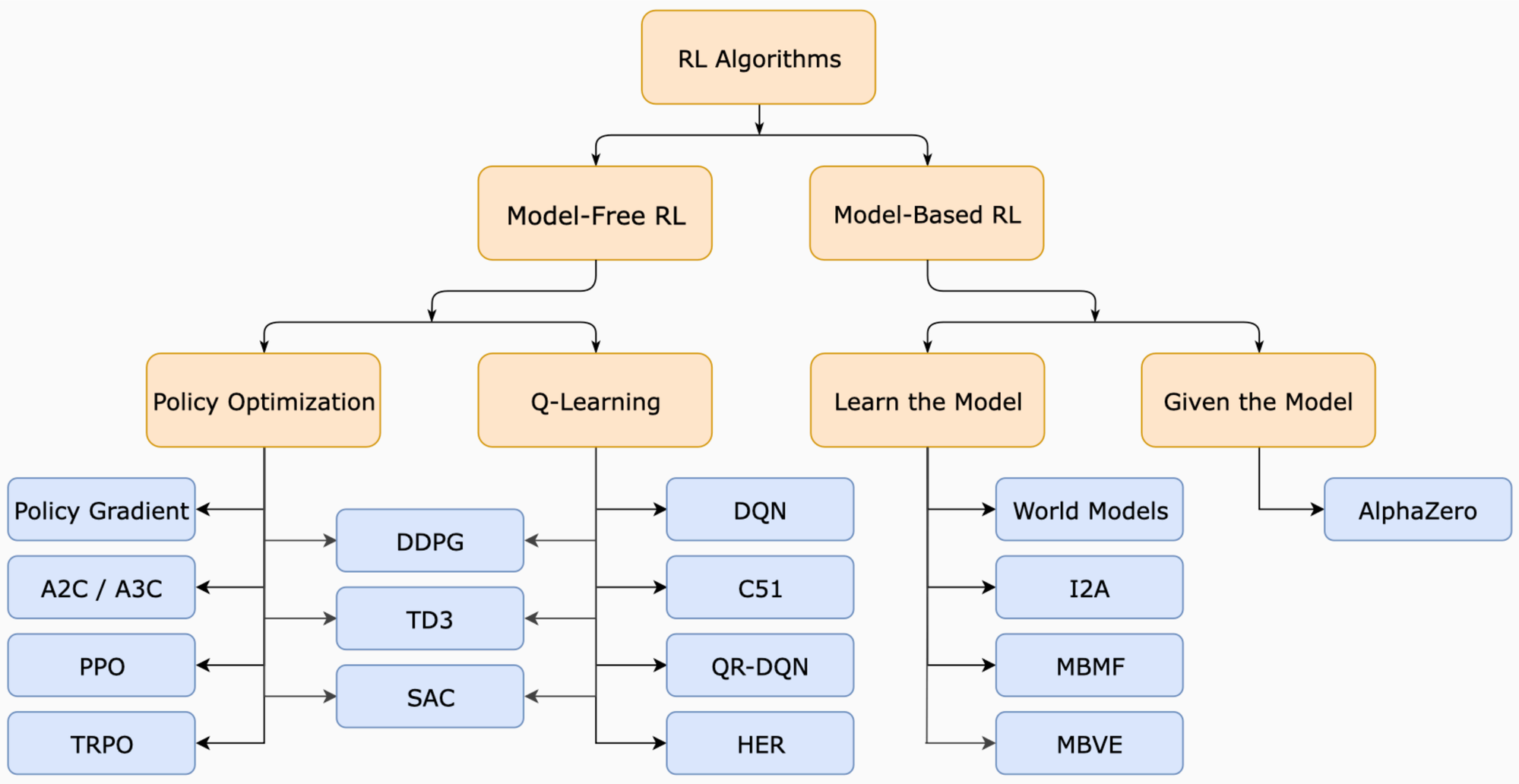
  + Efficient for learning complex policy

  - Require a lot training data

  - Not transferable and lack of interoperability

Widely used in safety-critical applications

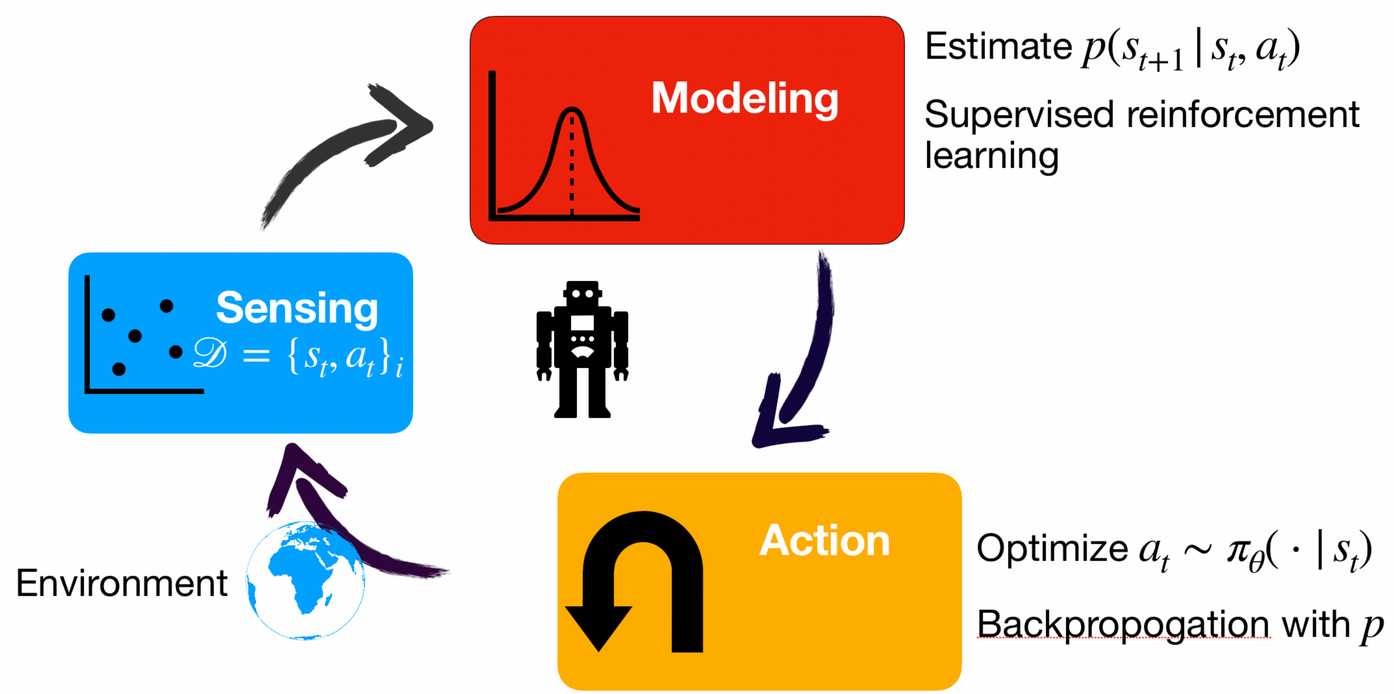
Popular RL algorithms



https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

Summary

- Model-based RL
  - Choose the format of the models
  - Refitting with new data
  - Replanning at each step (MPC: iLQR or CEM)



Worthy reading

- Blog: Model-Based Reinforcement Learning: Theory and Practice
  - https://bair.berkeley.edu/blog/2019/12/12/mbpo/
- Lecture 9, Deep RL Bootcamp
  - https://sites.google.com/view/deep-rl-bootcamp/lectures