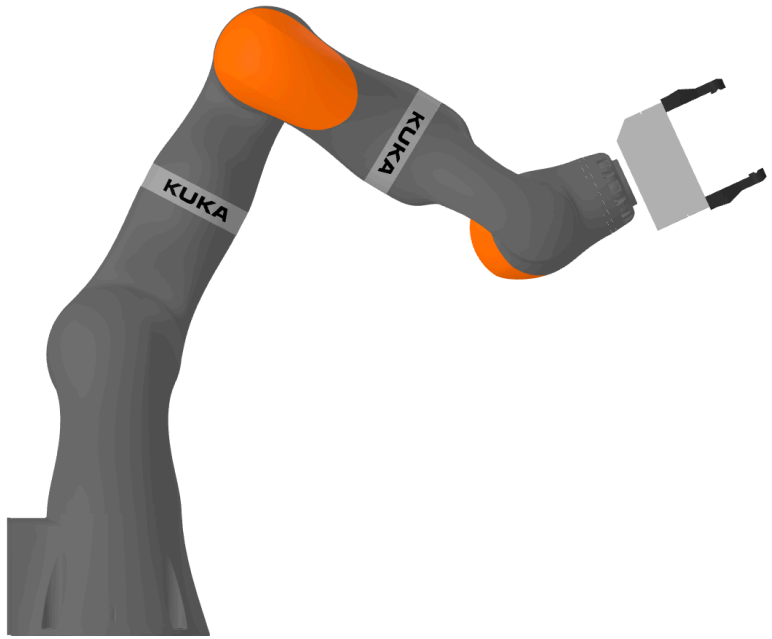


Model-based Learning II

MIT 6.4210/2

Robotic Manipulation

Fall 2026, Lecture 13

Massachusetts Institute of Technology

Reinforcement learning

Lecture 13: planning with models

Leslie Kaelbling

6.4210 Fall 2026

Learning a policy with a learned model

Once we learn a world model, we can use it as a simulator and do RL

- We generate transitions and rewards in the simulator, then train a policy using the RL algorithms we already know
- With differentiable deterministic dynamics and policy, and differentiable rewards, we can compute policy gradients analytically by backpropagating through simulated trajectories
- During a policy update, the model parameters stay fixed

We move computation into training so that action selection is cheap

Notes Section 3.1, Direct methods: policy gradients with a known model

Learning a policy or planning now

A learned policy moves computation into training
Planning uses a model to reason about the current situation

- Now we will consider the more general case in which actions may be torques or discrete commands, and their outcomes may be stochastic
- We can compare possible futures using a known or learned model
- After acting, we observe the outcome and plan again

The plan must account for rewards, dynamics, and limited computation

The planning objective

We search from s_0 over a sequence $\mathbf{a} = (a_0, \ldots, a_{H-1})$

$$J(\mathbf{a}) = \mathbb{E}\left[\sum_{k=0}^{H-1} \gamma^k R(s_k, a_k) + \gamma^H \hat{V}(s_H)\right], \qquad \mathbf{a}^* = \arg\max_{\mathbf{a} \in \mathcal{A}^H} J(\mathbf{a})$$

- H limits computation; the real task may continue much longer
- $\hat{V}(s_H)$ estimates the return from continuing *optimally* after the planning horizon; it plays the role of a search heuristic
- A fixed sequence is *open loop*: later actions cannot depend on the outcomes of earlier actions

s_0: current state estimate $\mathcal{A}$: allowed actions γ: discount R: reward

Replanning with model predictive control

We plan H steps ahead, execute the first action, and plan again

Inputs: model p_θ, reward R, terminal value $\hat{V}$, horizon H, discount γ

$s \leftarrow$ current state estimate
While the task continues:
  $\mathbf{a} \leftarrow \mathsf{Plan}(s, p_\theta, R, \hat{V}, H, \gamma)$
  $a \leftarrow a_0$
  $o \leftarrow \mathsf{ExecuteAndObserve}(a)$
  $s \leftarrow \mathsf{UpdateStateEstimate}(s, a, o)$

- New observations let us respond to unintended outcomes
- Moving the horizon forward reveals useful steps beyond the old horizon

A terminal estimate helps compensate for the short horizon

Approximations in planning

- **Finite horizon:** we truncate the future and may add a terminal value
- **Sampled outcomes:** a few rollouts approximate an expectation
- **Determinization:** one predicted outcome stands in for a distribution
- **Sampled actions:** a finite set approximates continuous choices
- **Partial observability:** we approximate beliefs and future observations

The next methods search for action sequences
Tree search can represent different actions after different outcomes

Scoring candidate action sequences

Candidate i contains an action for each of the H steps:

$$\mathbf{a}^{(i)} = (a_0^{(i)}, \ldots, a_{H-1}^{(i)}), \qquad i = 1, \ldots, M$$

For each candidate, we simulate K rollouts from the same current state

$$s_0^{(i,j)} = s_0, \qquad s_{k+1}^{(i,j)} \sim p_\theta(\cdot \mid s_k^{(i,j)}, a_k^{(i)})$$

$$\hat{J}(\mathbf{a}^{(i)}) = \hat{J}_i = \frac{1}{K} \sum_{j=1}^{K} \left[\sum_{k=0}^{H-1} \gamma^k R(s_k^{(i,j)}, a_k^{(i)}) + \gamma^H \hat{V}(s_H^{(i,j)}) \right]$$

All rollouts of candidate i use the same actions, even when states differ

i: candidate index j: rollout index k: planning step

Random shooting

We sample from an arbitrary distribution q over length-H action sequences

Inputs: horizon H, distribution q, M candidates, K rollouts per candidate

1 For $i = 1, \ldots, M$:
  Sample $\mathbf{a}^{(i)} = (a_0^{(i)}, \ldots, a_{H-1}^{(i)}) \sim q$
  Simulate K model rollouts and compute $\hat{J}_i$
2 Return $\underset{\mathbf{a} \in \{\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(M)}\}}{\arg\max} \hat{J}(\mathbf{a})$

- We might sample each action uniformly from its allowed range
- The sampling distribution q stays fixed during the search

The cross-entropy method

CEM adapts the distribution q we use to sample possible solutions

Inputs: horizon H, initial μ, Σ, M candidates, $L < M$ elites, iteration budget

1 Until the budget runs out:

Sample $\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(M)} \sim \mathcal{N}(\mu, \Sigma)$

Score each candidate using $\hat{J}_i$

Let E index the L highest-scoring candidates, called elites

$\mu \leftarrow \frac{1}{L} \sum_{i \in E} \mathbf{a}^{(i)}$

$\Sigma \leftarrow \frac{1}{L} \sum_{i \in E} (\mathbf{a}^{(i)} - \mu)(\mathbf{a}^{(i)} - \mu)^\top + \varepsilon I$

2 Return $\arg\max_{\mathbf{a} \in \mathcal{S}} \hat{J}(\mathbf{a})$

q: proposal distribution μ, Σ: mean and covariance
$\mathcal{S}$: all sampled candidates $\varepsilon > 0$: variance floor Chua et al. (2018)

Model predictive path integral control

MPPI also adapts the distribution q we use to sample possible solutions

$$q_{\mu,\Sigma} = \mathcal{N}(\mu, \Sigma), \qquad \mathbf{a}^{(i)} \sim q_{\mu,\Sigma}$$

We assign a smoothly varying weight to each candidate

$$\widetilde{w}_i = e^{\hat{J}_i/\eta} \frac{q_0(\mathbf{a}^{(i)})}{q_{\mu,\Sigma}(\mathbf{a}^{(i)})}, \qquad w_i = \frac{\widetilde{w}_i}{\sum_j \widetilde{w}_j}$$

$$\mu_{\text{new}} = \sum_i w_i \mathbf{a}^{(i)}, \qquad \Sigma \text{ stays fixed}$$

Lower temperature η emphasizes higher-return candidates
The density ratio corrects for the current sampling distribution

μ, Σ: mean and covariance q_0: reference distribution $\eta > 0$: temperature
Williams et al., Information Theoretic Model Predictive Control

The MPPI update loop

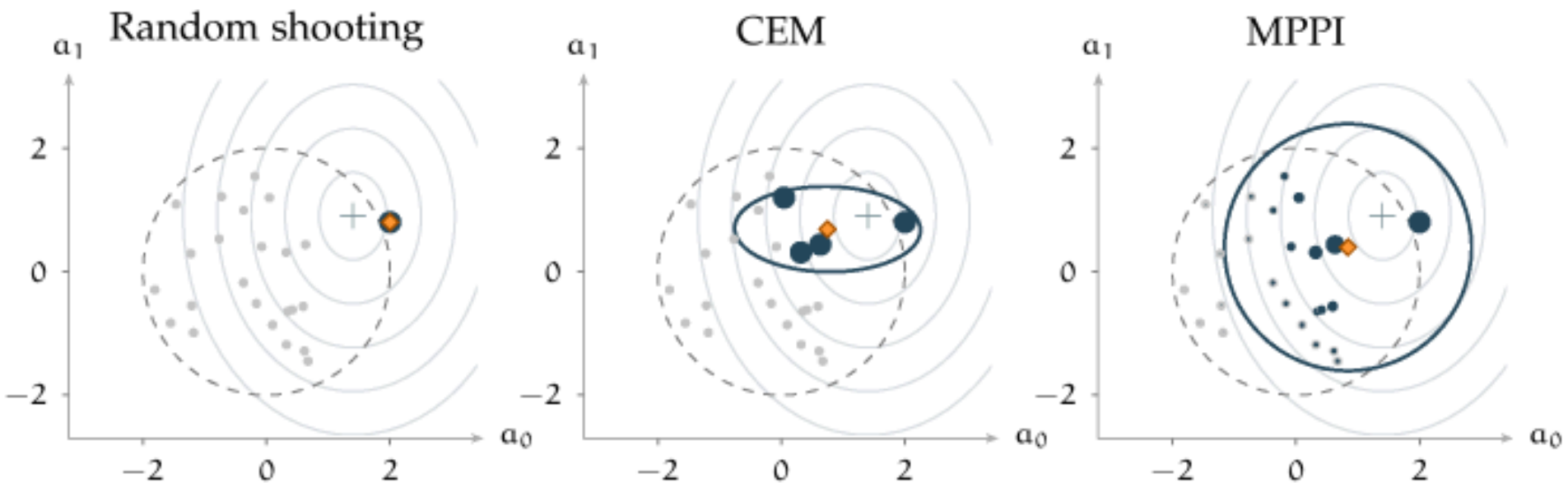
Inputs: horizon H, initial μ, fixed Σ, q_0, η, sample count M, iteration budget

1 Until the budget runs out:
  Sample $\mathbf{a}^{(i)} \sim \mathcal{N}(\mu, \Sigma)$ for $i = 1, \ldots, M$
  Score each candidate using $\hat{J}_i$
  Compute $\tilde{w}_i \leftarrow e^{\hat{J}_i/\eta} q_0(\mathbf{a}^{(i)}) / q_{\mu,\Sigma}(\mathbf{a}^{(i)})$
  Normalize $w_i \leftarrow \tilde{w}_i / \sum_j \tilde{w}_j$
  Update the mean $\mu \leftarrow \sum_i w_i \mathbf{a}^{(i)}$
2 Return μ

Reweighting lets us reuse samples from $q_{\mu,\Sigma}$ to estimate the mean under $q^*(\mathbf{a}) \propto q_0(\mathbf{a}) e^{J(\mathbf{a})/\eta}$

$q_{\mu,\Sigma} = \mathcal{N}(\mu, \Sigma)$: proposal q_0: reference q^*: target distribution
$\eta > 0$: temperature $\hat{J}_i$: estimated return w_i: normalized weight

Random shooting, CEM, and MPPI



Each panel uses the same 24 candidate sequences

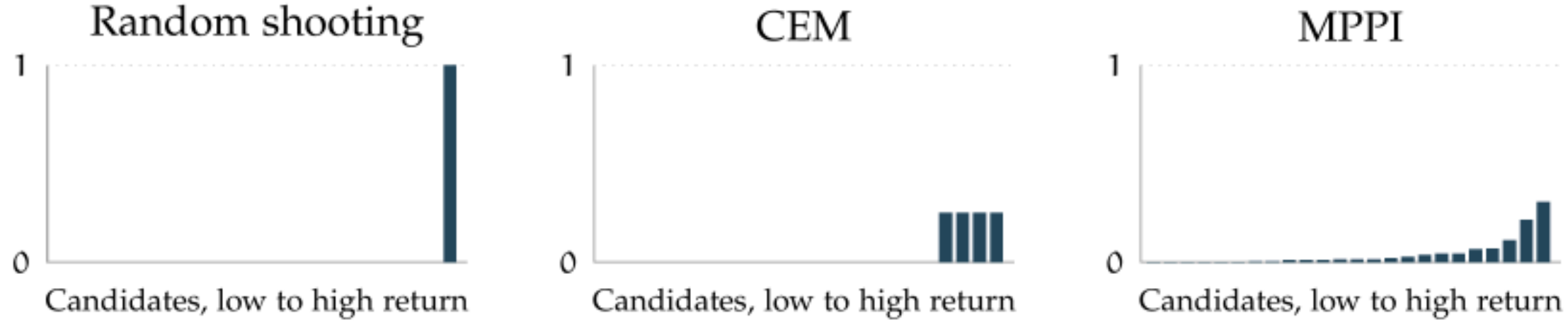
Ellipses: old (dashed) and new (solid) distributions; diamonds: sequence or mean

Contours: equal return; +: optimum; dots: candidates; dot area: sample weight

CEM fits covariance; MPPI keeps it fixed

a_0, a_1: first and second actions of a candidate sequence

How the sampling methods weight candidates



- Random shooting keeps only the highest-scoring candidate
- CEM gives each of four elites weight 1/4
- MPPI gives every candidate a smooth weight

For this update, $\eta = 1$ and $q_0 = q_{\mu,\Sigma}$

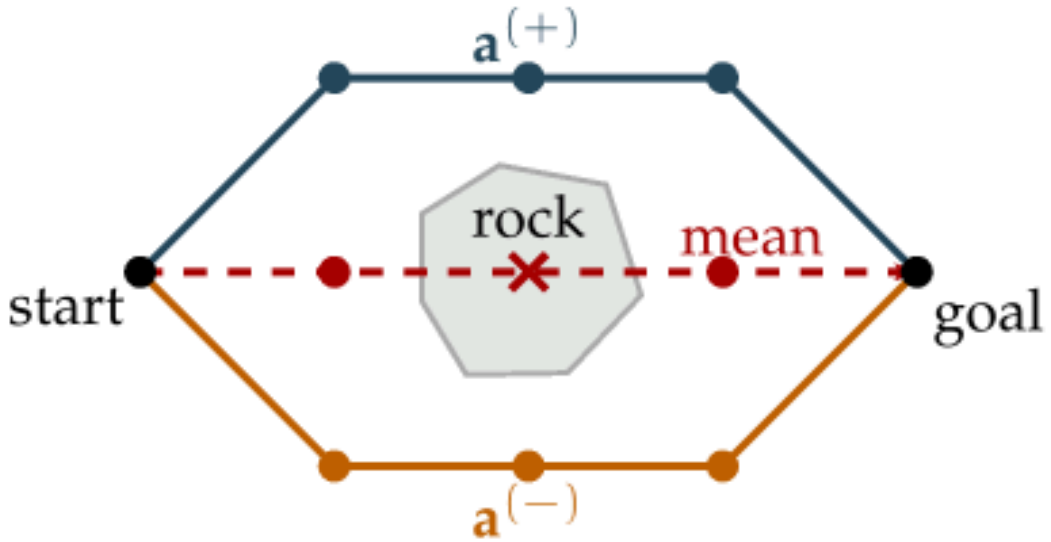
η: MPPI temperature q_0: reference distribution $q_{\mu,\Sigma}$: proposal distribution

Averaging actions can produce a bad plan

Both Gaussian CEM and MPPI require actions in a vector space where averaging actions makes sense

Point robot: $s_{k+1} = s_k + a_k$
Start $(-2, 0)$, goal $(2, 0)$

$$\bar{a}_k = \tfrac{1}{2}(a_k^{(+)} + a_k^{(-)}) = (1, 0)$$



Two routes avoid the rock
Their mean goes straight through it

CEM: sample around the mean; return the best sampled sequence
MPPI: return the mean

s_k: position a_k: displacement $\bar{a}_k$: mean displacement

Gradient-based trajectory optimization

With differentiable dynamics, rewards, and terminal values, we optimize the actions

Inputs: initial $\mathbf{a}$, step size α, model f_θ, iteration budget

1 Until the budget runs out:
  Set the rollout start to the current state s_0
  Simulate $s_{k+1} \leftarrow f_\theta(s_k, a_k)$ for $k = 0, \ldots, H-1$
  Compute $J = \sum_{k=0}^{H-1} \gamma^k R(s_k, a_k) + \gamma^H \hat{V}(s_H)$
  Backpropagate to obtain $\mathbf{g} \leftarrow \nabla_{\mathbf{a}} J$
  $\mathbf{a} \leftarrow \Pi_{\mathcal{A}^H}(\mathbf{a} + \alpha \mathbf{g})$
2 Return $\mathbf{a}$

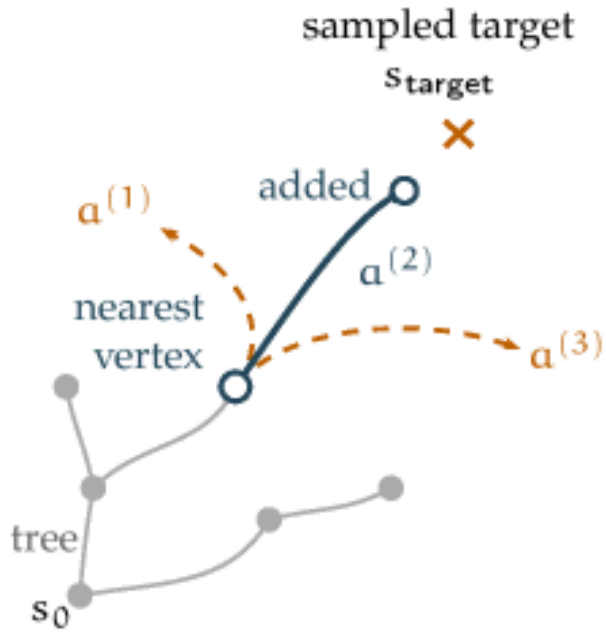
Initialization matters; optimization can exploit model errors

$\Pi_{\mathcal{A}^H}$: projection onto allowed action sequences

Kinodynamic RRT

Vertices store full states; edges follow the learned dynamics $s' = f_\theta(s, a)$

- We sample a target and find the nearest tree vertex
- We simulate several controls from that vertex
- We add the valid rollout ending closest to the target



We grow edges by forward simulation, without derivatives or exact connections

The output is one feasible path; basic RRT does not optimize return

LaValle and Kuffner (2001), Randomized Kinodynamic Planning

Kinodynamic RRT by forward simulation

Inputs: start s_0, goal region $\mathcal{G}$, model f_θ, state distance d, validity test, budget

1 Initialize a tree T with root s_0
2 Until the budget runs out:
    Sample a target state, occasionally in $\mathcal{G}$
    Find its nearest tree vertex using d
    Simulate sampled constant controls for short durations from that vertex
    Discard invalid trajectories; skip this iteration if none remain
    Add the valid trajectory whose endpoint is closest to the target
    If the endpoint is in $\mathcal{G}$, return its root-to-goal control sequence
3 Report that no path was found within the budget

LaValle, Planning Algorithms, Section 14.4.3

Tree search and contingent decisions

When we know a probabilistic transition model, we can consider many possible outcomes and select different actions depending on what happens

Sampling $s' \sim P(\cdot \mid s, a)$ is enough; we need not evaluate probabilities or densities

- In deterministic problems, branches represent our action choices
- In stochastic problems, the tree also branches on chance outcomes
- In games, opponents choose actions to favor their own objectives

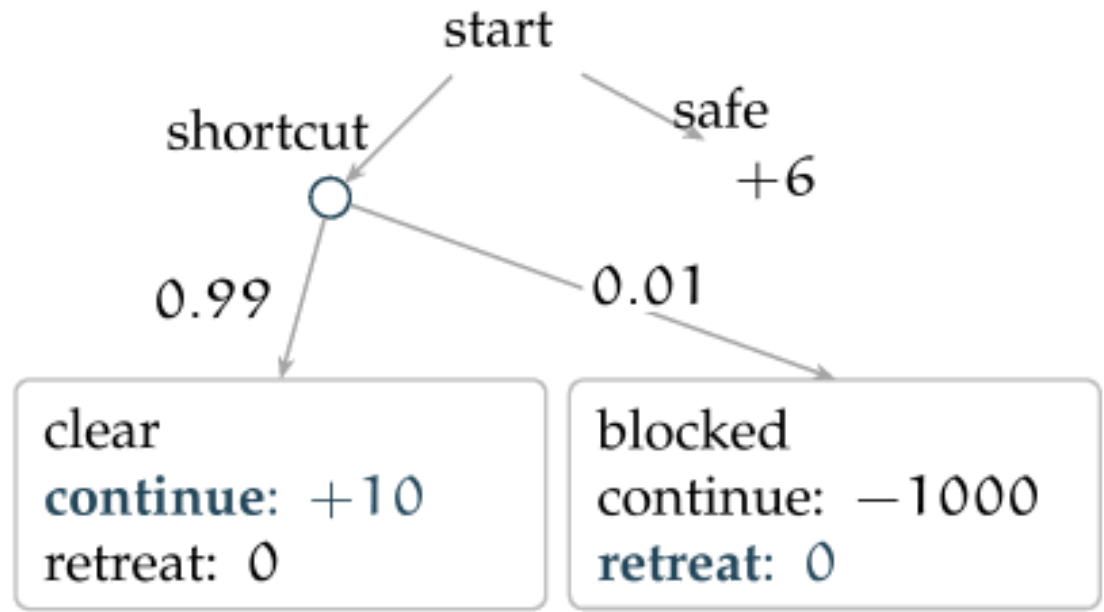
The resulting plan can be a partial *contingent policy*

$P(s' \mid s, a)$: transition model

An attractive shortcut with a rare bad outcome

A two-step task, with $\gamma = 1$

- The safe action ends the task with reward 6
- The shortcut gives reward 0, then reveals whether the route is clear
- After observing the route, we can continue or retreat



The contingent plan continues if clear and retreats if blocked

A sequence and a contingent policy have different values

If we assume the likely outcome, the shortcut appears to give 10

$$J(\text{shortcut, continue}) = 0.99(10) + 0.01(-1000) = -0.1$$
$$J(\text{shortcut, retreat}) = 0$$

A correctly evaluated fixed sequence therefore favors the safe action: 6

$$J(\text{shortcut, continue if clear, retreat if blocked}) = 0.99(10) + 0.01(0) = 9.9$$

- A tree can credit the future decision when choosing the first action
- Replanning alone does not give a fixed-sequence search that credit

A finite search can still miss the rare bad outcome

Monte Carlo tree search

MCTS uses repeated simulations to grow and evaluate a partial search tree
At our decision nodes, UCT balances value and exploration:

$$a = \arg\max_a \left[\bar{Q}(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}} \right]$$

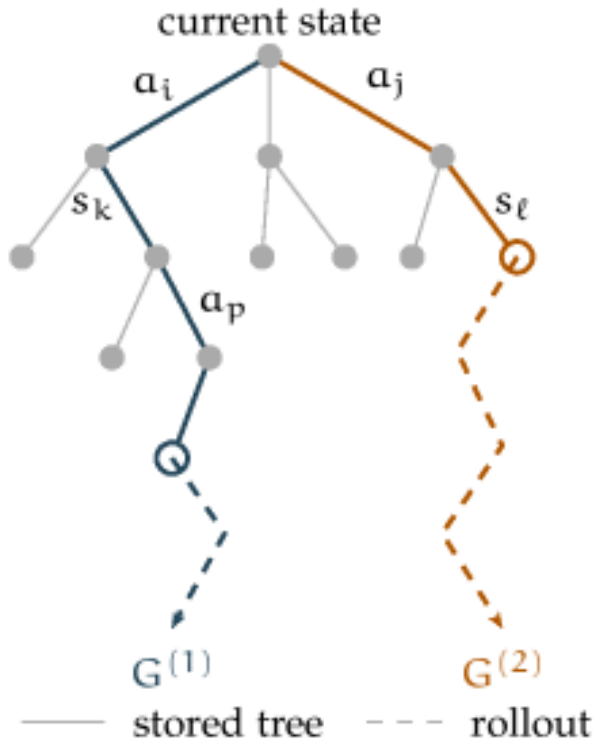
- High average return makes an action attractive
- Limited sampling adds an exploration bonus; unvisited actions go first
- Chance nodes sample outcomes using the model probabilities

$\bar{Q}(s,a)$: mean simulated return $N(s,a)$: edge visits $N(s) = \sum_a N(s,a)$
$c > 0$: exploration constant UCT: upper confidence bounds applied to trees
Kocsis and Szepesvári, Bandit Based Monte-Carlo Planning

MCTS simulations, rollouts, and backups

Branches alternate between chosen actions and sampled outcomes

1. Start a simulation at the root
2. Select tree actions; sample chance outcomes
3. Expand a new node
4. Estimate remaining return G with a rollout or leaf value
5. Back up along the path:
   $G \leftarrow R(s, a) + \gamma G$
   Update visit counts and average returns
6. Repeat to the computation limit
7. Execute the most visited root action



Open circles: new leaves

G: remaining-return estimate R: reward γ: discount

Continuous actions and progressive widening

A heavily visited node can support more candidate actions

$$m(s) = \max\{1, \lceil k_a N(s)^{\beta} \rceil\}, \qquad k_a > 0, \quad 0 < \beta < 1$$

- Below this limit, we propose a new action; otherwise we select an existing one
- Sublinear growth leaves time to evaluate each candidate repeatedly

$m(s)$: allowed action children $N(s)$: state visits k_a, β: widening parameters
Couëtoux et al., Continuous Upper Confidence Trees

AlphaZero: learning and planning improve each other

Learning improves planning

- A learned policy supplies action priors that focus MCTS
- A learned value function evaluates leaves without finishing every game

Planning improves learning

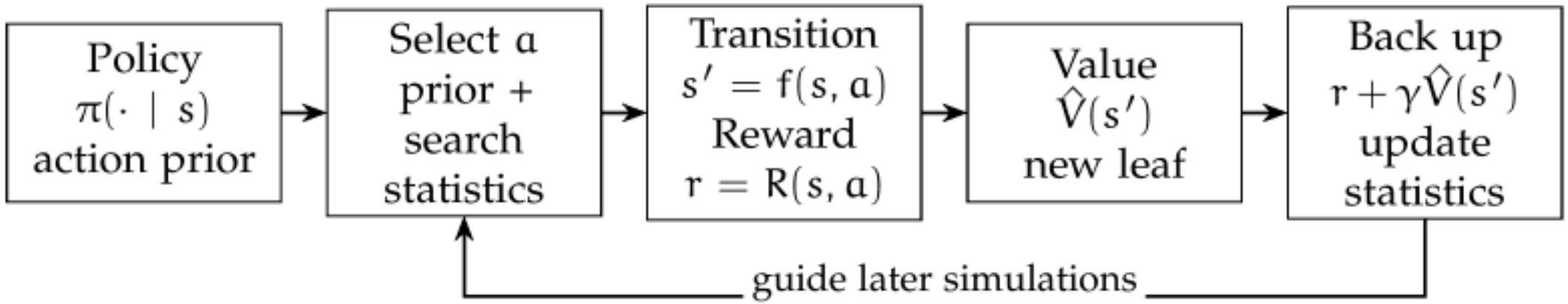
- Search visit counts provide improved policy-training targets
- Outcomes of search-driven self-play games train the value function

AlphaZero uses an exact transition model: it is given the game rules

Silver et al., Mastering Chess and Shogi by Self-Play with a General RL Algorithm

MuZero: learning the dynamics too

An encoder maps observation history to a latent search state

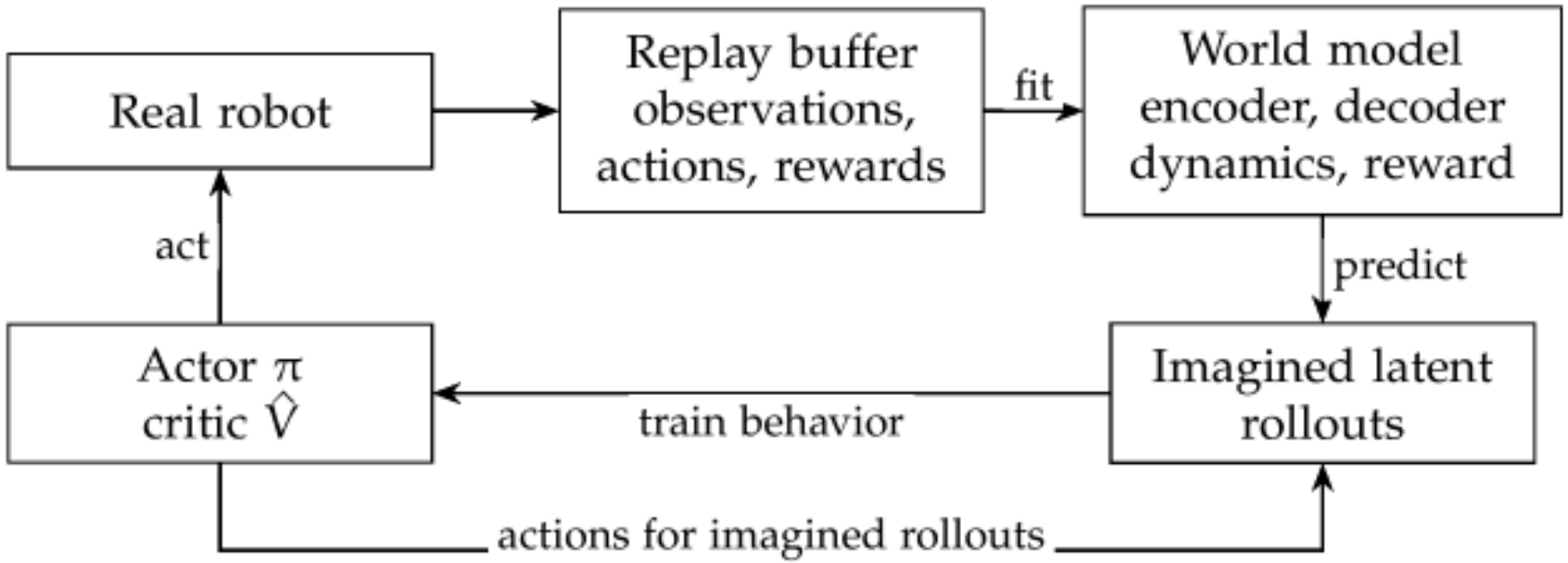


Rewards and leaf values are backed up to the root
Observed rewards, return targets, and search policies train the networks
Latent dynamics support planning without reconstructing observations

Schrittwieser et al., Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model

DayDreamer: learning through imagined experience

Dreamer trains an actor and critic inside a learned world model
DayDreamer applies this loop to physical robots



Wu et al., DayDreamer: World Models for Physical Robot Learning

Dreamer: model, behavior, and execution

- **World model:** an encoder estimates latent state from observations and history; dynamics and reward heads predict its evolution
- **Model training:** reconstruction, reward prediction, and agreement between inferred and predicted latent states
- **Behavior training:** imagined rewards and bootstrapped values train the critic; the actor maximizes predicted returns
- **Execution:** update the latent state and call the actor, without online tree search

For continuous actions, actor gradients pass through the dynamics
World-model parameters stay fixed during behavior updates

Other classes to take

- **6.8200: Sensorimotor Learning**
  Pulkit Agrawal
  Learning for control, imitation, learned models, and robot systems

- **6.7920: Reinforcement Learning: Foundations and Methods**
  Cathy Wu and Munther Dahleh
  Dynamic programming, TD learning, policy optimization, and foundations

- **6.4110: Representation, Inference, and Reasoning in AI**
  Leslie Pack Kaelbling
  Representations, inference, search, MDPs, and POMDPs

- **9.522: Statistical RL and Decision Making**
  Sasha Rakhlin
  Bandits, exploration, function approximation, and sample efficiency