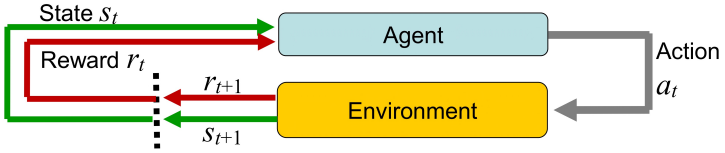
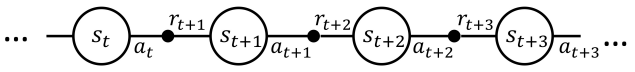
I Introduction

Reinforcement Learning

Goal: Learn from interaction with the environment while only getting a sparse feedback in form of a reward

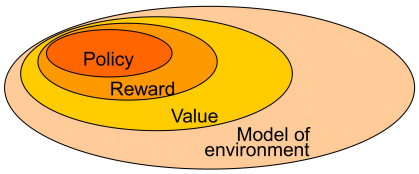




Components:

State: Observable state at discrete times

Action: Action to change the state of the environment
→ only agent changes state



Reward: Feedback from the environment about the last taken action

Value: value functions that evaluate an action w.r.t a goal and a policy

$V(s)$: value of a state s

$Q(s,a)$: value of taking action a in state s

Policy: A function mapping from states to actions determining the behavior of the agent.

$$\pi_t(s,a) = \text{probability that } a_t = a \text{ when } s_t = s$$

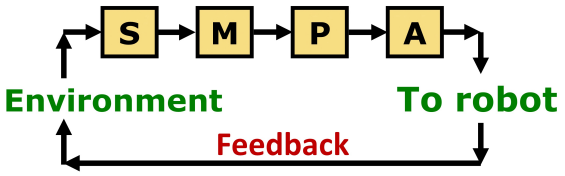
Greedy policy: A policy that maximizes the reward

Environment: Models the task to solve and the rewards to receive

Robot Learning

Apply machine learning paradigms to improve the capabilities of robots.

SMPA Architecture



Abstract model of information flow and control of a robot.

Sensory System: Perceive the state of the environment
→ subject to noise

Modeling System: Build an internal model of the environment

Inputs:
- sensor data
- previous states
- prior knowledge

Planning: Generate an adequate set of actions dependent on:
- state
- goal
- underlying behavior model

Acting: Motor control that performs the dictated action
→ cope with imprecise actuators and actuator limits
→ unforeseen disturbances

Exploration vs. Exploitation

We want to exploit our knowledge to improve action selection but simply taking the greedy action won't explore the action space to find better actions:
→ reduce exploration over the course of training

$$a_t^* = \arg\max_a Q_t(a)$$

$$a_t = a_t^* \Rightarrow \text{exploitation}$$

$$a_t \neq a_t^* \Rightarrow \text{exploration}$$

Action Selection

Methods to implement exploration and exploitation

Greedy: $a_t = a_t^* = \arg\max_a Q_t(a)$

Epsilon Greedy: $a_t = \begin{cases} a_t^* & \text{with probability } 1-\varepsilon \\ \text{random action with probability } \varepsilon \end{cases}$

Softmax: Choose action a with prob.: $\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}}$

Feedback Paradigms

Evaluative: feedback purely depends on taken action
→ reinforcement learning

Instructive: feedback does not depend on the action
→ give correct action instead
→ supervised learning

Associative: Map input to output and learn best output for given input

Non-associative: Learn one best output

II Tabular Solution Methods

Multi-arm Bandits

Problem: Repeatedly choose one out of n actions (play)

Reward: After each play a_t a reward r_t is received from a distribution with: $E\langle r_t \mid a_t \rangle = Q^*(a_t)$

Objective: Approximate optimal policy Q^* to find best action to choose

Non-stationary: The optimal action values change over time

Action Value: $Q_t(a) = \frac{r_1 + r_2 + \ldots + r_{k_a}}{k_a}$

→ sample average over all rewards received when taking action a

Recursive update: $Q_{k+1} = Q_k + \frac{1}{k+1}[r_{k+1} - Q_k]$

Derivation:

$$\begin{aligned} Q_{k+1} &= \frac{1}{k+1}\sum_{i=1}^{k+1} r_i \\ &= \frac{1}{k+1}\left(r_{k+1} + \sum_{i=1}^{k} r_i\right) \\ &= \frac{1}{k+1}(r_{k+1} + kQ_k + Q_k - Q_k) \\ &= Q_k + \frac{1}{k+1}[r_{k+1} - Q_k] \end{aligned}$$

Non-stationary: Exponential weighting to give more recent rewards more influence:

$$Q_{k+1} = Q_k + \alpha[r_{k+1} - Q_k]$$

for *constant* α, $0 < \alpha \leq 1$

$$= (1-\alpha)^k Q_0 + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} r_i$$

exponential, recency-weighted average

Derivation:

$$\begin{aligned} Q_{k+1} &= Q_k + \alpha\left[R_k - Q_k\right] \\ &= \alpha R_k + (1-\alpha)Q_k \\ &= \alpha R_k + (1-\alpha)\left[\alpha R_{k-1} + (1-\alpha)Q_{k-1}\right] \\ &= \alpha R_k + (1-\alpha)\alpha R_{k-1} + (1-\alpha)^2 Q_{k-1} \\ &= \alpha R_k + (1-\alpha)\alpha R_{k-1} + (1-\alpha)^2 \alpha R_{k-2} + \\ &\qquad \cdots + (1-\alpha)^{k-1}\alpha R_1 + (1-\alpha)^k Q_1 \\ &= (1-\alpha)^k Q_1 + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} R_i. \end{aligned}$$

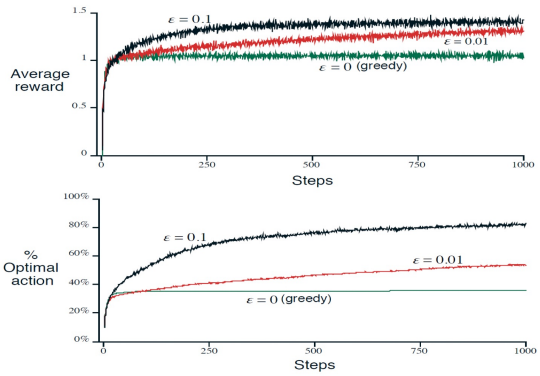
Convergence if: $\sum_{k=1}^{\infty} \alpha_k(a) = \infty$ and $\sum_{k=1}^{\infty} \alpha_k^2(a) < \infty$

Satisfied for: $\alpha_k = \frac{1}{k}$ but not for fixed α

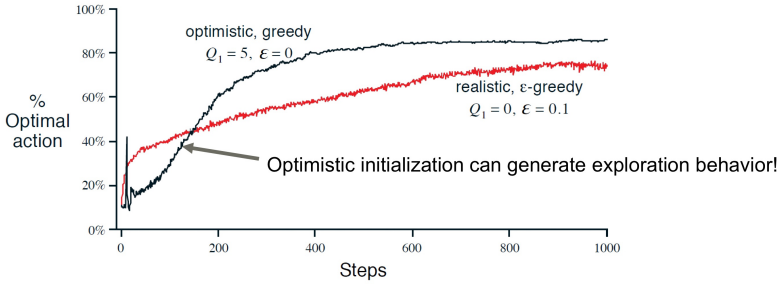
Convergence: Follow law of large numbers: $\lim_{k_a \to \infty} Q_t(a) = Q^*(a)$

Evaluation:

Epsilon Greedy:



Initial values: Set initial action values to optimistic beliefs



Binary Bandit

Bandit task with only two actions and rewards:

$a_t = 1$ or $a_t = 2$ $\quad r_t = success$ or $r_t = failure$

Learning Automata:

Supervised: $d_t = \begin{cases} a_t & \text{if } success \\ \text{the other action} & \text{if } failure \end{cases}$

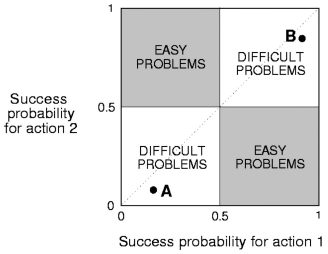
→ choose action that was d_t most often

Reward: L_{R-I} (Linear, reward-inaction)

On $success$: $\pi_{t+1}(a_t) = \pi_t(a_t) + \alpha(1 - \pi_t(a_t))$ $\quad 0 < \alpha < 1$

(the other action probs. are adjusted to still sum to 1)

On $failure$: no change

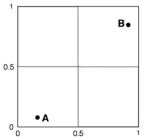


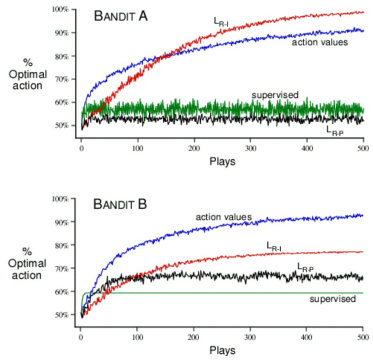
Reward-penalty: On $success$: $\pi_{t+1}(a_t) = \pi_t(a_t) + \alpha(1 - \pi_t(a_t))$ $\quad 0 < \alpha < 1$

(the other action probs. are adjusted to still sum to 1)

On $failure$: $\pi_{t+1}(a_t) = \pi_t(a_t) + \alpha(0 - \pi_t(a_t))$ $\quad 0 < \alpha < 1$

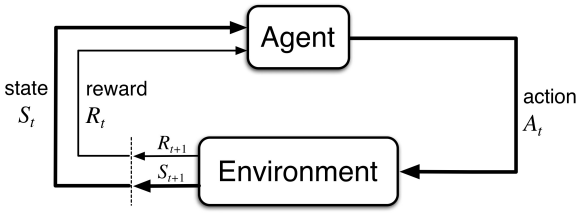
Evaluation:





Finite Markov Decision Processes

Goal: Maximize the cumulative reward (return) over the long run by approximating the optimal value functions.



Model: The agent-environment interaction is modeled as a Markov Decision Process (MDP)

Markov property:
$$\Pr\{R_{t+1}=r, S_{t+1}=s' \mid S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\}$$

State and action sets:
$$S, A(s)\,; s \in S$$

Environment dynamics:
$$p(s', r|s, a) = \Pr\{R_{t+1}=r, S_{t+1}=s' \mid S_t, A_t\}$$

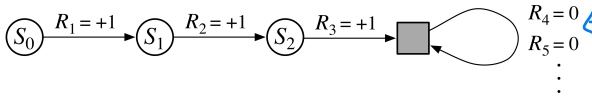
Transition probabilities:
$$p(s'|s,a) = \Pr\{S_{t+1}=s' \mid S_t=s, A_t=a\} = \sum_{r\in\mathcal{R}} p(s', r|s, a)$$

Reward expectation:

State-action:
$$r(s,a) = \mathbb{E}[R_{t+1} \mid S_t=s, A_t=a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s', r|s, a)$$

State-action-next state:
$$r(s,a,s') = \mathbb{E}[R_{t+1} \mid S_t=s, A_t=a, S_{t+1}=s'] = \frac{\sum_{r\in\mathcal{R}} r\, p(s', r|s, a)}{p(s'|s,a)}.$$

Episodic Task: Interaction breaks naturally into episodes
→ finite time horizon



← absorbing state with reward 0

Continuing Task: Interaction is not episodic
→ infinite time horizon

Return:
$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

State value function:
$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t=s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t=s\right]$$

→ expected return starting from the given state dependent on the policy

Action value function:
$$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t=s, A_t=a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t=s, A_t=a\right].$$

→ expected return starting from the given state performing the given action and then following the policy

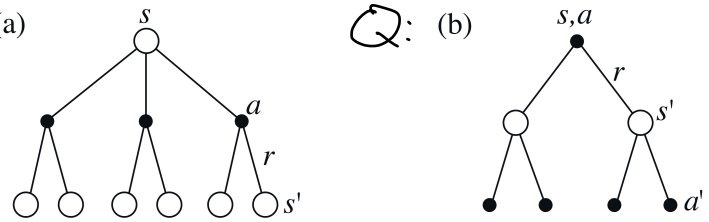
Bellman equation:
$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t=s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t=s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_t=s\right] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\left[r + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_{t+1}=s'\right]\right] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right], \qquad (3.12)
\end{aligned}$$

→ recursive formulation
→ relationship between state values

Backup Diagrams

Diagrams that visualize the dependencies between values and how they are update in the backup operation.

MDP: V: (a) Q: (b)



Optimal value functions

Partial ordering of policies: $\pi \geq \pi'$ if and only if $V^{\pi}(s) \geq V^{\pi'}(s)$ for all $s \in S$

Optimal policy: Larger or equal than all others

→ any policy that is greedy w.r.t v^*

From state value:

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_*(s')\Big] \end{aligned}$$

From action value:

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\Big|\; S_t = s, A_t = a\Big] \\ &= \sum_{s',r} p(s', r|s, a)\Big[r + \gamma \max_{a'} q_*(s', a')\Big], \end{aligned}$$

Optimal state value: $v_*(s) = \max_\pi v_\pi(s)$

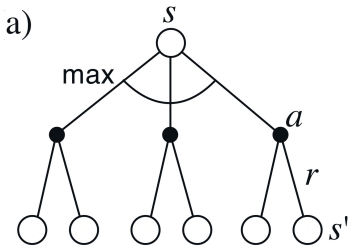
Optimal action value: $q_*(s,a) = \max_\pi q_\pi(s,a)$

Bellman Optimality:

→ for finite MDPs unique solution independent of the policy

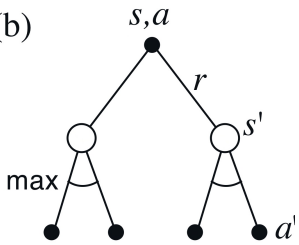
State value:

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi^*}[G_t \mid S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi^*}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \;\middle|\; S_t = s, A_t = a\right] \\ &= \max_a \mathbb{E}_{\pi^*}\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \;\middle|\; S_t = s, A_t = a\right] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s', r|s, a)\big[r + \gamma v_*(s')\big]. \end{aligned}$$



Action value:

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\Big|\; S_t = s, A_t = a\Big] \\ &= \sum_{s',r} p(s', r|s, a)\Big[r + \gamma \max_{a'} q_*(s', a')\Big]. \end{aligned}$$



Dynamic Programming

Methods to solve the problem of computing optimal value functions.

- → assumes perfect model of the environment
- → computational expensive (exponential many state variables)
- → theoretical basis

Sweep: Starting from the last state backup through the state trajectory and update the value functions

$$V_0 \leftarrow V_1 \leftarrow \cdots \leftarrow V_k \leftarrow V_{k+1} \leftarrow \cdots \leftarrow V^\pi$$

Policy Evaluation

Goal: For a given policy compute the state values

Backup function:

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_k(s')\Big] \end{aligned}$$

Algorithm:

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

Policy Improvement

Goal: For an updated value function improve the policy to choose the action maximizing the return

- → decide whether it is better to select action a in state s and following the policy afterwards or not

Action value:

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big]. \end{aligned}$$

- → used to make the decision

Policy Improvement Theorem:

Let π and π' be two policies such that for all $s \in S$:

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s)$$

Then π' is a better policy than π, i.e. for all $s \in S$:

$$V^{\pi'}(s) \geq V^\pi(s)$$

Proof:

$$\begin{aligned} v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2})] \mid S_t = s] \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \mid S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \mid S_t = s] \\ &\vdots \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \mid S_t = s] \\ &= v_{\pi'}(s). \end{aligned}$$

Policy update:

$$\begin{aligned} \pi'(s) &= \arg\max_a q_\pi(s, a) \\ &= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \arg\max_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big], \end{aligned}$$

- → make greedy w.r.t. the action values

Policy Iteration

Iteratively perform policy evaluation and improvement to improve policy and approximate optimal value functions.

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$$

Algorithm:

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $a \leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
   If $a \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return V and π; else go to 2

Value Iteration

Truncate the policy evaluation to a single backup to reduce the computational overhead
→ convergence guarantees are not lost

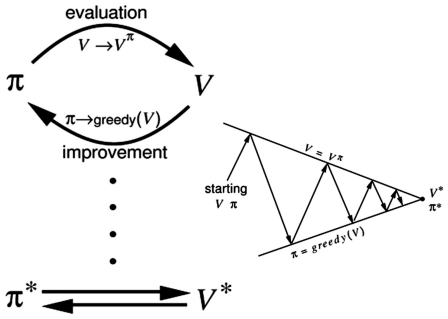
Truncated policy evaluation:

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_k(s')\Big], \end{aligned}$$

↑ only backup one action

Generalized Policy Iteration

Describes any interaction between policy evaluation and improvement independent of their granularity



Asynchronous DP

Instead of backing up long sweeps, backup states at random with the available values
→ convergence guaranteed as long as all states are backed up
→ intelligent selection using agent's experience

Monte Carlo Methods

Monte Carlo methods learn purely from actual or simulated experience.
- → only episodic tasks
- → estimates of state values are independent
- → no bootstrap

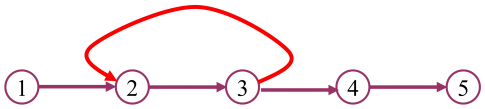
Experience Paradigms

On-line: Use samples from actual experience
- → no model necessary while attaining optimality

Simulated: Simulate experiences
- → no need for full model, only transitions necessary

Monte Carlo Policy Evaluation:

Task:
Goal: learn state value functrion $V^{\pi}(s)$
Given: some number of episodes under π which contain s
Idea: Average returns observed after visits to s



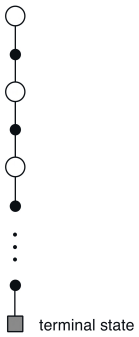
Every-Visit MC: Average return every time state s is visited in an episode

First-Visit MC: Average returns only for the first time state s is visited in an episode

Algorithm:

Initialize:
$\pi \leftarrow$ policy to be evaluated
$V \leftarrow$ an arbitrary state-value function
$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:
Generate an episode using π
For each state s appearing in the episode:
$G \leftarrow$ return following the first occurrence of s
Append G to $Returns(s)$
$V(s) \leftarrow$ average($Returns(s)$)



For action values $Q^{\pi}(s,a)$:

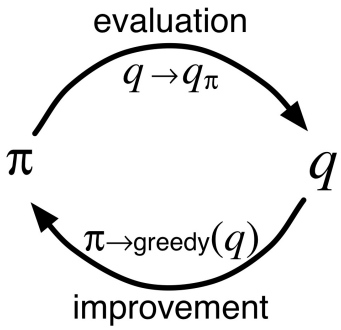
Without a model state values are not sufficient to perform a one step look-ahead to estimate the policy. Thus, estimating the action values becomes useful.
- → converges if every state-action pair is visited

Method:

1. **Exploring states:** Sample random state-action pairs to start from
   - → every pair has a non-zero probability
2. Average return starting from state s and action a

Monte Carlo Control

Iterative value estimation and policy improvement



Convergence:

$$\begin{aligned} q_{\pi_k}(s, \pi_{k+1}(s)) &= q_{\pi_k}(s, \arg\max_a q_{\pi_k}(s,a)) \\ &= \max_a q_{\pi_k}(s,a) \\ &\geq q_{\pi_k}(s, \pi_k(s)) \\ &= v_{\pi_k}(s). \end{aligned}$$

→ policy improvement theorem
→ with $k \to \infty$ & exploring states converges

Evaluation:
1. Update with multiple episode until a given level of performance
2. Alternate between evaluation and improvement every episode

Monte Carlo Paradigms:

On-policy: Evaluate and improve the policy used to make decisions

Off-policy: Evaluate and improve a policy different from the one used to generate episodes

Behavior policy: Generates episodes

Estimation policy: Policy that is learned

ε-greedy policies: Soft policy to ensure that every action is sampled

$$\underbrace{\frac{\varepsilon}{|A(s)|}}_{\text{non-max}} \qquad \underbrace{1-\varepsilon+\frac{\varepsilon}{|A(s)|}}_{\text{greedy}}$$

$\longrightarrow$ $\pi(s,a) \geq \varepsilon / |A(s)| > 0$

Incremental updates:

$$V_n = \frac{\sum_{k=1}^{n} w_k R_k}{\sum_{k=1}^{n} w_k}$$

non-incremental

$$\begin{aligned} V_{n+1} &= V_n + \frac{w_{n+1}}{W_{n+1}}\left[R_{n+1} - V_n\right] \\ W_{n+1} &= W_n + w_{n+1} \\ V_0 &= W_0 = 0 \end{aligned}$$

incremental equivalent

Algorithms:

On-policy with Exploring States

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s,a) \leftarrow$ arbitrary
  $\pi(s) \leftarrow$ arbitrary
  $Returns(s,a) \leftarrow$ empty list

Repeat forever:
  Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ s.t. all pairs have probability > 0
  Generate an episode starting from S_0, A_0, following π
  For each pair s,a appearing in the episode:
    $G \leftarrow$ return following the first occurrence of s,a
    Append G to $Returns(s,a)$
    $Q(s,a) \leftarrow$ average$(Returns(s,a))$
  For each s in the episode:
    $\pi(s) \leftarrow \arg\max_a Q(s,a)$

On-policy with ε-greedy policy

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
$Q(s,a) \leftarrow$ arbitrary
$Returns(s,a) \leftarrow$ empty list
$\pi(a|s) \leftarrow$ an arbitrary ε-soft policy

Repeat forever:
(a) Generate an episode using π
(b) For each pair s, a appearing in the episode:
$G \leftarrow$ return following the first occurrence of s, a
Append G to $Returns(s,a)$
$Q(s,a) \leftarrow \text{average}(Returns(s,a))$
(c) For each s in the episode:
$a^* \leftarrow \arg\max_a Q(s,a)$
For all $a \in \mathcal{A}(s)$:

$$\pi(a|s) \leftarrow \begin{cases} 1-\varepsilon+\varepsilon/|\mathcal{A}(s)| & \text{if } a = a^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}$$

Off-policy

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
$Q(s,a) \leftarrow$ arbitrary
$N(s,a) \leftarrow 0$; Numerator and
$D(s,a) \leftarrow 0$; Denominator of $Q(s,a)$
$\pi \leftarrow$ an arbitrary deterministic policy

Repeat forever:
(a) Select a policy π' and use it to generate an episode:
$s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T, s_T$
(b) $\tau \leftarrow$ latest time at which $a_\tau \neq \pi(s_\tau)$
(c) For each pair s, a appearing in the episode at time τ or later:
$t \leftarrow$ the time of first occurrence of s, a such that $t \geq \tau$
$w \leftarrow \prod_{k=t+1}^{T-1} \frac{1}{\pi'(s_k,a_k)}$
$N(s,a) \leftarrow N(s,a) + wR_t$
$D(s,a) \leftarrow D(s,a) + w$
$Q(s,a) \leftarrow \frac{N(s,a)}{D(s,a)}$
(d) For each $s \in \mathcal{S}$:
$\pi(s) \leftarrow \arg\max_a Q(s,a)$

Importance Sampling

Sample from π' and estimate expectation for state-action values under π

Prob of state trajectory:

$$\prod_{k=t}^{T-1} \pi(a_k|s_k)\, p(s_{k+1}|s_k, a_k)$$

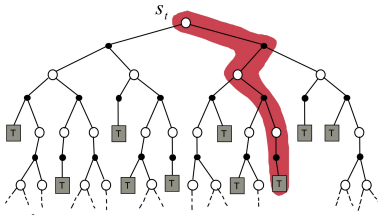
Relative prob:

$$\rho_t^T = \frac{\prod_{k=t}^{T-1} \pi(a_k|s_k) p(s_{k+1}|s_k,a_k)}{\prod_{k=t}^{T-1} \pi'(a_k|s_k) p(s_{k+1}|s_k,a_k)} = \frac{\prod_{k=t}^{T-1} \pi(a_k|s_k)}{\prod_{k=t}^{T-1} \pi'(a_k|s_k)}$$

Temporal-Difference Learning

Combines ideas of Monte Carlo and DP methods. TD methods learn from experience while updating value estimates based on other estimates

MC: $V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)]$



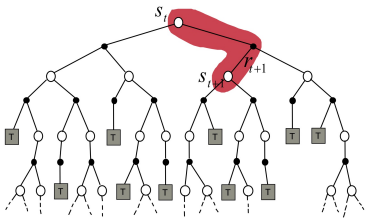
TD Prediction

TD(0):

$$V(S_t) \leftarrow V(S_t) + \alpha\left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]$$

step size; target difference

TD(0):

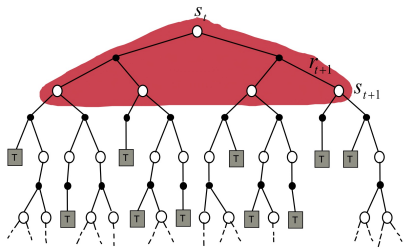


n-step TD: $V_{t+1}(s) = V_t(s) + \Delta_t(s)$

Update: $\Delta_t(S_t) = \alpha\left[G_t^{t+n}(V_t(S_{t+n})) - V_t(S_t)\right]$

Return: $G_t^{t+n}(c) = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_h + \gamma^n c_h$

DP: $V(s_t) \leftarrow E_\pi\{r_{t+1} + \gamma V(s_t)\}$



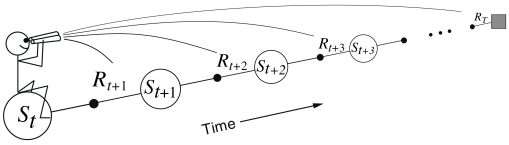
TD(λ): Average rewards from different steps.

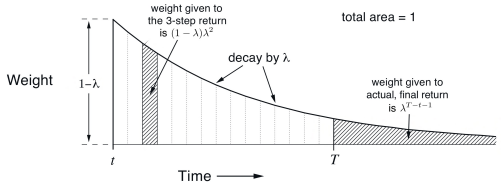
Forward view: $\Delta_t(S_t) = \alpha\left[L_t - V_t(S_t)\right]$

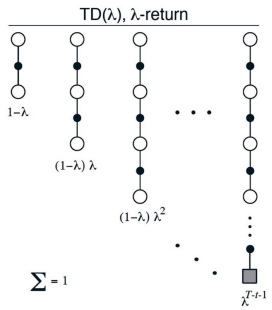
λ-return: $L_t = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} G_t^{t+n}(V_t(S_{t+n})) + \lambda^{T-t-1} G_t$

(G_t^{t+n}: n-step; G_t: final)

TD(λ):







Backward view:

Instead of looking forward, define elegibility traces that represent how recent a state was visited and is eligible to learn from the current error.

Elegibility traces:

If state s is not visited: $E_t(s) = \gamma\lambda E_{t-1}(s)$

If state s is visited:

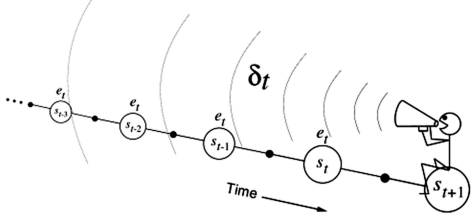
Accumulating traces: $E_t(S_t) = \gamma\lambda E_{t-1}(S_t) + 1$

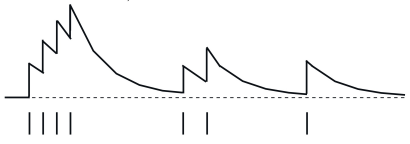
Replacing traces: $E_t(S_t) = 1$

Dutch traces: $E_t(S_t) = (1-\alpha)\gamma\lambda E_{t-1}(S_t) + 1$

Error: $\delta_t = R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)$

Update: $\Delta V_t(s) = \alpha\,\delta_t\,E_t(s)$





Algorithm:

Initialize $V(s)$ arbitrarily (but set to 0 if s is terminal)
Repeat (for each episode):
  Initialize $E(s) = 0$, for all $s \in \mathcal{S}$
  Initialize S
  Repeat (for each step of episode):
    $A \leftarrow$ action given by π for S
    Take action A, observe reward, R, and next state, S'
    $\delta \leftarrow R + \gamma V(S') - V(S)$
    $E(S) \leftarrow E(S) + 1$ (accumulating traces)
    or $E(S) \leftarrow (1-\alpha)E(S) + 1$ (dutch traces)
    or $E(S) \leftarrow 1$ (replacing traces)
    For all $s \in \mathcal{S}$:
      $V(s) \leftarrow V(s) + \alpha\delta E(s)$
      $E(s) \leftarrow \gamma\lambda E(s)$
    $S \leftarrow S'$
  until S is terminal

Equivalence of Views:

$$\underbrace{\sum_{t=0}^{T-1}\Delta V_t^{TD}(s)}_{\text{Backward updates}} = \underbrace{\sum_{t=0}^{T-1}\Delta V_t^{\lambda}(s_t)I_{ss_t}}_{\text{Forward updates}}$$

Algebra shown in Sutton & Barto book

$$\sum_{t=0}^{T-1}\Delta V_t^{TD}(s) = \sum_{t=0}^{T-1}\alpha I_{ss_t}\sum_{k=t}^{T-1}(\gamma\lambda)^{k-t}\delta_k \qquad \sum_{t=0}^{T-1}\Delta V_t^{\lambda}(s_t)I_{ss_t} = \sum_{t=0}^{T-1}\alpha I_{ss_t}\sum_{k=t}^{T-1}(\gamma\lambda)^{k-t}\delta_k$$

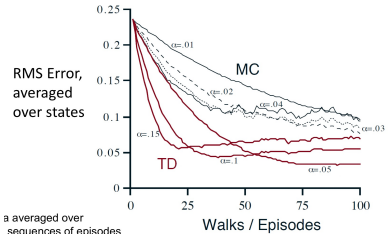
Batch updating: Train completely on a finite amount of data

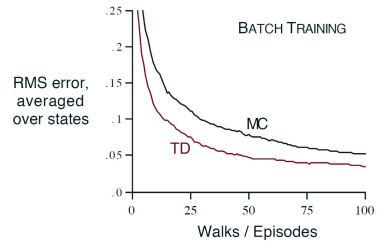
→ process batch and update estimates afterwards
→ better estimates
→ converges to an optimal estimate for small α

Comparison to MC methods:

MC method: Solution minimizes the mean-squared error on the training set

TD(0): Solution is the maximum-likelihood estimate
→ certainty-equivalence estimate
→ converges faster
→ better results





TD(1): Monte Carlo behavior

Advantages

1. TD methods can learn purely from experience
2. TD methods are fully incremental
3. Learning can be done without knowing the final outcome

TD Control

Sarsa: On-policy using TD(0) prediction and an ε-greedy

Update function:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Sarsa(λ): Keep eligible traces for state-action pairs and perform Sarsa

Traces:

$$E_t(s,a) = \gamma\lambda E_{t-1}(s,a) + I_{sS_t} I_{aA_t} \qquad \text{(accumulating)}$$
$$E_t(s,a) = (1-\alpha)\gamma\lambda E_{t-1}(s,a) + I_{sS_t} I_{aA_t} \qquad \text{(dutch)}$$
$$E_t(s,a) = (1 - I_{sS_t} I_{aA_t})\gamma\lambda E_{t-1}(s,a) + I_{sS_t} I_{aA_t} \qquad \text{(replacing)}$$

Update:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\,\delta_t\,E_t(s,a)$$
$$\text{with:} \quad \delta_t = R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1}) - Q_t(S_t, A_t)$$

Algorithm:

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat (for each episode):
  $E(s,a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
  Initialize S, A
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $\delta \leftarrow R + \gamma Q(S',A') - Q(S,A)$
    $E(S,A) \leftarrow E(S,A) + 1$ (accumulating traces)
    or $E(S,A) \leftarrow (1-\alpha)E(S,A) + 1$ (dutch traces)
    or $E(S,A) \leftarrow 1$ (replacing traces)
    For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta E(s,a)$
      $E(s,a) \leftarrow \gamma\lambda E(s,a)$
    $S \leftarrow S'$; $A \leftarrow A'$
  until S is terminal

Q-Learning: Update using the best successor action instead of the one dictated by the policy

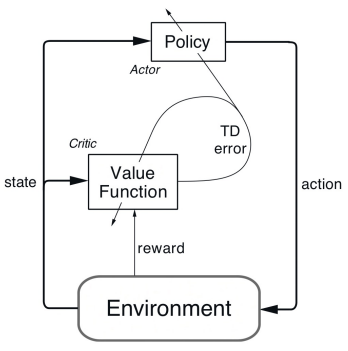
Update function:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$;
  until S is terminal

Actor-critic Methods

Policies (actor) are learned directly from the TD error given by an estimated state-value function (critic).



TD Error:

$$\delta_t = R_{t+1} + \gamma V_t(S_{t+1}) - V(S_t).$$

Policy:

$$\pi_t(a|s) = \Pr\{A_t = a \mid S_t = s\} = \frac{e^{H_t(s,a)}}{\sum_b e^{H_t(s,b)}}$$

Update:

$$H_{t+1}(S_t, A_t) = H_t(S_t, A_t) + \beta\delta_t$$

III Function Approximation

Motivation:

1. Continuous state and action spaces
2. Better generalization to unseen states

Gradient-Descent Methods

Use gradient descent to approximate the state values

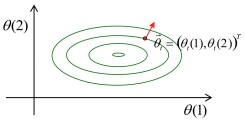
Parameters: $\theta_t = (\theta_t(1), \theta_t(2), \ldots, \theta_t(n))$ (w)

→ represent state values through parameters

Error: $MSE(\theta_t) = \sum_{s \in S} P(s)\left[V^{\pi}(s) - V_t(s)\right]^2$

→ $V_t(s)$ might be unknown

$$\nabla_{\vec{\theta}} f(\vec{\theta}_t) = \left(\frac{\partial f(\vec{\theta}_t)}{\partial \theta(1)}, \frac{\partial f(\vec{\theta}_t)}{\partial \theta(2)}, \ldots, \frac{\partial f(\vec{\theta}_t)}{\partial \theta(n)}\right)^T$$



Iteratively move down the gradient:

$$\vec{\theta}_{t+1} = \vec{\theta}_t - \alpha \nabla_{\vec{\theta}} f(\vec{\theta}_t)$$

Update: $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\left[V_t - \hat{v}(S_t, \mathbf{w}_t)\right]\nabla \hat{v}(S_t, \mathbf{w}_t)$

TD(λ) update: $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\left[G_t^{\lambda} - \hat{v}(S_t, \mathbf{w}_t)\right]\nabla \hat{v}(S_t, \mathbf{w}_t)$

TD(λ) Gradient-Descent:

Initialize $\vec{\theta}$ arbitrarily
Repeat (for each episode):
  $\vec{e} = 0$ // eligibility trace for parameters
  $s \leftarrow$ initial state of episode
  Repeat (for each step of episode):
    $a \leftarrow$ action given by π for s
    Take action a, observe reward, r, and next state, s'
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $\vec{e} \leftarrow \gamma\lambda\vec{e} + \nabla_{\vec{\theta}} V(s)$
    $\vec{\theta} \leftarrow \vec{\theta} + \alpha\delta\vec{e}$
    $s \leftarrow s'$
  until s is terminal

Linear Methods

Special case of gradient descent methods where the values are linearly dependent on the parameters

Function:

$$\vec{\phi}_s = (\phi_s(1), \phi_s(2), \ldots, \phi_s(n))^T$$

$$V_t(s) = \vec{\theta}_t^T \vec{\phi}_s = \sum_{i=1}^{n} \theta_t(i)\phi_s(i)$$

Gradient: $\nabla_{\vec{\theta}} V_t(s) = \vec{\phi}_s$

Convergence: Linear gradient descent TD(λ) converges:

- Step size decreases appropriately
- On-line sampling
  (states sampled from the on-policy distribution)
- Converges to parameter vector $\vec{\theta}_{\infty}$ with property:

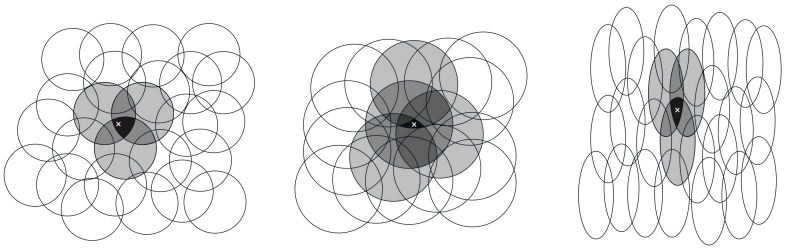
$$MSE(\vec{\theta}_{\infty}) \leq \frac{1-\gamma\lambda}{1-\gamma} MSE(\vec{\theta}^*)$$

(Tsitsiklis & Van Roy, 1997)

$\vec{\theta}^*$: best parameter vector

Coarse Coding: Encode the input into a feature vector to enable interaction of the state values in the linear method

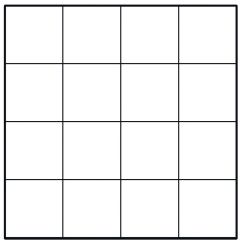
→ binary features
→ overlapping
→ shape determines "range" of generalization of input to other inputs
→ encodes prior domain knowledge

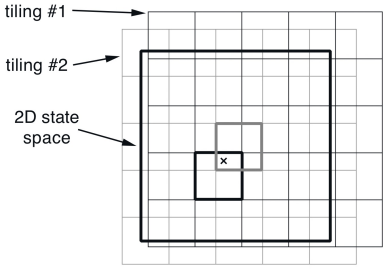


a) Narrow generalization b) Broad generalization c) Asymmetric generalization

Tile Coding: Special case of coarse coding splitting the input space into partitions (tiles)

→ constant number of features
→ easy computation of linear functions

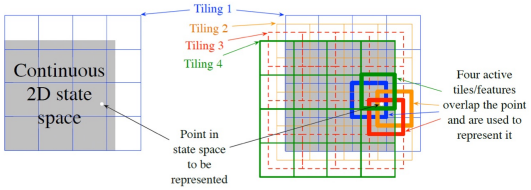


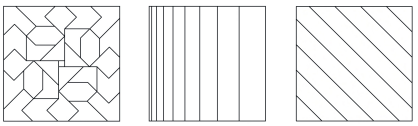


Shape of tiles ⇒ Generalization

#Tilings ⇒ Resolution of final approximation

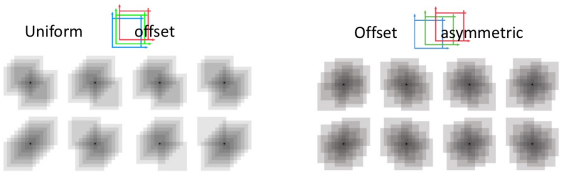
Examples: Four Tilings





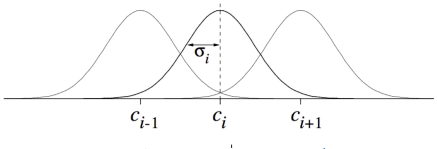
a) Irregular b) Log stripes c) Diagonal stripes

Generalization

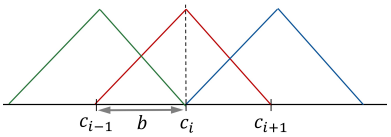


Continuous features: Instead of a 0-1 encoding use a smooth function that indicates how much a feature is present in the input

RBF: $\phi_s(i) = \exp\left(-\frac{\|s - c_i\|^2}{2\sigma_i^2}\right)$



Piecewise linear: $\phi_s(i) = \max\left(0, 1 - \frac{|s - c_i|}{b}\right)$



Curse of Dimensionality

With high-dimensional state space tile coding might become impractical. The complexity of the target function increases with the size of the state space, making it infeasible to be approximated.

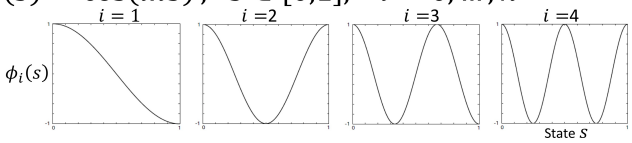
Kanerva Coding: Use prototypes in the input space to control the size of the state space

Polynomial Embedding: $\phi_i(\boldsymbol{s}) = \prod_{j=1}^{k} s_j^{\,c_{i,j}}$, $c_{i,j} \in \{0,1,\dots,n\}$

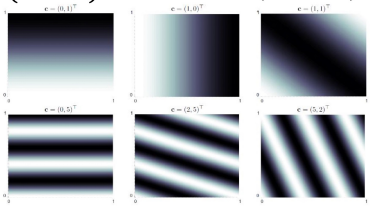
4D representation $(1, s_1, s_2, s_1 s_2)$ or

9D representation $(1, s_1, s_2, s_1 s_2, s_1^2, s_2^2, s_1 s_2^2, s_1^2 s_2, s_1^2 s_2^2)$

Fourier Embedding: 1D: $\phi_i(s) = \cos(i\pi s)$, $s \in [0,1]$, $i = 0,\dots,n$



2D: $\phi_i(s) = \cos\left(\pi \boldsymbol{s}^T \boldsymbol{c}^i\right)$, $s_i \in [0,1]$, $\boldsymbol{c}^i = (c_1{}^i, \dots, c_k{}^i)^T, c_k{}^i \in \{0,\dots,n\}$, j = 1, …, k



Control with Function Approximation

Gradient-descent update:

Forward view:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\Big[Q_t - \hat{q}(S_t, A_t, \mathbf{w}_t)\Big]\nabla\hat{q}(S_t, A_t, \mathbf{w}_t)$$

Backward view:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\,\delta_t\,\mathbf{e}_t,$$

with:
$$\delta_t = R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)$$
$$\mathbf{e}_t = \gamma\lambda\mathbf{e}_{t-1} + \nabla\hat{q}(S_t, A_t, \mathbf{w}_t)$$

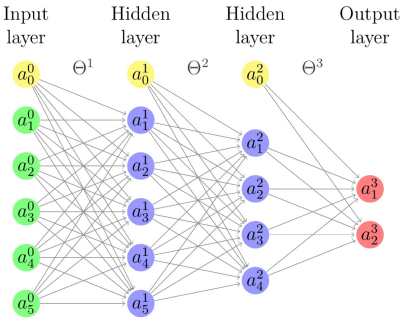
Watkins' Q(λ):

Initialize $\vec{\theta}$ arbitrarily
Repeat (for each episode):
  $\vec{e} = \vec{0}$
  $s, a \leftarrow$ initial state and action of episode
  $\mathcal{F}_a \leftarrow$ set of features present in s, a // binary features
  Repeat (for each step of episode):
    For all $i \in \mathcal{F}_a$: $e(i) \leftarrow e(i) + 1$
    Take action a, observe reward, r, and next state, s
    $\delta \leftarrow r - \sum_{i\in\mathcal{F}_a} \theta(i)$
    For all $a \in \mathcal{A}(s)$:
      $\mathcal{F}_a \leftarrow$ set of features present in s, a
      $Q_a \leftarrow \sum_{i\in\mathcal{F}_a} \theta(i)$
    $\delta \leftarrow \delta + \gamma \max_a Q_a$ // greedy
    $\vec{\theta} \leftarrow \vec{\theta} + \alpha\delta\vec{e}$
    With probability $1-\varepsilon$: // soft policy
      For all $a \in \mathcal{A}(s)$:
        $Q_a \leftarrow \sum_{i\in\mathcal{F}_a} \theta(i)$
      $a \leftarrow \arg\max_a Q_a$
      $\vec{e} \leftarrow \gamma\lambda\vec{e}$
    else
      $a \leftarrow$ a random action $\in \mathcal{A}(s)$
      $\vec{e} \leftarrow 0$
  until s is terminal

Sarsa(λ):

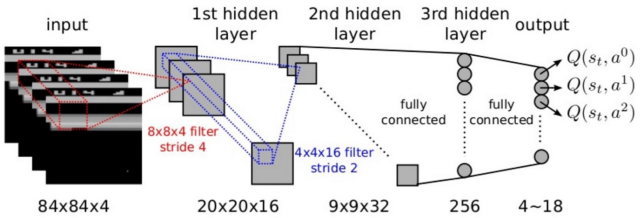
Initialize $\vec{\theta}$ arbitrarily
Repeat (for each episode):
  $\vec{e} = \vec{0}$
  $s, a \leftarrow$ initial state and action of episode
  $\mathcal{F}_a \leftarrow$ set of features present in s, a
  Repeat (for each step of episode):
    For all $i \in \mathcal{F}_a$:
      $e(i) \leftarrow e(i) + 1$ (accumulating traces)
      or $e(i) \leftarrow 1$ (replacing traces)
    Take action a, observe reward, r, and next state, s
    $\delta \leftarrow r - \sum_{i\in\mathcal{F}_a} \theta(i)$
    With probability $1-\varepsilon$:
      For all $a \in \mathcal{A}(s)$:
        $\mathcal{F}_a \leftarrow$ set of features present in s, a
        $Q_a \leftarrow \sum_{i\in\mathcal{F}_a} \theta(i)$
      $a \leftarrow \arg\max_a Q_a$
    else
      $a \leftarrow$ a random action $\in \mathcal{A}(s)$
      $\mathcal{F}_a \leftarrow$ set of features present in s, a
      $Q_a \leftarrow \sum_{i\in\mathcal{F}_a} \theta(i)$
    $\delta \leftarrow \delta + \gamma Q_a$
    $\vec{\theta} \leftarrow \vec{\theta} + \alpha\delta\vec{e}$
    $\vec{e} \leftarrow \gamma\lambda\vec{e}$
  until s is terminal

Function Approximation with Neural Networks



Learning directly on pixels:

Convolutional NN



Quadratic loss

$$Loss = \frac{1}{2}.\underbrace{[r + max_{a_{t+1}}(Q(s_{t+a}, a_{t+1}; \theta_{t-1}))}_{\text{target}} - \underbrace{Q(s, a; \theta)}_{\text{prediction}}]^2$$

Deep Q-Learning:

Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

IV Policy Gradient Methods

Motivation:
1. Representing the state-action values might be more complex than learning the policy directly
2. States are only partially observed
3. For large problems this requires approximations

Parametrized Policy:

A policy defined through a family of functions with learnable parameters.

Linear functions:

$$\pi_\theta(s) = \arg\max_a \hat{Q}_\theta(s,a)$$

with: $\hat{Q}_\theta(s,a) = \theta_0 + \theta_1 f_1(s,a) + \theta_2 f_2(s,a) + \ldots + \theta_n f_n(s,a)$

→ not differentiable → no gradient

probabilistic policies:

$$\pi_\theta(s,a) = \Pr(a \mid s) = \frac{\exp(\hat{Q}_\theta(s,a))}{\sum_{a' \in A} \exp(\hat{Q}_\theta(s,a'))}$$

→ continuous function
→ differentiable

Log gradients:

$$\frac{\partial \log(\pi_\theta(s,a))}{\partial \theta_i} = f_i(s,a) - \sum_{a'} \pi_\theta(s,a') f_i(s,a')$$

→ required for many algorithms
→ assumes linear value function

General Algorithm:

1. Select the family of parametrized policies π_θ
2. Compute the gradient of the return w.r.t. θ
3. Update parameters: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
4. Repeat until convergence

Gradient Estimation:

Let: $J(\theta)$ expected discounted reward for policy π_θ
→ gradient can typically not be computed in closed form

Empirical Gradient Estimation:

1. For each set of parameters run M trials
   → gradient from single trial is too noisy
2. Estimate gradient for each trial:

   Compute: $\frac{\partial J(\theta)}{\partial \theta_i} = \lim_{\varepsilon \to 0} \frac{J(\theta_1, \ldots, \theta_{i-1}, \theta_i + \varepsilon, \theta_{i+1}, \ldots, \theta_n) - J(\theta)}{\varepsilon}$ for small ε
3. Average gradients

Complexity: $O(M(n+1))$

→ impractical in most situations

Stochastic Gradient Estimation:

1. Start with policy: π described by $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)$
2. Generate M random policies: $\pi_i = (\theta_1 + \Delta_1, \ldots, + \theta_n + \Delta_n\}$
   Δ_i is $-\varepsilon, 0, or + \varepsilon$
3. Estimate gradients by averaging results over the same Δ_i

Likelihood Ratio Gradient Estimation

Let: F: real-valued function over D
X: random variable over D distributed according to $P_\theta(x)$

Expectation: $J(\theta) = E[F(X)|\theta] = \sum_{x \in D} P_\theta(x) F(x)$

Gradient:

$$\begin{aligned}\nabla_\theta J(\theta) &= \sum_{x \in D} (\nabla_\theta P_\theta(x)) F(x) \\ &= \sum_{x \in D} P_\theta(x) \frac{(\nabla_\theta P_\theta(x))}{P_\theta(x)} F(x) \quad \text{//derivative of ln(x): 1/x} \\ &= \sum_{x \in D} P_\theta(x) \underbrace{\nabla_\theta \ln(P_\theta(x))}_{z_\theta(x)} F(x) = E[z_\theta(X) F(X)]\end{aligned}$$

Approximation: $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{j=1}^{N} z_\theta(x_j) F(x_j)$

x_j is the j'th sample of X

→ we only need to sample x and compute z_θ

Application to Policy Gradient

Approximation:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{j=1}^{N}\sum_{t=1}^{T_j}\left(\nabla_\theta \log \pi_\theta(s_{j,t}, a_{j,t})\right)\sum_{k=t+1}^{T_j} r(s_{j,k})$$

- N: # of trajectories of current policy
- T_j: length of trajectory j
- $\nabla_\theta \log \pi_\theta(s_{j,t}, a_{j,t})$: Direction to move parameters in order to increase the probability that policy selects a_{jt} in state s_{jt}
- $\sum_{k=t+1}^{T_j} r(s_{j,k})$: Observed reward after taking a_{jt} in state s_{jt}

Derivation:

1. Define:

$X = (s_1, a_1, s_2, a_2, \ldots, s_T)$ as sequence of T states and T-1 actions

distributed according to $P_\theta(s_1, a_1, \ldots, s_T) = \prod_{t=1}^{T-1} \pi_\theta(s_t, a_t) p(s_t, a_t, s_{t+1})$

Total reward: $F(X) = \sum_{t=1}^{T} r(s_t)$

Expected reward: $J(\theta) = E[F(X)|\theta] = E\left[\sum_{t=1}^{T} r(s_t)\right]$

2. Input into the definition: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{j=1}^{N} z_\theta(x_j) F(x_j)$

$$z_\theta(x_t) = \nabla_\theta \log(P_\theta(x)) = \nabla_\theta \log\left(\prod_{t=1}^{T-1} \pi_\theta(s_t, a_t) p(s_t, a_t, s_{t+1})\right)$$

$$= \nabla_\theta \sum_{t=1}^{T-1}\left[\log(\pi_\theta(s_t, a_t)) + \log(p(s_t, a_t, s_{t+1}))\right]$$

$$= \sum_{t=1}^{T-1} \nabla_\theta \log(\pi_\theta(s_t, a_t))$$ ← model free

3. Put together and shift index
→ action a_t does not influence future rewards

$$\sum_{t=1}^{T-1} \nabla_\theta \log \pi_\theta(s_t, a_t) \sum_{k=t+1}^{T} r(s_k) \longrightarrow \nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{j=1}^{N}\sum_{t=1}^{T_j}\left(\nabla_\theta \log \pi_\theta(s_{j,t}, a_{j,t})\right)\sum_{k=t+1}^{T_j} r(s_{j,k})$$

(same annotations as above)

Bayesian Optimization

Add a prior on the parameters to guide the choice of the parameters to maximize the return.

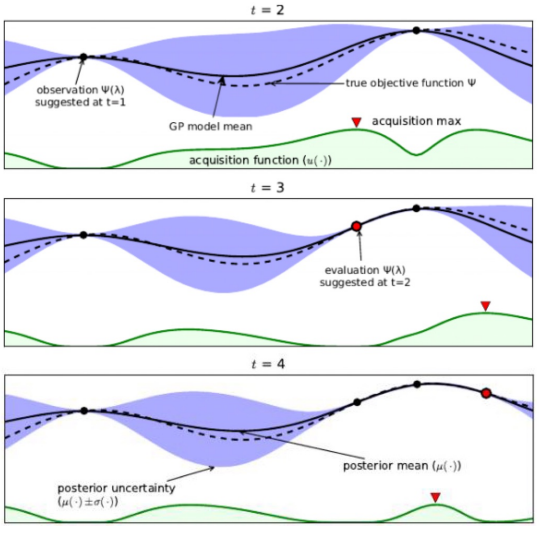
Selection:

Maximum Mean (MM)
- Selects the points which has the highest output mean
- Purely exploitative

Maximum Upper bound Interval (MUI)
- Select point with highest 95% upper confidence bound
- Purely explorative approach

Maximum Probability of Improvement (MPI)
- It computes the probability that the output is more than (1+m) times of the best current observation, m>0.
- Explorative and Exploitative

Maximum Expected of Improvement (MEI)
- Similar to MPI but parameter free
- It simply computes the expected amount of improvement after sampling at any point



Basic Policy Gradient Algorithm

→ overhead in storage for episodes
→ small number of updates per episode

Repeat until stopping condition

1. Execute π_θ for N episodes to get set of state, action, reward sequences
2. $$\nabla_\theta \leftarrow \frac{1}{N}\sum_{j=1}^{N}\sum_{t=1}^{T_j}\left(\nabla_\theta \log \pi_\theta(s_{j,t}, a_{j,t}) \sum_{k=t+1}^{T_j} r(s_{j,k})\right)$$
3. $$\theta \leftarrow \theta + \alpha \nabla_\theta$$

Online Policy Gradient (OLPOMDP)

Initialize θ, set $z=0$ [Baxter & Bartlett, 2000]

Repeat forever

1. Observe state *s*
2. Draw action *a* according to distribution $\pi_\theta(s)$
3. Execute *a* and observe reward *r*
4. $z \leftarrow \beta z + \nabla_\theta \log \pi_\theta(s,a)$ // discounted sum of gradient directions
5. $\theta \leftarrow \theta + \alpha \cdot r \cdot z$

→ step 4 computes eligibility traces

Pegasus: Use a fixed set of start states to eliminate probabilistic behavior and use deterministic optimization

Applications:

Bi-Pedal Walking:

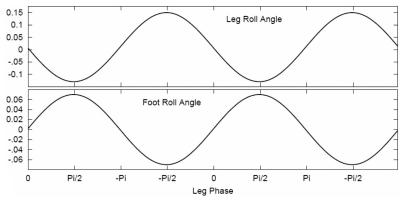
Open Loop Gait Engine:

Target speed vector: $\mathbf{v}_{\mathbf{Robot}} = (v^{\mathrm{x}}_{\mathrm{Robot}}, v^{\mathrm{y}}_{\mathrm{Robot}}, v^{\theta}_{\mathrm{Robot}})$

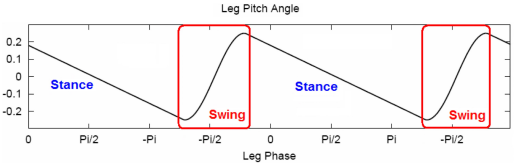
Timing:
Step frequency ψ
Trunk phase $-\pi \leq \phi_{\mathrm{Trunk}} < \pi$
Leg phase $\phi_{\mathrm{Leg}} = \phi_{\mathrm{Trunk}} \pm \pi/2$

Motion primitives:

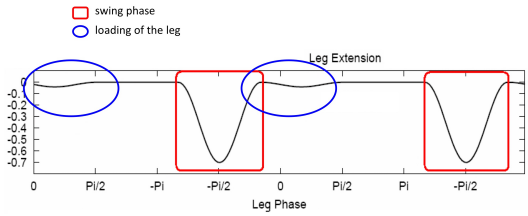
Lateral weight shifting:



Leg swing:



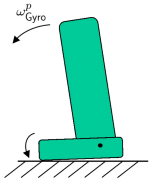
Leg shortening:



Gyroscope Feedback:

$$\theta^r_{\text{FootGyro}} = \theta^r_{\text{Foot}} + \Lambda \cdot K^r \cdot \omega^r_{\text{Gyro}}$$
$$\theta^p_{\text{FootGyro}} = \theta^p_{\text{Foot}} + K^p \cdot \omega^p_{\text{Gyro}}.$$

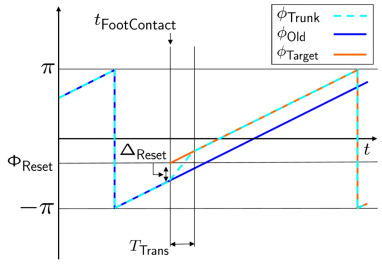
with gains $K^{r/p}$ and leg sign $\Lambda = \pm 1$



→induce torque in the opposite direction of the measured angular velocity

Phase resetting:

Adapt to variable step durations by resetting the trunk phase



RL Setup:

Parameters:

Open-Loop Parameters	
ψ	step frequency
a_{Swing}	swing amplitude of the leg in forward direction
a_{Shift}	amplitude for lateral weight shifting
$\lambda_{\text{FootSwing}}$	partial balance of the leg swing with the foot angle
$\lambda_{\text{FootPitch}}$	general tilt of the robot in sagittal direction
$a_{\text{FootPitch}}$	oscillates the robot in sagittal direction with every step
$\rho_{\text{FootPitch}}$	phase shift of the $a_{\text{FootPitch}}$ oscillation
$\lambda_{\text{LegSpread}}$	offset to the lateral leg angle
a_{Arms}	amplitude of the arm movement

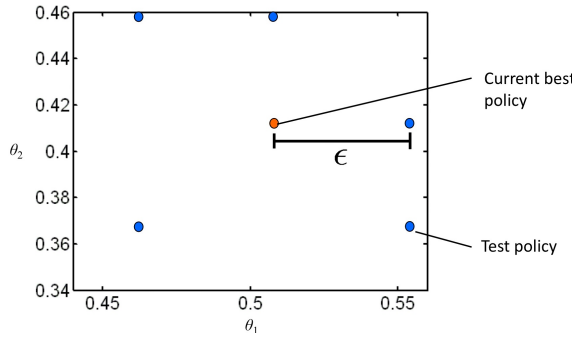
Feedback Control Parameters	
K^r	gain of lateral gyro feedback
K^p	gain of sagittal gyro feedback
Φ_{Reset}	nominal trunk phase for phase resetting
T_{Trans}	transition time for phase resetting

Reward:

$$f(d,v) = \begin{cases} \frac{v}{v_{\max}} \cdot \frac{d}{d_{\text{exp}}} & \text{if } d < d_{\text{exp}} \\ \frac{v}{v_{\max}} + 1 & \text{else.} \end{cases}$$

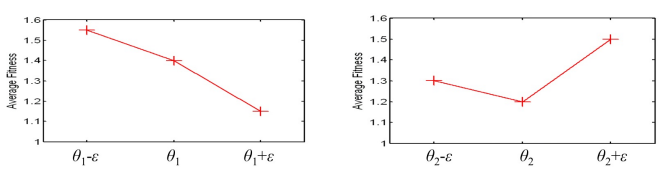
Test policies:

Add randomly $\{-\varepsilon, 0, \varepsilon\}$ to the individual parameters θ_i of current best policy



Parameter Update

Average rewards per parameter change

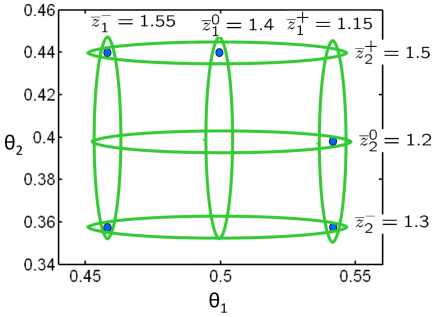


Construct adjustment vector $a = (a_1, \ldots, a_n)$, where

$$a_j = \begin{cases} 0 & \text{if } \bar{z}^0_j > \bar{z}^+_j \text{ and } \bar{z}^0_j > \bar{z}^-_j, \\ \bar{z}^+_j - \bar{z}^-_j & \text{otherwise.} \end{cases}$$

Normalize adjustment to length η and add to θ:

$$\theta^{i+1} \leftarrow \theta^i + \eta \frac{a}{\|a\|}$$



Controlling Helicopters

Model:

12 dimensional state space
- Helicopter pose (position + orientation) + velocities

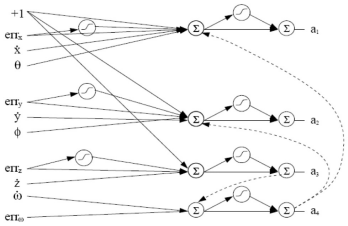
4 dimensional actions
- 2 rotor-plane pitch
- rotor blade tilt
- tail rotor tilt

Actions are selected every 20 ms

Hovering:

Desired hovering position: (x^*,y^*,z^*,ω^*)

Simple policy, computed by neural network



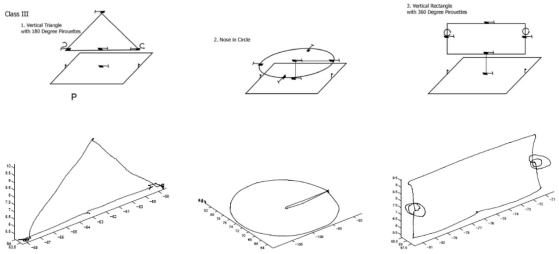
- Edges obtained using prior knowledge
- Learns control gains (9 parameters)

Quadratic reward function:
- Costs for deviation from desired position and orientation
- Quadratic costs for actions

Maneuvers:

Fly three maneuvers from the most difficult RC helicopter competition class

Trajectory following:
- Costs for distance to desired trajectory
- Reward for making progress along the trajectory

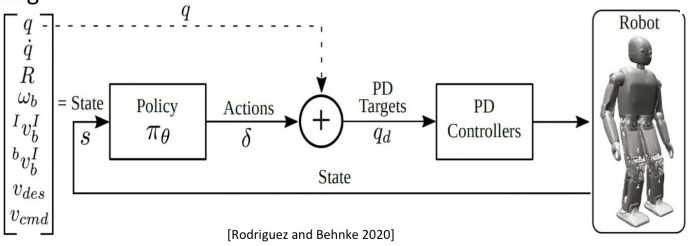


Omni-directional Walking:

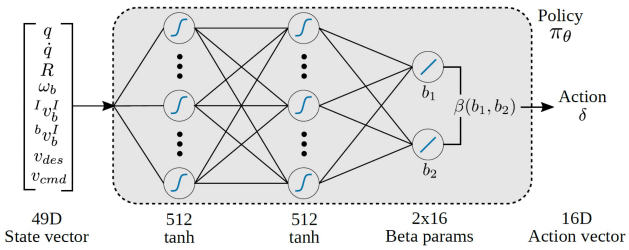
State includes joint positions and velocities, robot orientation, robot speed

Actions are increments of joint positions

Simple reward structure
- Velocity tracking
- Pose regularization
- Not falling

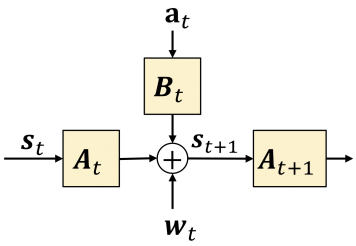


[Rodriguez and Behnke 2020]



V Linear Quadratic Regulation

Linear dynamic system



States: $\mathbf{s} \in R^n$

Actions: $\mathbf{a} \in R^d$

State Transitions: $P_{s,a}^{(t)}: \mathbf{s}_{t+1} = \mathbf{A}_t\mathbf{s}_t + \mathbf{B}_t\mathbf{a}_t + \mathbf{w}_t$

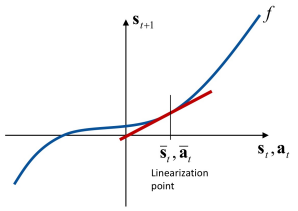
$\mathbf{A}_t$ - n×n state dynamics matrix

$\boldsymbol{B}_t$ - n×d action matrix

Zero mean Gaussian noise: $\mathbf{w}_t = \mathbf{N}(\mathbf{0}, \boldsymbol{\Sigma}_w)$

Linearization:

$\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$



$$\mathbf{s}_{t+1} \approx f(\bar{\mathbf{s}}_t, \bar{\mathbf{a}}_t) + \nabla_{\mathbf{s}} f(\bar{\mathbf{s}}_t, \bar{\mathbf{a}}_t)^T \cdot (\mathbf{s}_t - \bar{\mathbf{s}}_t) + \nabla_{\mathbf{a}} f(\bar{\mathbf{s}}_t, \bar{\mathbf{a}}_t)^T \cdot (\mathbf{a}_t - \bar{\mathbf{a}}_t)$$

$$\boldsymbol{s}_{t+1} = \boldsymbol{A}_t\boldsymbol{s}_t + \boldsymbol{B}_t\boldsymbol{a}_t + \cancel{\boldsymbol{c}_t}$$

arguments of linear function

Add one state dimension set to one to account for „state bias"

Reinforcement Learning:

Quadratic rewards: $r^{(t)}(\mathbf{s}_t, \mathbf{a}_t) = -(\mathbf{s}_t^T\mathbf{Q}_t\mathbf{s}_t + \mathbf{a}_t^T\mathbf{R}_t\mathbf{a}_t)$

Q: cost matrix

→ is positive semi-definite

Objective: Maximize rewards $r(\mathbf{s}_0, \mathbf{a}_0) + r(\mathbf{s}_1, \mathbf{a}_1) + \ldots + r(\mathbf{s}_T, \mathbf{a}_T)$

One step back-up: $V_t^*(\mathbf{s}_t) = \mathbf{s}_t^T\boldsymbol{\Phi}_t\mathbf{s}_t + \Psi_t$

with: $\boldsymbol{\Phi}_t = \mathbf{A}_t^T(\boldsymbol{\Phi}_{t+1} - \boldsymbol{\Phi}_{t+1}\mathbf{B}_t\underbrace{(\mathbf{B}_t^T\boldsymbol{\Phi}_{t+1}\mathbf{B}_t - \mathbf{R}_t)^{-1}\mathbf{B}_t^T\boldsymbol{\Phi}_{t+1})\mathbf{A}_t}_{\mathbf{L}_t} - \mathbf{Q}_t$

$$\Psi_t = -trace(\boldsymbol{\Sigma}_w\boldsymbol{\Phi}_{t+1}) + \Psi_{t+1}$$

→ discrete-time Ricatti equation

Derivation:

Let: terminal state

$$\begin{aligned} V_T^* &= \max_{a_T} r(\mathbf{s}_T, \mathbf{a}_T) \\ &= \max_{a_T} (-\mathbf{s}_T^T\mathbf{Q}_T\mathbf{s}_T - \mathbf{a}_T^T\mathbf{R}_T\mathbf{a}_T) \\ &= -\mathbf{s}_T^T\mathbf{Q}_T\mathbf{s}_T \quad because \quad \mathbf{a}_T^T\mathbf{R}_T\mathbf{a}_T \geq 0 \end{aligned}$$

Show via induction:

IH: $V_{t+1}^*(\mathbf{s}_{t+1}) = \mathbf{s}_{t+1}^T\boldsymbol{\Phi}_{t+1}\mathbf{s}_{t+1} + \Psi_{t+1}$

$\boldsymbol{\Phi}_{t+1} \in R^{n\times n}, \Psi_{t+1} \in R$

Start: $V_T^*(\mathbf{s}_T) = -\mathbf{s}_T^T\mathbf{Q}_T\mathbf{s}_T$

$\Rightarrow \quad \boldsymbol{\Phi}_T = -\mathbf{Q}_T, \ \Psi_T = 0$

$V_T^*(\mathbf{s}_T) = \mathbf{s}_T^T\boldsymbol{\Phi}_T\mathbf{s}_T + \Psi_T$

Step:

1. Use dynamic programming

$$V_t^*(\mathbf{s}_t) = \max_{\mathbf{a}_t}\left[r(\mathbf{s}_t, \mathbf{a}_t) + \sum_{\mathbf{s}_{t+1}} P_{\mathbf{s}_t,\mathbf{a}_t}(\mathbf{s}_{t+1})V_{t+1}^*(\mathbf{s}_{t+1})\right]$$

$$= \max_{\mathbf{a}_t}\left[r(\mathbf{s}_t, \mathbf{a}_t) + E_{\mathbf{s}_{t+1}\leftarrow P_{\mathbf{s}_t,\mathbf{a}_t}}[V_{t+1}^*(\mathbf{s}_{t+1})]\right]$$

$$V_t^*(\mathbf{s}_t) = \max_{\mathbf{a}_t}(-\mathbf{s}_t^T\mathbf{Q}_t\mathbf{s}_t - \mathbf{a}_t^T\mathbf{R}_t\mathbf{a}_t \ \text{// immediate costs}$$
$$+\underbrace{E_{\mathbf{s}_{t+1}\leftarrow N(\mathbf{A}_t\mathbf{s}_t+\mathbf{B}_t\mathbf{a}_t,\boldsymbol{\Sigma}_w)}}_{P_{\mathbf{s}_t,\mathbf{a}_t}}\underbrace{[\mathbf{s}_{t+1}^T\boldsymbol{\Phi}_{t+1}\mathbf{s}_{t+1} + \Psi_{t+1}]}_{V_{t+1}^*(\mathbf{s}_{t+1})})$$

simplifies to quadratic function of $\mathbf{a}_t$

2. Solve quadratic function for a

$$\mathbf{a}_t = -\underbrace{(\mathbf{B}_t^T\boldsymbol{\Phi}_{t+1}\mathbf{B}_t - \mathbf{R}_t)^{-1}\mathbf{B}_t^T\boldsymbol{\Phi}_{t+1}\mathbf{A}_t}_{\mathbf{L}_t} \cdot \mathbf{s}_t$$

Optimal policy:

$$\begin{aligned} \pi_t^*(\mathbf{s}_t) &= \arg\max_{\mathbf{a}_t} r(\mathbf{s}_t, \mathbf{a}_t) + E_{\mathbf{s}_{t+1}\leftarrow P_{\mathbf{s}_t,\mathbf{a}_t}}\left[V_{t+1}^*(\mathbf{s}_{t+1})\right] \\ &= -\mathbf{L}_t\mathbf{s}_t \end{aligned}$$

Properties

Optimal policy does not depend on noise covariance

$$\pi^*(\mathbf{s}_t) = (\mathbf{B}_t^T \mathbf{\Phi}_{t+1} \mathbf{B}_t - \mathbf{R}_t)^{-1} \mathbf{B}_t^T \mathbf{\Phi}_{t+1} \mathbf{A}_t \cdot \mathbf{s}_t$$

$$V_t^*(\mathbf{s}_t) = \mathbf{s}_t^T \mathbf{\Phi}_t \mathbf{s}_t + \Psi_t$$

where

$$\mathbf{\Phi}_t = \mathbf{A}_t^T (\mathbf{\Phi}_{t+1} - \mathbf{\Phi}_{t+1} \mathbf{B}_t (\mathbf{B}_t^T \mathbf{\Phi}_{t+1} \mathbf{B}_t - \mathbf{R}_t)^{-1} \mathbf{B}_t^T \mathbf{\Phi}_{t+1}) \mathbf{A}_t - \mathbf{Q}_t$$

$$\Psi_t = -trace(\mathbf{\Sigma}_w \mathbf{\Phi}_{t+1}) + \Psi_{t+1}$$

Not needed to determine policy

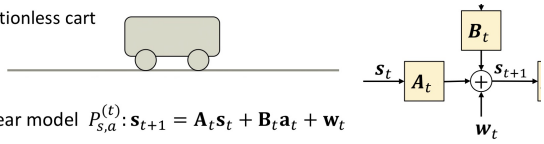
Only occurrence of noise covariance

$$P_{\mathbf{s},\mathbf{a}}^{(t)}: \mathbf{s}_{t+1} = \mathbf{A}_t \mathbf{s}_t + \mathbf{B}_t \mathbf{a}_t + \mathbf{w}_t \quad \mathbf{w}_t = \mathbf{N}(\mathbf{0}, \mathbf{\Sigma}_w)$$

Examples:

Frictionless cart:

Frictionless cart



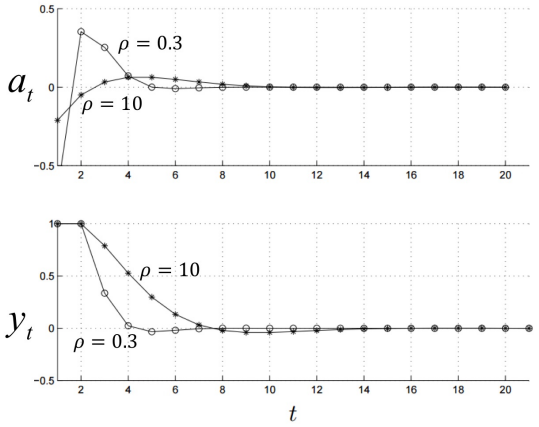
Linear model $P_{s,a}^{(t)}: \mathbf{s}_{t+1} = \mathbf{A}_t \mathbf{s}_t + \mathbf{B}_t \mathbf{a}_t + \mathbf{w}_t$

State contains position and velocity: $\boldsymbol{s_t} = (x_t, \dot{x}_t)^T$

Action is acceleration: $\boldsymbol{a_t} = (\ddot{x}_t)$

State dynamics matrix: $\mathbf{A}_t = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$, i.e. velocity $\dot{x}$ is integrated to position x

Action matrix $\boldsymbol{B}_t = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, i.e. acceleration $\ddot{x}$ is integrated to velocity $\dot{x}$



Coming back to the friction-less cart:

$$\mathbf{s}_{t+1} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \mathbf{s}_t + \begin{bmatrix} 0 \\ 1 \end{bmatrix} a_t$$

Single output: $y_t = \mathbf{C}\mathbf{s}_t = \begin{bmatrix} 1 & 0 \end{bmatrix} \mathbf{s}_t$

Initial state: $\mathbf{s}_0 = (1,0)$

Time horizon: N=20

Costs: $\mathbf{Q}_t = \mathbf{C}^T \mathbf{C}$ $\quad \mathbf{R}_t = \rho \mathbf{I}$

Differential Dynamic Programming:

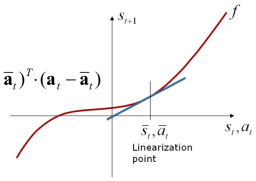
- Given simulator: $\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$ deterministic, non-linear

1. Come up with a nominal trajectory

$$\bar{\mathbf{s}}_0, \bar{\mathbf{a}}_0, \bar{\mathbf{s}}_1, \bar{\mathbf{a}}_1, \ldots, \bar{\mathbf{s}}_T, \bar{\mathbf{a}}_T$$

2. Linearize f around nominal trajectory

$$\mathbf{s}_{t+1} \approx f(\bar{\mathbf{s}}_t, \bar{\mathbf{a}}_t) + \nabla_{\mathbf{s}} f(\bar{\mathbf{s}}_t, \bar{\mathbf{a}}_t)^T \cdot (\mathbf{s}_t - \bar{\mathbf{s}}_t) + \nabla_{\mathbf{a}} f(\bar{\mathbf{s}}_t, \bar{\mathbf{a}}_t)^T \cdot (\mathbf{a}_t - \bar{\mathbf{a}}_t)$$
$$= \mathbf{A}_t \mathbf{s}_t + \mathbf{B}_t \mathbf{a}_t$$



3. Use LQR to get π_t

4. Use simulator to get new nominal trajectory

$$\bar{\mathbf{s}}_0 = initial\ state; \quad \bar{\mathbf{a}}_t = \pi_t(\bar{\mathbf{s}}_t); \quad \bar{\mathbf{s}}_{t+1} = f(\bar{\mathbf{s}}_t, \bar{\mathbf{a}}_t)$$

Goto step 2

[Abbeel et el. NIPS 2006]

II Partially Observable Markov Decision Processes

Idea: The state is not fully observable. Thus, we have to represent the state as a belief state that gathers its knowledge from incomplete/noisy measurements. Agent decisions and rewards are obtained by integrating over the belief state.

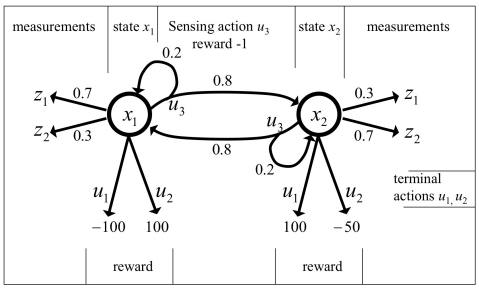
Value Function:

$$V_T(b) = \gamma \max_u \left[r(b,u) + \int V_{T-1}(b') p(b' \mid u, b)\, db' \right]$$

- $V_T(b)$: Value function horizon T
- $r(b,u)$: Immediate rewards
- $V_{T-1}(b')$: Value function horizon T-1
- $p(b' \mid u, b)$: State transition probability

Reward:

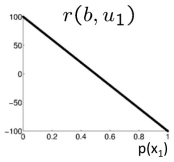
$$\begin{aligned} r(b,u) &= E_x[r(x,u)] \\ &= \int r(x,u) p(x)\, dx \end{aligned}$$

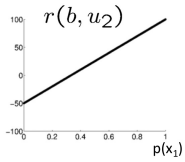


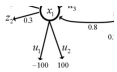
Back-up algorithm:

For each timstep and combination of action and measurement compute the value. The values/ action sequences are then represented as linear constraints.

Rewards: $r(b,u) = p_1\, r(x_1,u) + p_2\, r(x_2,u)$

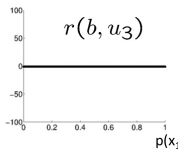


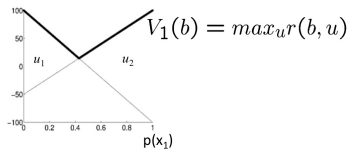




Control Choice:

$$\begin{aligned} T=1:\ V_1(b) &= \max_u r(b,u) \\ &= \max \begin{Bmatrix} -100\, p_1 & +100\,(1-p_1) & // u_1 \\ 100\, p_1 & -50\,(1-p_1) & // u_2 \\ -1 & & // u_3 \end{Bmatrix} \end{aligned}$$





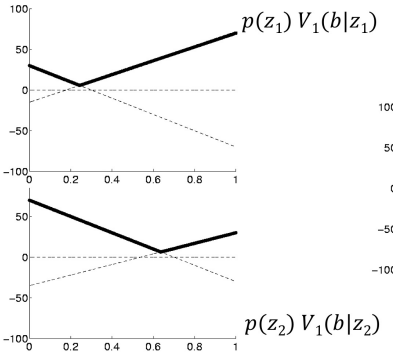
→ value is determined by immediate rewards

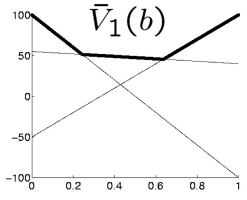
Sensing:

Belief state update:

$$\begin{aligned} p_1' &= p(x_1 \mid z) \\ &= \frac{p(z_1 \mid x_1)\, p(x_1)}{p(z_1)} \\ &= \frac{0.7\, p_1}{p(z_1)} \end{aligned} \qquad p_2' = \frac{0.3\,(1-p_1)}{p(z_1)}$$

non-linear → cancels out





Value after measurement

$$\begin{aligned} \bar{V}_1(b) = E_z[V_1(b|z)] &= \sum_{i=1}^{2} p(z_i) V_1(b|z_i) \\ &= \sum_{i=1}^{2} p(z_i) V_1\left(\frac{p(z_i|x_1)p_1}{p(z_i)}\right) \\ &= \sum_{i=1}^{2} p(z_i) \frac{1}{p(z_i)} V_1(p(z_i|x_1)p_1) \\ &= \sum_{i=1}^{2} V_1(p(z_i|x_1)p_1) \end{aligned}$$

$$\begin{aligned} \bar{V}_1(b) &= E_z[V_1(b \mid z)] \\ &= \sum_{i=1}^{2} p(z_i)\, V_1(b \mid z_i) \\ &= \max \begin{Bmatrix} -70\, p_1 & +30\,(1-p_1) \\ 70\, p_1 & -15\,(1-p_1) \end{Bmatrix} // z_1 \\ &\quad + \max \begin{Bmatrix} -30\, p_1 & +70\,(1-p_1) \\ 30\, p_1 & -35\,(1-p_1) \end{Bmatrix} // z_2 \end{aligned}$$

combined & pruned

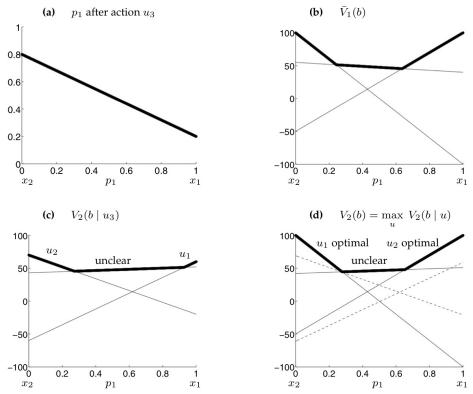
$$= \max \begin{Bmatrix} -100\, p_1 & +100\,(1-p_1) \\ +40\, p_1 & +55\,(1-p_1) \\ +100\, p_1 & -50\,(1-p_1) \end{Bmatrix} \begin{matrix} \to u_1 \\ \to u_3 \\ \to u_2 \end{matrix}$$

then best

State transition:

Belief state update:

$$
\begin{aligned}
p_1' &= E_x[p(x_1 \mid x, u_3)] \\
&= \sum_{i=1}^{2} p(x_1 \mid x_i, u_3) p_i \\
&= 0.2p_1 + 0.8(1 - p_1) \\
&= 0.8 - 0.6p_1
\end{aligned}
$$



Value after transition:

→simply input into value after sensing

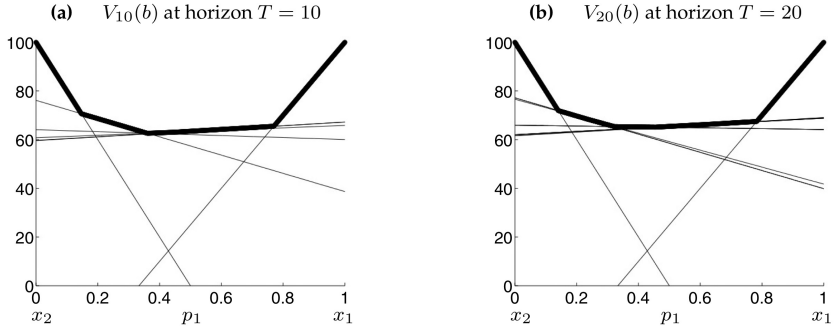
$$
\begin{aligned}
\bar{V}_1(b \mid u_3) &= \max \left\{ \begin{array}{ll} -100\ (0.8 - 0.6\ p_1) & +100\ (1 - (0.8 - 0.6\ p_1)) \\ 40\ (0.8 - 0.6\ p_1) & +55\ (1 - (0.8 - 0.6\ p_1)) \\ 100\ (0.8 - 0.6\ p_1) & -50\ (1 - (0.8 - 0.6\ p_1)) \end{array} \right\} \\
&= \max \left\{ \begin{array}{ll} -100\ (0.8 - 0.6\ p_1) & +100\ (0.2 + 0.6\ p_1) \\ 40\ (0.8 - 0.6\ p_1) & +55\ (0.2 + 0.6\ p_1) \\ 100\ (0.8 - 0.6\ p_1) & -50\ (0.2 + 0.6\ p_1) \end{array} \right\} \\
&= \max \left\{ \begin{array}{ll} 60\ p_1 & -60\ (1 - p_1) \\ 52\ p_1 & +43\ (1 - p_1) \\ -20\ p_1 & +70\ (1 - p_1) \end{array} \right\}
\end{aligned}
$$

Value of belief state:

$$
\bar{V}_2(b) = \max \left\{ \begin{array}{ll} -100\ p_1 & +100\ (1 - p_1) \\ 100\ p_1 & -50\ (1 - p_1) \\ 59\ p_1 & -61\ (1 - p_1) \\ 51\ p_1 & +42\ (1 - p_1) \\ -21\ p_1 & +69\ (1 - p_1) \end{array} \right\} \begin{array}{l} \leftarrow \text{pruned} \\ \\ \\ \\ \leftarrow \text{pruned} \end{array}
$$

Deep Horizons:

It is crucial to prune otherwise the number of constraints will explode

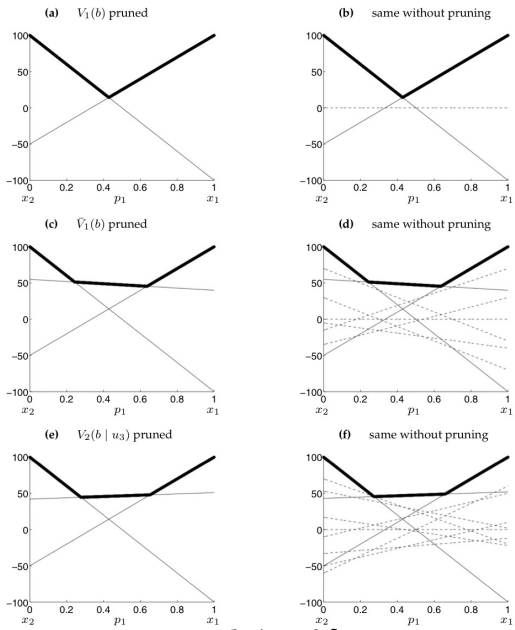


POMDP(T):

Algorithm POMDP(T):

$\Upsilon = (0; 0, \ldots, 0)$
for $\tau = 1$ *to* T *do*
  $\Upsilon' = \emptyset$
  for all $(u'; v_1^k, \ldots, v_N^k)$ *in* Υ *do*
    for all control actions u *do*
      for all measurements z *do*
        for $j = 1$ *to* N *do*
          $v_{u,z,j}^k = \sum_{i=1}^{N} v_i^k\ p(z \mid x_i)\ p(x_i \mid u, x_j)$
        endfor
      endfor
    endfor
  endfor
  for all control actions u *do*
    for all $k(1), \ldots, k(M) = (1, \ldots, 1)$ *to* $(|\Upsilon|, \ldots, |\Upsilon|)$ *do*
      for $i = 1$ *to* N *do*
        $v_i' = \gamma \left[r(x_i, u) + \sum_z v_{u,z,i}^{k(z)} \right]$
      endfor
      add $(u; v_1', \ldots, v_N')$ *to* Υ'
    endfor
  endfor
  optional: prune Υ'
  $\Upsilon = \Upsilon'$
endfor
return Υ

Pruning:



T=20: $10^{547\,864}$
T=30: $10^{561\,012\,337}$

Point-based Value Iteration

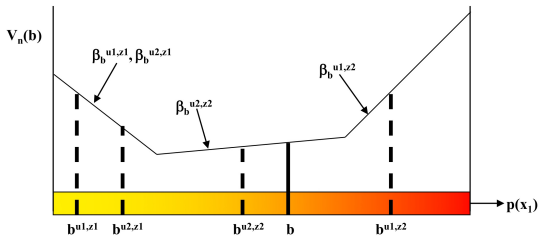
Idea: Instead of keeping complete history for all states only keep a small subset of representative belief points

Value update: Only update belief points

For each point $b \in B$

For each action and observation (u, z):

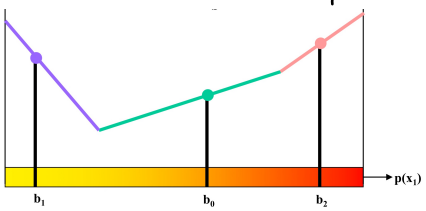
1. Project forward $b \rightarrow b^{u,z}$ and find best value: $\beta_b^{u,z}(x) = V_n(b^{u,z})$
2. Sum over observations: $\beta_b^u(x) = r(x,u) + \gamma \sum_{z,x'} p(x'|x,u)p(z|x)\beta_b^{u,z}(x')$
3. Maximize value: $V_{n+1} \leftarrow \text{argmax}_u \, \beta_b^u$



Belief point set expansion

A new point is sampled after T value updates:

1. Sample new points by simulating one step
2. Take point farest away using the L1 distance



Complexity

	Exact Update	Point-based Update
I - Projection	$S^2A\Omega\Gamma_n$	$S^2A\Omega B$
II - Sum	$SA\underbrace{\Gamma_n^{\Omega}}_{\Gamma_{n+1}}$	$SA\Omega B^2$
III - Max	$SA\Gamma_n$	SAB

S = # states

Γ_n = # solution vectors at iteration *n*

A = # actions

B = # belief points (does not grow)

Ω = # observations

→ complexity does not grow with time

Bounded Error

Let: Δ: set of reachable beliefs

B: set of belief points

Theorem: *For any belief set B and any horizon n, the error of the PBVI algorithm* $\eta_n = \|V_n^B - V_n^*\|$ *is bounded by:*

$$\eta_n \leq \frac{(R_{\max} - R_{\min})\varepsilon_B}{(1-\gamma)^2}$$

$$where \quad \varepsilon_B = \max_{b' \in \Delta} \min_{b \in B} \| b - b' \|_1$$

→ depends on how dense we sample B

QMDPS

Idea: Only consider the first step to be stochastic. After that the world becomes fully observable

Algorithm:

Algorithm QMDP($b = (p_1, \ldots, p_N)$):

$\hat{V} = \text{MDP\_discrete\_value\_iteration}()$

for all control actions u do

$$Q(x_i, u) = r(x_i, u) + \sum_{j=1}^{N} \hat{V}(x_j)\, p(x_j \mid u, x_i)$$

endfor

return $\arg\max_u \sum_{i=1}^{N} p_i\, Q(x_i, u)$ // weight action values by state belief

Augmented MDPs

Idea: Augment the state to include an uncertainty component and work with traditional method

Augmented state:

$$\bar{b} = \begin{pmatrix} \arg\max_x b(x) \\ H_b(x) \end{pmatrix}, \qquad H_b(x) = -\int b(x)\, \log b(x)\, dx$$

Algorithm:

Algorithm AMDP_value_iteration( ):

for all $\bar{b}$ *do* // learn model
  for all u *do*
    for all $\bar{b}$ *do* // initialize model
      $\hat{P}(\bar{b}, u, \bar{b}') = 0$
    endfor
    $\hat{\mathcal{R}}(\bar{b}, u) = 0$
    repeat n *times* // learn model
      generate b *with* $f(b) = \bar{b}$
      sample $x \sim b(x)$ // belief sampling
      sample $x' \sim p(x' \mid u, x)$ // motion model
      sample $z \sim p(z \mid x')$ // measurement model
      calculate $b' = B(b, u, z)$ // Bayes filter
      calculate $\bar{b}' = f(b')$ // belief state statistic
      $\hat{P}(\bar{b}, u, \bar{b}') = \hat{P}(\bar{b}, u, \bar{b}') + \frac{1}{n}$ // learn transitions prob's
      $\hat{\mathcal{R}}(\bar{b}, u) = \hat{\mathcal{R}}(\bar{b}, u) + \frac{r(u,s)}{n}$ // learn payoff model
    endrepeat
  endfor
endfor
for all $\bar{b}$ // initialize value function
  $\hat{V}(\bar{b}) = r_{\min}$
endfor
repeat until convergence // value iteration
  for all $\bar{b}$ *do*
    $\hat{V}(\bar{b}) = \gamma \max_u \left[\hat{\mathcal{R}}(u, \bar{b}) + \sum_{\bar{b}'} \hat{V}(\bar{b}')\, \hat{P}(\bar{b}, u, \bar{b}')\right]$
endfor
return $\hat{V}, \hat{P}, \hat{\mathcal{R}}$ // return value fct & model

Greedy policy:

Algorithm policy_AMDP($\hat{V}$, $\hat{P}$, $\hat{\mathcal{R}}$, b):

$\bar{b} = f(b)$

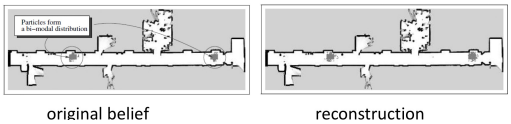
return $\arg\max_u \left[\hat{\mathcal{R}}(u, \bar{b}) + \sum_{\bar{b}'} \hat{V}(\bar{b}')\, \hat{P}(\bar{b}, u, \bar{b}')\right]$

Belief Compression

Idea: Compress belief states using PCA or other dimension reduction tools and solve low-dimensio POMDP

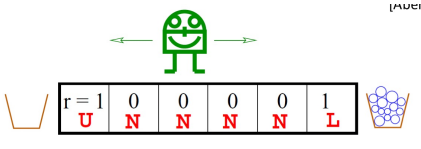
$b \approx f(U\tilde{b})$

common belief unlikely belief



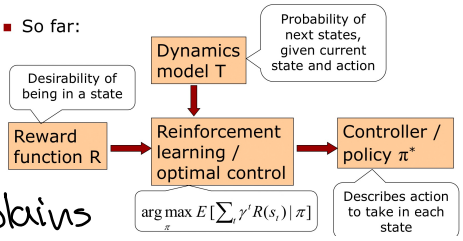
original belief reconstruction

[Aberdeen]



- Agent perceives U, N, L
- In U it can drop a load
- In L it can get a load
- In N it cannot observe if loaded or not
- Optimal policy: Move left when loaded, otherwise move right
- A policy based on current state and observations is not sufficient
- Partial observability because agent cannot detect it is loaded
- How many states? Naïve: 2×6
- But: To determine policy, we need only a single bit (loading_state)

VII Inverse Reinforcement Learning



Task: Given policy π and dynamics T, recover rewards R

Find a reward function which explains the expert behavior:

$$E[\sum_t \gamma^t R^*(s_t)|\pi^*] \geq E[\sum_t \gamma^t R^*(s_t)|\pi] \quad \forall \pi$$

→ rewards "explain" teacher policy better than any other

Problems:

1. reward function is ambiguit
2. we only observe expert traces rather than complete policies
3. assumes expert is optimal
4. assumes we can enumerate all policies

Motivation:

1. Model natural behaviors
2. Imitation learning through inverse RL
   → rewards provide the most succinct and transferable definition of the task
3. Modeling of other agents, both adversarial and cooperative

Behavioral Cloning

Given teacher demonstrations learn the teacher's policy directly.

1. Fix policy class → SVM, NN etc.
2. Fit to training data: $(s_0, a_0), (s_1, a_1), (s_2, a_2), \ldots$

→ in most planning-orientated tasks the rewards are typically more succinct than the optimal policy

Inverse RL Approaches:

1. Max margin
2. Feature expectation matching
3. Interpret reward function as a parameterization of a policy class

Feature-based Reward Function: $R(s) = w^T \Phi(s)$

w: n-dimensional vector of feature weights

Φ(s): Basis functions S -> R^n

Expected discounted reward:

$$E[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi] = E[\sum_{t=0}^{\infty} \gamma^t w^T \phi(s_t) \mid \pi]$$
$$= w^T E[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \mid \pi]$$
$$= w^T \mu(\pi)$$

Inverse RL problem: $w^{*T} \mu(\pi^*) \geq w^{*T} \mu(\pi) \quad \forall \pi$

Properties:

1. Feature expectations can be estimated from sample trajectories
   → rely only on expert feature expectations $\mu(\pi^*)$
2. Number of required expert demonstrations scales with number of features in reward function
3. Number of expert demonstrations does not depend on:
   - complexity of optimal policy
   - size of the state space

Max Margin Optimization: Find $\min_w \|w\|_2^2 + C\sum_i \xi^{(i)}$ [Ratliff et al., 2006

subject to

$$w^T \mu(\pi^{(i)*}) \geq w^T \mu(\pi^{(i)}) + m(\pi^{(i)*}, \pi^{(i)}) - \xi^{(i)} \quad \forall i, \pi^{(i)}$$

Derivation:

1. Original approach: Minimize $\|w\|_2^2$

   subject to $w^T \mu(\pi^*) \geq w^T \mu(\pi) + 1 \quad \forall \pi \neq \pi^*$

2. Resolve reward ambiguity:

   "Structured prediction" max margin:

   - Minimize $\|w\|_2^2$

     subject to $w^T \mu(\pi^*) \geq w^T \mu(\pi) + m(\pi^*, \pi) \quad \forall \pi$

   $m(\pi^*, \pi)$: Margin between two policies
   → e.g. number of states two policies disagree
   → Different policies should have larger margin

3. Consider expert suboptimality and different experts

Soft margin:

- Find $\min_w \|w\|_2^2 + C\xi$

subject to $w^T \mu(\pi^*) \geq w^T \mu(\pi) + m(\pi^*, \pi) - \xi \quad \forall \pi$

Multiple MDP:

- Find $\min_w \|w\|_2^2 + C\sum_i \xi^{(i)}$

subject to

$$w^T \mu(\pi^{(i)*}) \geq w^T \mu(\pi^{(i)}) + m(\pi^{(i)*}, \pi^{(i)}) - \xi^{(i)} \quad \forall i, \pi^{(i)}$$

Constraint Generation Algorithm

0. Initialize policy $\Pi^{(i)} = \{\}$

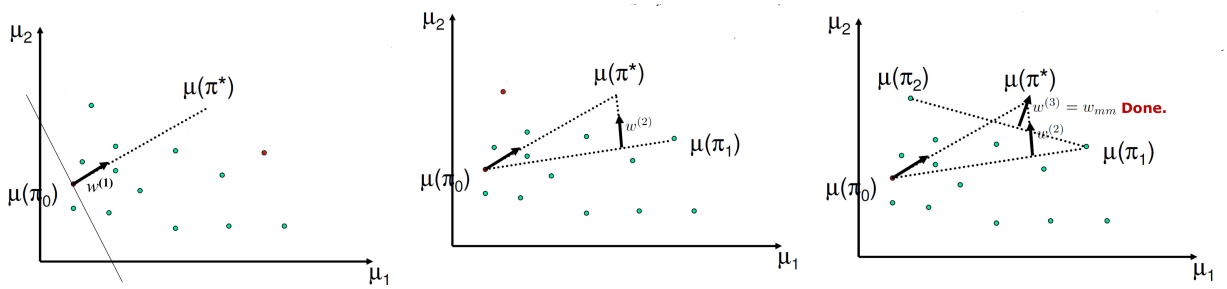
Until (no constraint is violated.):

1. Solve $\min_{w} \|w\|_2^2 + C\sum_i \xi^{(i)}$

   s.t. $w^T\mu(\pi^{(i)*}) \geq w^T\mu(\pi^{(i)}) + m(\pi^{(i)*}, \pi^{(i)}) - \xi^{(i)} \quad \forall i, \forall \pi^{(i)} \in \Pi^{(i)}$

2. Find most violated constraint: $\max_{\pi^{(i)}} w^T\mu(\pi^{(i)}) + m(\pi^{(i)*}, \pi^{(i)})$
   → find the optimal policy for the current estimate of the reward function

3. For all i: $\pi^{(i)}$ to $\Pi^{(i)}$



Feature Matching:

For a policy π to perform as well as the expert policy π^*, it suffices that the feature expectations match:

$\|\mu(\pi) - \mu(\pi^*)\|_1 \leq \varepsilon$

$\Rightarrow \forall w, \|w\| \leq 1 : \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t R_{w^*}(s_t)|\pi] = w^{*\top}\mu(\pi) \geq w^{*\top}\mu(\pi^*) - \epsilon = \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t R_{w^*}(s_t)|\pi^*] - \epsilon$

Apprenticeship Learning

First recover the reward function and then learn an optimal policy for this.

Let: $R_w(s) = w^\top\phi(s)$ for a feature map $\phi : S \rightarrow \Re^n$

Algorithm:

0: Initialize with some policy π_0

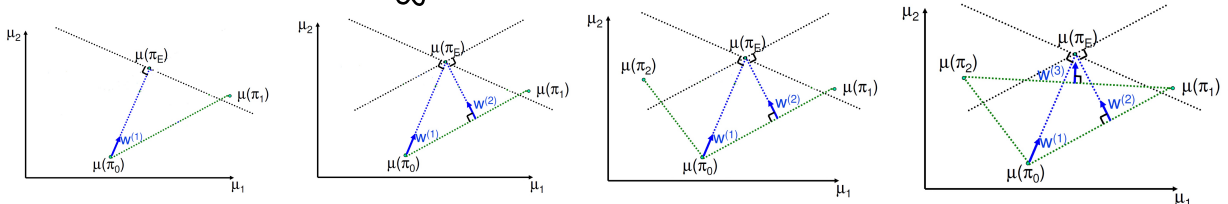
1: For i=1,... until $\gamma \leq \varepsilon/2$

2: Find a reward function such that the teacher maximally outperforms all previously found controllers:

$$\max_{\gamma, w:\|w\|_2 \leq 1} \gamma$$

$$s.t. \quad \mathrm{E}[\sum_{t=0}^{T} R_w(s_t)|\pi^*] \geq \mathrm{E}[\sum_{t=0}^{T} R_w(s_t)|\pi] + \gamma \quad \forall \pi \in \{\pi_0, \pi_1, \ldots, \pi_{i-1}\}$$

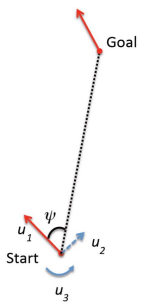
3: Find optimal policy π_i for the current reward estimate R_w



Examples

Human Locomotion:

- Model motion as dynamical system:



$$\begin{aligned}
\dot{x} &= cos\phi\, v_{tan} - \sin\phi\, v_{orth} \\
\dot{y} &= sin\phi\, v_{tan} + \cos\phi\, v_{orth} \\
\dot{\phi} &= \omega \\
\dot{v}_{tan} &= u_1 \quad \text{// forward acceleration} \\
\dot{\omega} &= u_3 \quad \text{// rotational acceleration} \\
\dot{v}_{orth} &= u_2 \quad \text{// lateral acceleration}
\end{aligned}$$

- Cost function weights control effort and angular deviation:

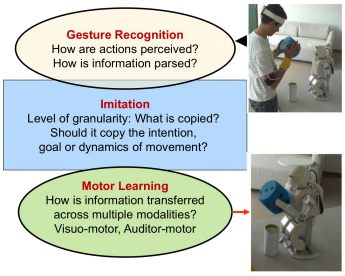
$$\int_0^T [\alpha_0 + \alpha_1 u_1(t)^2 + \alpha_2 u_2(t)^2 + \alpha_3 u_3(t)^2 + \alpha_4 \Psi(z(t), z_e)^2] dt$$

38

Imitation Learning

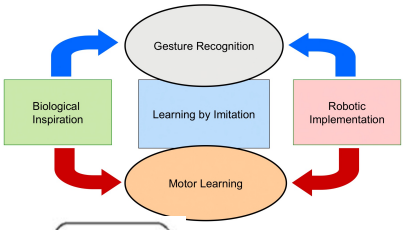
Task: Learn a task from the experience of others. The learner needs to transfer knowledge to its own domain

Important aspects:
- Whom to imitate?
- When to imitate?
- What to imitate?
- How to imitate?



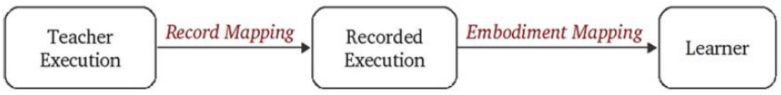
Biological Background:

Humans, especially children, learn mainly from imitation.

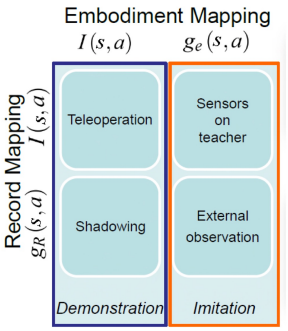


Robot learning from demonstration

Mapping:



Mapping types:



Teacher → Learner

Problems:

Correspondence problem: Imitation of same action corresponding to different object or robot

→ requires reinforcement learning

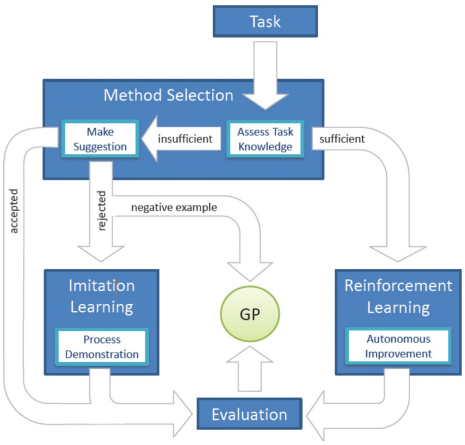
Imperfect demonstrations: requires reinforcement learning

Intent identification: Learn when and for what the action is needed

→ requires inverse reinforcement learning

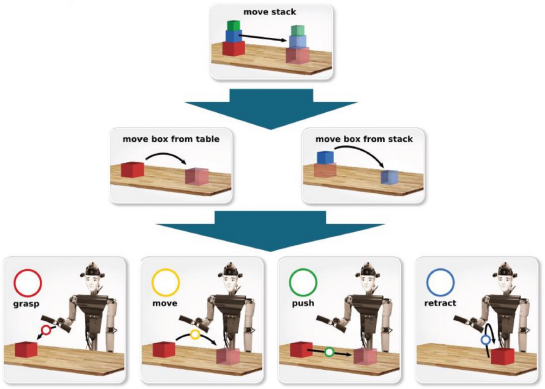
Interactive Learning

Combine imitation and reinforcement learning.
→ request demonstration only when task knowledge is insufficient



Hierarchical Task Decomposition

Model complex tasks on multiple levels of granularity



VIII Dynamic Movement Primitives

Motivation: Movement primitives

Certain motions are very repetitive with minor tweaks. For this we can use pre-computed solutions that we can minorly change for increased efficiency

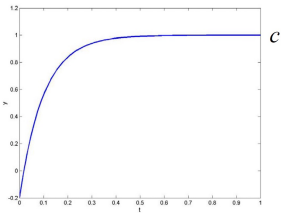
Goals:

1. Ability to adapt movement primitives to new situations
2. Sequencing of primitives to achieve complex movements
3. Ability to adapt to external signals
4. Easy programming through (one-shot) learning from demonstration

Differential Equations

Equation that describes how a dynamic system will (naturally) evolve over time

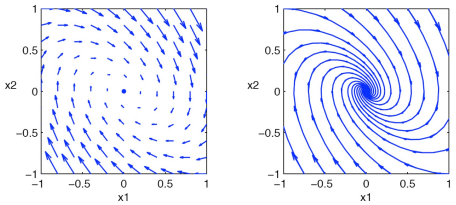
$$\dot{y} = \alpha(c - y)$$



Phase Portraits:

Express 2D dynamics as a vector field

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -x_1 - x_2 \end{bmatrix}$$

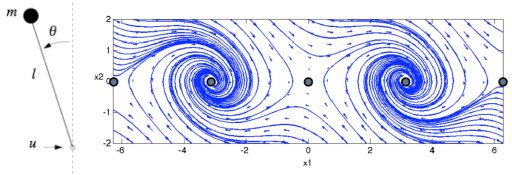


Equilibrium Points

Stationary conditions in the dynamic system such that $\dot{x} = 0$

Example:

$$\frac{dx}{dt} = \begin{bmatrix} x_2 \\ \sin x_1 - \gamma x_2 \end{bmatrix} \Rightarrow x_e = \begin{bmatrix} \pm n\pi \\ 0 \end{bmatrix}$$



Stable:

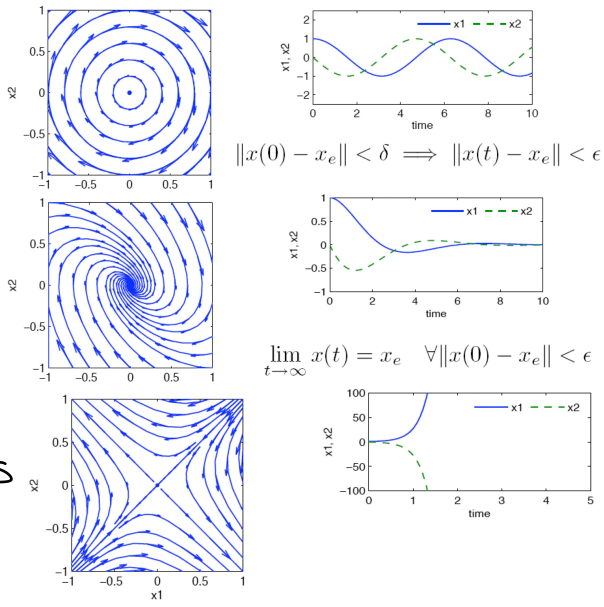
Initial conditions near the equilibrium point stay near

Asymptotically Stable:

All nearby initial conditions converge to the equilibrium point

Unstable:

Some nearby initial conditions diverge



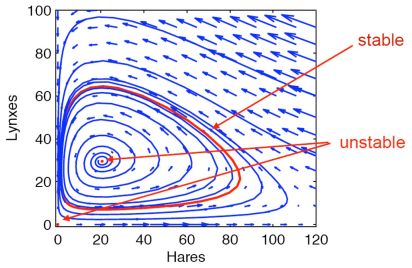
$$\|x(0) - x_e\| < \delta \implies \|x(t) - x_e\| < \epsilon$$

$$\lim_{t \to \infty} x(t) = x_e \quad \forall \|x(0) - x_e\| < \epsilon$$

Limit Cycles:

Cycle where nearby initial conditions converge to the cycle. The cycle itself is stable

$$\frac{dH}{dt} = rH\left(1 - \frac{H}{k}\right) - \frac{aHL}{c+H} \qquad H \geq 0$$
$$\frac{dL}{dt} = b\frac{aHL}{c+H} - dL \qquad L \geq 0.$$



Stability Analysis

Linear systems: Simply check the eigenvalues
Asymptotically stable, if: $\operatorname{Re}\lambda_i < 0 \quad \forall \lambda_i \in \lambda(A)$

Non-linear systems: Linearize system at a fixed point and analyze the linear system

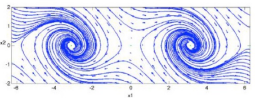
Lyapunov stability:

1. Define non-negative energy function representing the distance from the equilibrium point
2. Show that it is decreasing

Example: Inverted Pendulum

System dynamics

$$\frac{dx}{dt} = \begin{bmatrix} x_2 \\ \sin x_1 - \gamma x_2 \end{bmatrix}, \quad x = (\theta, \dot{\theta})$$



Upward equilibrium:

- $\theta = x_1 \ll 1 \implies \sin x_1 \approx x_1$

$$\frac{dx}{dt} = \begin{bmatrix} x_2 \\ x_1 - \gamma x_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -\gamma \end{bmatrix} x$$

- Eigenvalues: $-\frac{1}{2}\gamma \pm \frac{1}{2}\sqrt{4+\gamma^2}$

Positive Eigenvalue exists => **unstable**

Downward equilibrium:

- Linearize around $x_1 = \pi + z_1$: $\sin(\pi + z_1) = -\sin z_1 \approx -z_1$

$$\begin{aligned} z_1 &= x_1 - \pi \\ z_2 &= x_2 \end{aligned} \implies \frac{dz}{dt} = \begin{bmatrix} z_2 \\ -z_1 - \gamma\, z_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & -\gamma \end{bmatrix} z$$

- Eigenvalues: $-\frac{1}{2}\gamma \pm \frac{1}{2}\sqrt{-4+\gamma^2}$

All Eigenvalues have negative real part => **stable**

Solving Differential Equations

First-order linear systems: $\dot{y} = \alpha(c - y)$

Wanted: $y(t)$

Analytical solution: $y(t) = (y_0 - c)\exp(-\alpha t) + c$

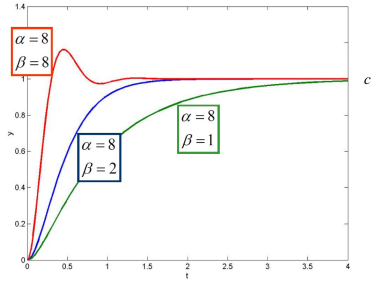
Numerical integration: Euler method, Runge-Kutta

Second-order linear systems:

$$\dot{y} = \alpha(\beta(c - x) - y)$$
$$\dot{x} = y$$

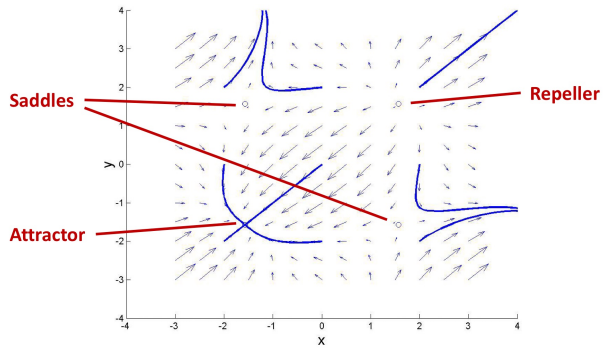
Fixed points: $y = 0, x = c$

Behavior depends on parameters:



Second-order Non-linear System:

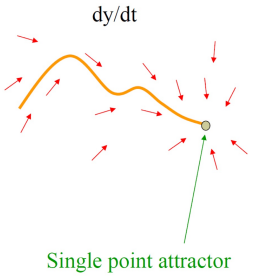
$$\dot{y} = -2\cos x - \cos y$$
$$\dot{x} = -2\cos y - \cos x$$



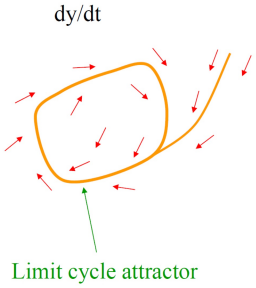
Dynamic Movement Primitives

Approaches:

Discrete:

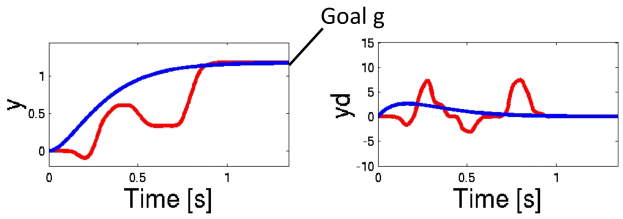


Rythmic:



Single point attractor:

$$\dot{z} = \alpha_z(\beta_z(g - y) - z)$$
$$\dot{y} = z + \frac{\sum_{i=1}^{N} \Psi_i w_i}{\sum_{i=1}^{N} \Psi_i} v$$



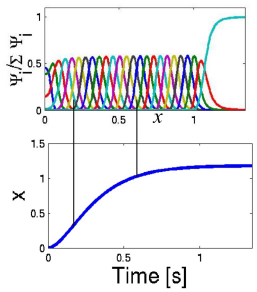
g: goal attractor point (from demonstration)

α_z, β_z: variables shaping the dynamic system ($\alpha_z = 4\beta_z$)

→ can also be learned

Ψ_i: RBF function: $\Psi_i = \exp\left(-\frac{1}{2\sigma_i^2}(\tilde{x} - c_i)^2\right)$

→ allows following (non-linear) smooth trajectories



Phase system:

$$\dot{v} = \alpha_v(\beta_v(g - x) - v)$$
$$\dot{x} = v$$

→ allows synchronization of multiple systems

Trajectory plan dynamics:

$$\dot{z} = \alpha_z\left(\beta_z\left(g - y\right) - z\right)$$
$$\dot{y} = \alpha_y\left(f\left(x, v\right) + z\right)$$

$$f(x,v) = \frac{\sum_{i=1}^{k} w_i b_i v}{\sum_{i=1}^{k} w_i}$$

Canonical system:

$$\dot{v} = \alpha_v\left(\beta_v\left(g - x\right) - v\right)$$
$$\dot{x} = \alpha_x v$$

Learning the Attractor

Given:
1. Demonstrated trajectory $y(t)_{demo}$
2. Goal g

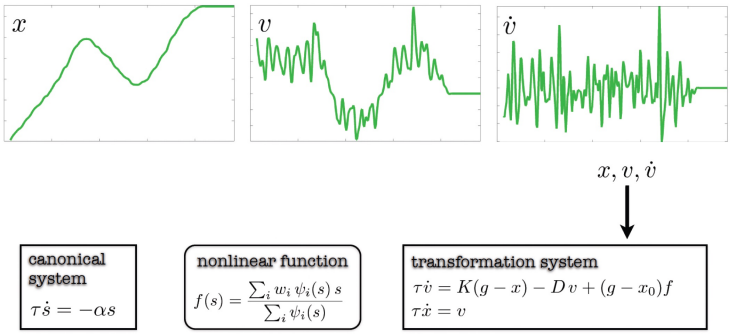
Find: Parameters of the dynamical system

Steps:
1. Extract movement duration
2. Adjust time constant of canonical dynamics to movement duration
3. Supervised learning for function approximation:

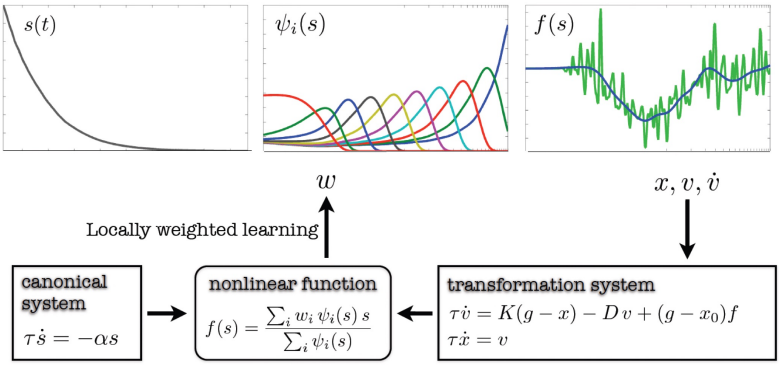
$$\dot{y} = \alpha_y \left(f(x,v) + z \right) \qquad \dot{y}_{\text{target}} = \frac{\dot{y}_{demo}}{\alpha_y} - z = f(x,v)$$

Example:

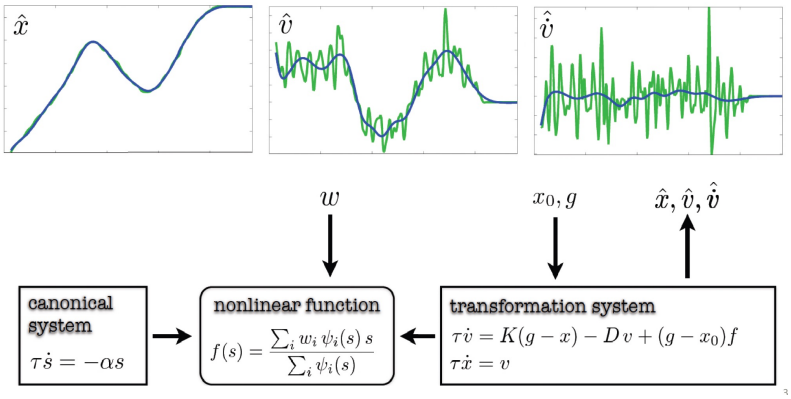
Demonstration:



Function Approximation:



Reproduction:



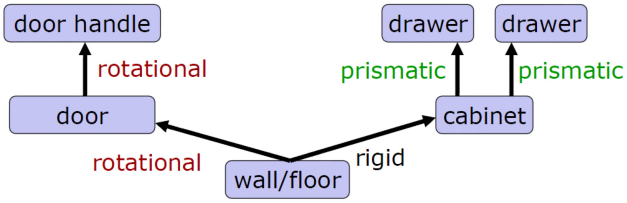
IX Learning Kinematic Models

Motivation:

1. Robots need to deal with articulated objects
   → doors, cabinets etc.
2. Improve interaction skills over time
3. Generalize to interactions with unseen objects

Method:

1. Learn models describing the relationship between two object parts



2. Infer kinematic topology of the scene
   → how are object parts connected

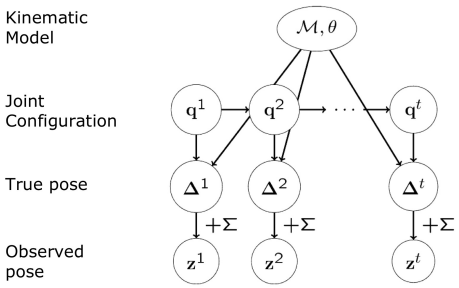
Problem definition:

Given: sequence of observations of an articulated link: $\mathcal{D}_\mathbf{z} = (\mathbf{z}^1, \mathbf{z}^2, \ldots, \mathbf{z}^t)$

Find: Estimate the most likely model and parameters for the model: $\hat{\mathcal{M}}, \hat{\theta} = \arg\max_{\mathcal{M},\theta} p(\mathcal{M}, \theta \mid \mathcal{D}_\mathbf{z})$

Structured Process Model:

Kinematic model should explain the true pose



Bayesian Model Inference:

Solve $\hat{\mathcal{M}}, \hat{\theta} = \arg\max_{\mathcal{M},\theta} p(\mathcal{M}, \theta \mid \mathcal{D}_\mathbf{z})$

1. **Model fitting:** $\hat{\theta} = \arg\max_{\theta} p(\theta \mid \mathcal{D}_\mathbf{z}, \mathcal{M})$
   → find optimal parameters for each model

   Maximum-likelihood estimator: $\hat{\theta} = \arg\max_{\theta} p(\mathcal{D}_\mathbf{z} \mid \mathcal{M}, \theta)$

   Data likelihood: $\mathbf{z} \sim \begin{cases} \Delta + \mathcal{N}(0, \Sigma_\mathbf{z}) & \text{if inlier} \\ \mathcal{U}(W) & \text{if outlier} \end{cases}$
   → mixture of gaussian around true pose and uniform for outliers

2. **Model comparison:** $\hat{\mathcal{M}} = \arg\max_{\mathcal{M}} \int p(\mathcal{M}, \theta \mid \mathcal{D}_\mathbf{z}) d\theta$
   → find the best model for given observations

   Selection: Minimize Bayesian Information Criterion

$$\hat{\mathcal{M}} = \arg\min_{\mathcal{M}} \mathrm{BIC}(\mathcal{M}) \quad \text{with:} \quad \mathrm{BIC}(\hat{\mathcal{M}}) = \underbrace{-2 \log p(\mathcal{D}_\mathbf{z} \mid \mathcal{M}, \theta)}_{\text{Negative data likelihood}} + \underbrace{k \log n}_{\text{Model complexity}}$$

Kinematic Models:

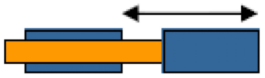
Each model provides a forward and inverse kinematic function.

Motion compositions:

$$\mathbf{x}_1 \oplus \mathbf{x}_2 = \mathbf{x}_1\mathbf{x}_2$$
$$\mathbf{x}_1 \ominus \mathbf{x}_2 = (\mathbf{x}_1)^{-1}\mathbf{x}_2$$

Rigid Model: no movable link

Prismatic joint: Pull/push connection



Parameters: origin $\boldsymbol{a}$, axis $\boldsymbol{e}$ of movement

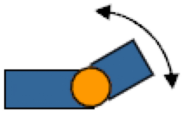
Forward kinematics:

$$f_{\mathcal{M}^{\text{prismatic}},\theta}(q) = \mathbf{a} \oplus \mathbf{e}q$$

Inverse kinematics:

$$f^{-1}_{\mathcal{M}^{\text{prismatic}},\theta}(\mathbf{z}) = \mathbf{e}^T trans(\mathbf{a} \ominus \mathbf{z})$$

Rotational joint: Rotate around the link



Parameters: rotation axis $\mathbf{c}$, rigid rotation $\mathbf{r}$

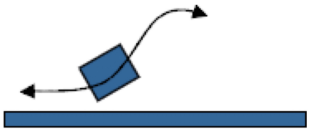
Forward kinematics:

$$f_{\mathcal{M}^{\text{rotational}},\theta}(q) = \mathbf{c} \oplus Rot_Z(q) \oplus \mathbf{r}$$

Inverse kinematics:

$$f^{-1}_{\mathcal{M}^{\text{rotational}},\theta}(\mathbf{z}) = Rot_Z^{-1}(\mathbf{c} \ominus (\mathbf{z} \ominus \mathbf{r}))$$

Non-parametric Model: No specific motion



→ objects are neither prismatic or rotational
→ still objects exhibits a structured motion

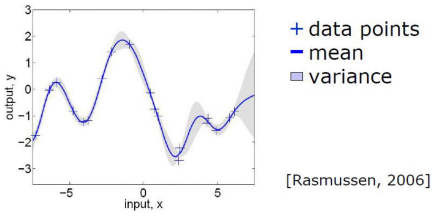
Idea: Assume data lies on a low-dimensional manifold and use a gaussian process model to model the movement

Forward kinematics:

$$f_{\mathcal{M}^{\text{GP}},\theta}(\mathbf{q}) = \mathbf{z} + \epsilon$$

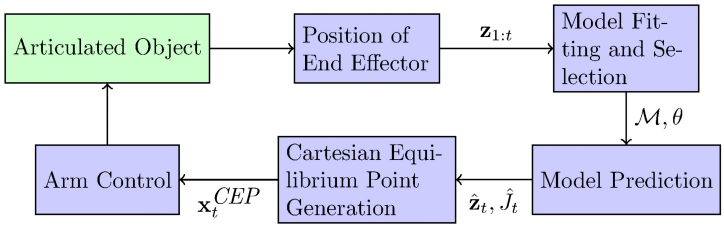
Inverse kinematics:

$$f^{-1}_{\mathcal{M}^{\text{GP}},\theta}(\mathbf{z}) = \mathbf{q} + \delta$$



Online Estimation and Control

→ learn kinematic model while manipulating an object



Exploit Prior Information

Instead of learning a model from scratch, we can use prior knowledge since most objects belong to a small subset of classes.
→ find a small subset of representative models

Model Clustering: Iteratively define new models from dataset and then decide whether to merge the models:

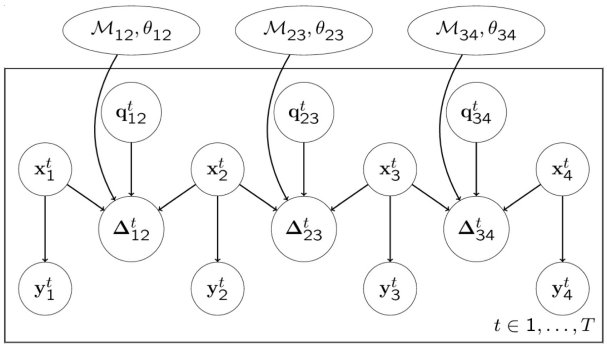
$$\text{If } \max_{j=1,\dots,m} p(\mathcal{M}_1, \dots, \mathcal{M}_{j+new}, \dots, \mathcal{M}_m \mid \mathcal{D}) > p(\mathcal{M}_{new}, \mathcal{M}_1, \dots, \mathcal{M}_m \mid \mathcal{D})$$

Then: Merge with model j
Else: Add a new model

Kinematic Chains

→ consider multiple models across multiple links
→ kinematic structure is unknown
→ consider all structures and choose best one

Process model:



- Kinematic model
- Configuration q
- True poses x
- True transformation Δ
- Observed poses y

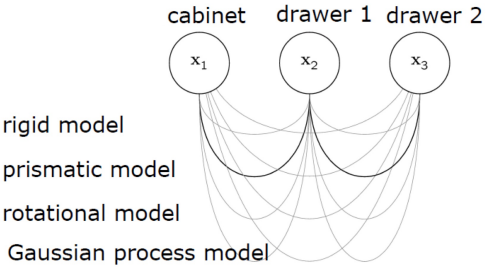
Kinematic graph:

Graph: $G = (V_G, E_G)$

Vertices: $V_G = \{1, \dots, p\}$
→ correspond to parts

Edges: $E_G \subset V_G \times V_G$
→ articulated links

Model for each edge: $\mathbb{M} = \{\mathcal{M}_{ij}, \theta_{ij} \mid (i,j) \in E_G\}$



Problem definition:

Given: t pose observations of an object consisting of p parts:

$$\mathcal{D}_y = \begin{pmatrix} y_1^1 & y_1^2 & \dots & y_1^t \\ y_2^1 & y_2^2 & \dots & y_2^t \\ \vdots & \vdots & \ddots & \vdots \\ y_p^1 & y_p^2 & \dots & y_p^t \end{pmatrix}$$

Find: most likely kinematic graph: $\hat{G} = \arg\max_G p(G \mid \mathcal{D}_y)$

Bayesian Model Inference: $\hat{G} = \arg\max_G p(G \mid \mathcal{D}_y)$

1. Link-wise model fitting (as before)
2. Link-wise model selection (as before)
3. Object-wise structure selection

MAP estimate: $\hat{E}_G = \arg\max_{E_G} \int p(E_G, \mathbb{M} \mid \mathcal{D}_y) d\mathbb{M}$

$\Leftrightarrow$ Minimize BIC. $\hat{E}_G = \arg\min_{E_G} \text{BIC}(E_G)$

Edge independence: $\hat{E}_G = \arg\min_{E_G} \sum_{(ij)\in E_G} \text{BIC}(\hat{\mathcal{M}}_{ij})$

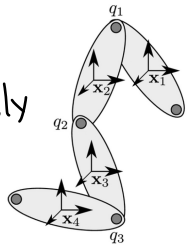
Kinematic Trees: minimum spanning tree problem

General: full evaluation over possible trees

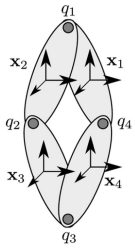
4. Object-wise DOF estimation

Effective DOF: closed chains typically have less DOF than joints

→ lower-dimensional configuration spaces increase likelihood of single configurations



3 joints
3 DOF



4 joints
1 DOF

Simultaneous localization and door state estimation

Idea: Improve localization by using doors as an additional feature

Particle filter: $p(trajectory, door\_states \mid measurements, actions, map)$

$= \underbrace{p(trajectory \mid ...)}_{\text{Particle filter}} \cdot \prod_k \underbrace{p(door\_state_k \mid trajectory, ...)}_{\text{Kalman filter}}$

Motion model:
- robot motion model
- zero-mean gaussian for doors

Observation model: Endpoint model

Door states: $p(z_t|x_{1:t}, z_{1:t-1}, m) = \int p(z_t|x_t, d_k)\ p(d_k|x_{1:t-1}, z_{1:t-1}) dd_k$

z measurements, x trajectories, m map, d door states

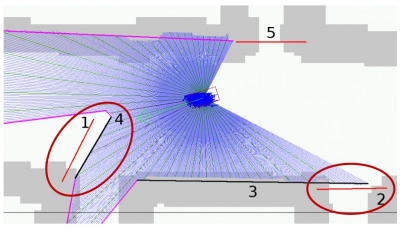
Done for each particle individually

Find line segments in scan, associate with doors

Update door opening angles with Kalman filtering

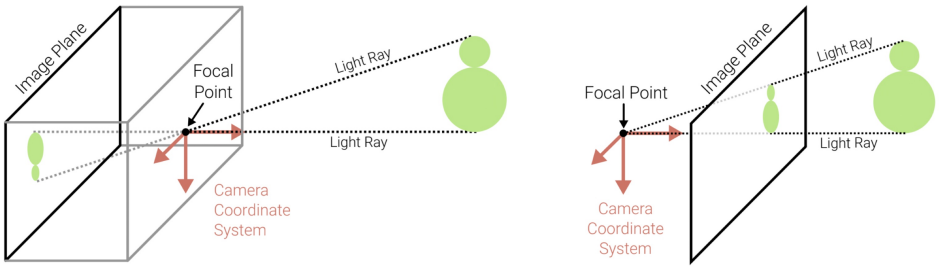
Long lines parallel to walls can correspond to closed doors

Extract polygon from scan only once (Douglas-Peucker algorithm)

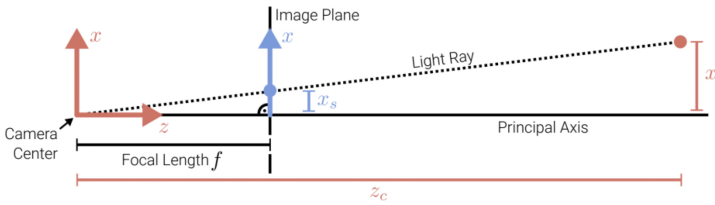


X Visual Perception

Pinhole camera:



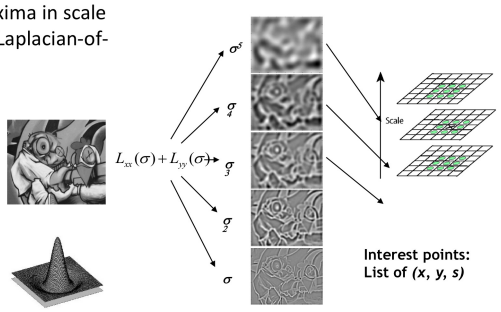
Camera parameters:



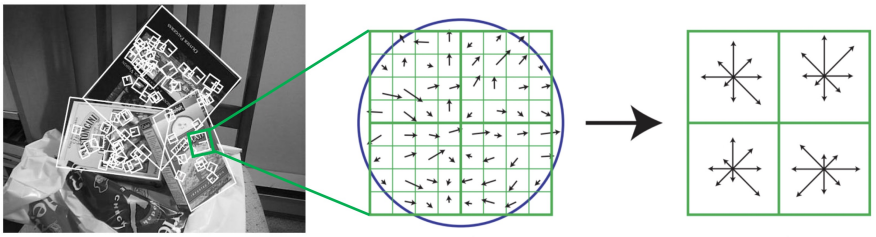
$$\frac{x_S}{f} = \frac{x_C}{z_C}$$

Laplacian-of-Gaussian:

Local maxima in scale space of Laplacian-of-Gaussian



SIFT descriptor:

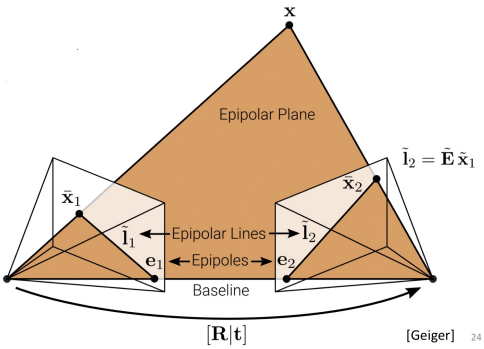


Epipolar geometry:

Objective: Recover camera pose and 3D structure from image correspondences

→ correspondence of projection lie on epipolar line in the other camera

→ eight correspondences to recover the camera pose



Epipolar plane:

spanned by the camer centers and projection

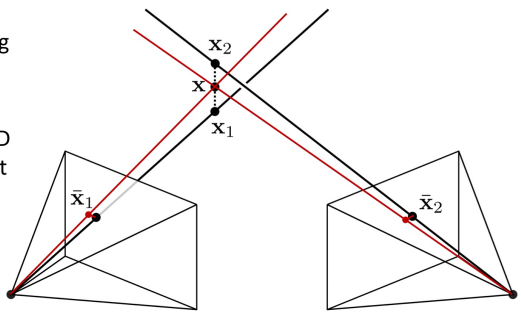
Triangulation:

Given the camera intrinsics and extrinsics, how can we recover 3D geometry?

Rays of corresponding points $\bar{x}_1, \bar{x}_2$ might not intersect

Objective: Recover 3D point $\boldsymbol{x}$ that is closest to the two rays

Approach:
Chose $\boldsymbol{x}$ to minimize reprojection error



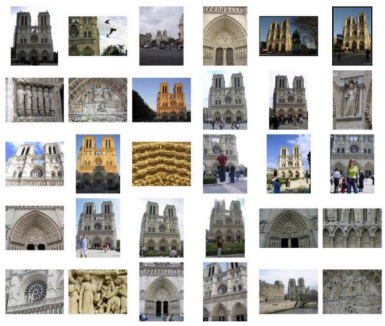
[Geiger]

Bundle adjustment:

Given: Point correspondences in different views of a scene

Objective: Estimate camera parameters and 3D point cloud

Approach: Minimize reprojection errors (distances between observed features and projected 3D points in image planes)



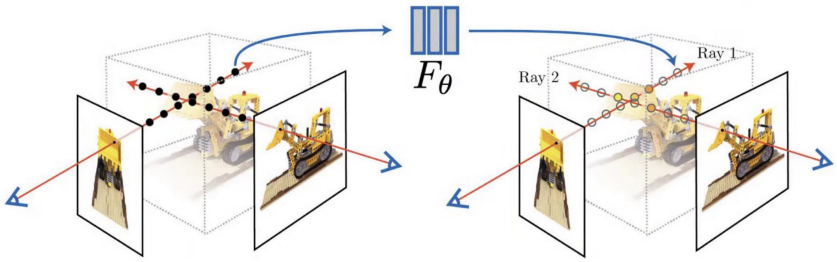
[Geiger]

NeRF: Render novel views of a scene

Volumetric rendering:

$$(\underbrace{x, y, z}_{\text{Spatial location}}, \underbrace{\theta, \phi}_{\text{Viewing direction}}) \rightarrow F_\theta \rightarrow (\underbrace{r, g, b}_{\text{Output color}}, \underbrace{\sigma}_{\text{Output density}})$$

F_θ: Fully-connected neural network 9 layers, 256 channels



Rendering of a ray:

Rendering model for ray $r(t) = o + td$:

$$C \approx \sum_{i=1}^{N} T_i \alpha_i c_i$$

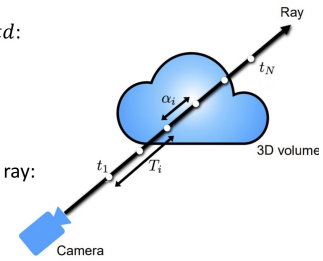
c_i: colors; T_i: weights

How much light is blocked earlier along ray:

$$T_i = \prod_{j=1}^{i-1} (1 - \alpha_j)$$

How much light is contributed by ray segment i:

$$\alpha_i = 1 - e^{-\sigma_i \delta t_i}$$ ← Density * Distance Between Points



Training: Minimize reconstruction loss

$$\min_\theta \sum_i || \text{render}_i(F_\theta) - I_i ||^2$$

→ reproduce all input views