Reinforcement Learning

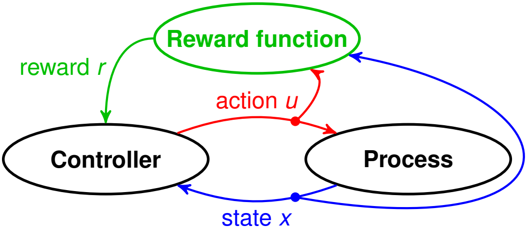
Part II: RL Using Function Approximation

Jens Kober Ivo Grondman Robert Babuška

Knowledge-Based Control Systems

Outline

Principle of RL



- Interact with a system through states and actions
- Receive rewards as performance feedback

This lecture: **approximate RL** – continuous states & actions

Recall: Solution of the RL Problem

- Q-function Q^{π} of policy π
- Optimal Q-function $Q^{*} = \max_{\pi} Q^{\pi}$
  Satisfies Bellman optimality equation:

$$Q^{*}(x,u) = \rho(x,u) + \gamma \max_{u'} Q^{*}(f(x,u),u')$$

- Optimal policy π^{*} – greedy in Q^{*}:

$$\pi^{*}(x) = \arg\max_{u} Q^{*}(x,u)$$

Types of RL Algorithms

By path to optimal solution

1. Off-policy – find Q^*, use it to compute π^*
2. On-policy – find Q^π, improve π, repeat

By level of interaction with the process

1. Online – learn by interacting with the process
2. Offline – data collected in advance (Monte-Carlo methods)

By model knowledge

1. Model-free – no f and ρ, only transition data (RL)
2. Model-based – f and ρ known (dynamic programming)
3. Model-learning – estimate f and ρ from transition data

Offline, Model-based Solution: Q-iteration (Discrete)

- Bellman optimality equation:

$$Q^*(x,u) = \rho(x,u) + \gamma \max_{u'} Q^*(f(x,u),u')$$

Turn it into an **iterative update**:

Q-iteration

repeat at each iteration ℓ
  for all x, u **do**
    $Q_{\ell+1}(x,u) \leftarrow \rho(x,u) + \gamma \max_{u'} Q_\ell(f(x,u),u')$
  end for
until convergence to Q^*

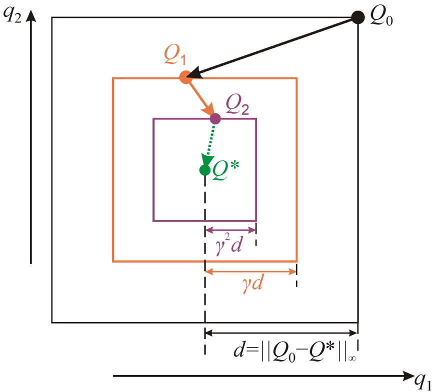
- Once Q^* available: $\pi^*(x) = \arg\max_u Q^*(x,u)$

Q-iteration Convergence

- Each update is a contraction with factor γ:

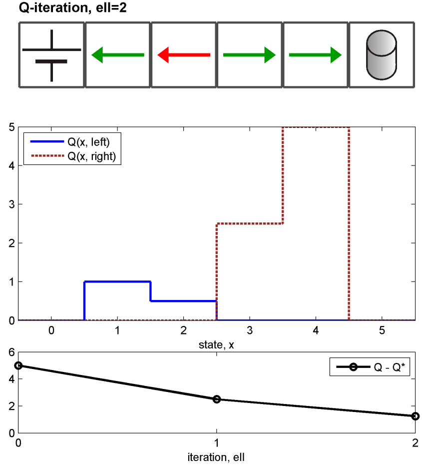
$$\|Q_{\ell+1} - Q^*\|_\infty \leq \gamma \, \|Q_\ell - Q^*\|_\infty$$

$\Rightarrow$ Q-iteration **monotonically converges** to Q^*



Cleaning Robot: Q-iteration Demo

Discount factor: $\gamma = 0.5$



Cleaning Robot: Q-iteration Progress

$$Q_{\ell+1}(x,u) \leftarrow \rho(x,u) + \gamma \max_{u'} Q_\ell(f(x,u),u')$$

	$x=0$	$x=1$	$x=2$	$x=3$	$x=4$	$x=5$
Q_0	0 ; 0	0 ; 0	0 ; 0	0 ; 0	0 ; 0	0 ; 0
Q_1	0 ; 0	1 ; 0	0 ; 0	0 ; 0	0 ; 5	0 ; 0
Q_2	0 ; 0	1 ; 0	0.5 ; 0	0 ; 2.5	0 ; 5	0 ; 0
Q_3	0 ; 0	1 ; 0.25	0.5 ; 1.25	0.25 ; 2.5	1.25 ; 5	0 ; 0
Q_4	0 ; 0	1 ; 0.625	0.5 ; 1.25	0.625 ; 2.5	1.25 ; 5	0 ; 0
Q_5	0 ; 0	1 ; 0.625	0.5 ; 1.25	0.625 ; 2.5	1.25 ; 5	0 ; 0
π^*	$*$	-1	1	1	1	$*$
V^*	0	1	1.25	2.5	5	0

Note: $Q_\ell = Q(x, \text{left})\,;\; Q(x, \text{right})$

Classical Q-function is a Table

- Separate Q-value for each x and u

0	1	.5	0.625	1.25	0
0	0.625	1.25	2.5	5	0

- In real-life control, X, U **continuous**!
  Tabular representation impossible

$\Rightarrow$ need to **approximate the Q-function**

Q-function Approximation

- In real-life control, X, U continuous

$\Rightarrow$ **approximate Q-function** $\widehat{Q}$ must be used

- Policy is greedy in $\widehat{Q}$, computed on demand for given x:

$$\pi(x) = \arg\max_u \widehat{Q}(x, u)$$

Q-function Approximation (cont'd)

- One option: use linearly parameterized approximation

$$\widehat{Q} = \sum_{i=1}^{N} \theta_i \phi_i(x, u)$$

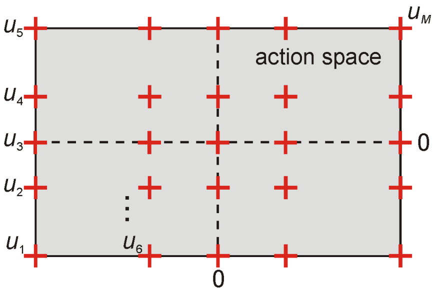
  with $\phi_i(x, u) : X \times U \mapsto \mathbb{R}$.
- $\pi(x) = \arg\max_u \widehat{Q}(x, u)$ is now a continuous optimization procedure!
- Approximator must ensure **efficient** $\arg\max$ **solution**

Approximating Over the Action Space

- Approximator must ensure efficient "arg max" solution

⇒ Typically: **action discretization**

- Choose M discrete actions $u_1, \ldots, u_M \in U$
  Solve "arg max" by explicit enumeration
- Example: grid discretization

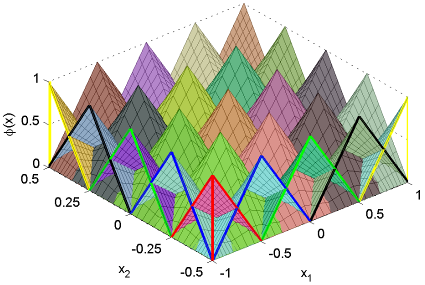


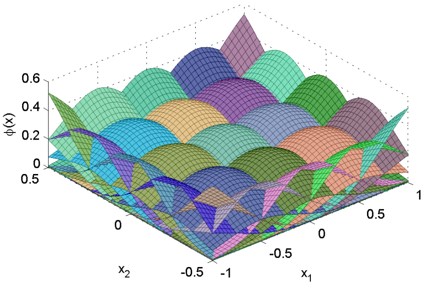
Approximating Over the State Space

- Typically: **basis functions**

$$\phi_1, \ldots, \phi_N : X \to [0, 1]$$

- Usually normalized: $\sum_i \phi_i(x) = 1$
- E.g., fuzzy approximation, RBF network approximation





TU Delft

Q-function Approximation Using Basis Functions

Given:

1. N basis functions $\phi_1, \dots, \phi_N$
2. M discrete actions $u_1, \dots, u_M$

Store:

3. $N \times M$ matrix of **parameters** θ
   (one for each pair basis function–discrete action)

Approximate Q-function

$$\widehat{Q}^{\theta}(x, u_j) = \sum_{i=1}^{N} \phi_i(x)\theta_{i,j} = [\phi_1(x) \dots \phi_N(x)] \begin{bmatrix} \theta_{1,j} \\ \vdots \\ \theta_{N,j} \end{bmatrix}$$

TU Delft

Policy from Approximate Q-function

- Recall optimal policy:

$$\pi^*(x) = \arg\max_u Q^*(x, u)$$

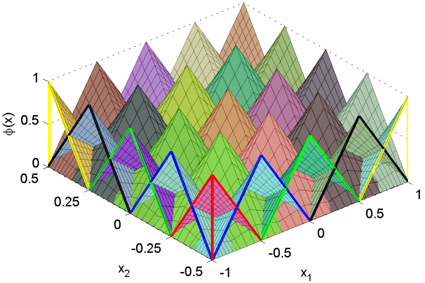
- Policy with discretized actions:

$$\widehat{\pi}^*(x) = \arg\max_{u_j,\ j=1,\dots,M} \widehat{Q}^{\theta^*}(x, u_j)$$

(θ^* = converged parameter matrix)

Fuzzy Approximator

- Basis functions: **pyramidal membership functions** (MFs) = cross-product of triangular MFs



- Each MF i has core (center) x_i
- $\theta_{i,j}$ can be seen as $\widehat{Q}(x_i, u_j)$

TU Delft

Fuzzy Q-iteration

Recall classical Q-iteration:

repeat at each iteration ℓ
  for all x, u **do**
    $Q_{\ell+1}(x, u) = \rho(x, u) + \gamma \max_{u'} Q_\ell(f(x, u), u')$
  end for
until convergence

Fuzzy Q-iteration

repeat at each iteration ℓ
  for all cores x_i, discrete actions u_j **do**
    $\theta_{\ell+1,i,j} = \rho(x_i, u_j) + \gamma \max_{j'} \widehat{Q}^{\theta_\ell}(f(x_i, u_j), u_{j'})$
  end for
until convergence

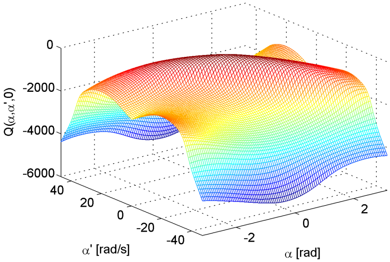
Another Example: Inverted Pendulum Swing-up

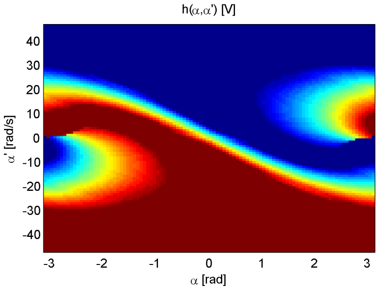
- $x = [\text{angle } \alpha, \ \text{velocity } \dot{\alpha}]^{\mathrm{T}}$
- u = voltage
- $\rho(x, u) = -x^T \begin{bmatrix} 5 & 0 \\ 0 & 0.1 \end{bmatrix} x - u^T 1 u$
- Discount factor $\gamma = 0.98$

- Goal: stabilize pointing up
- Insufficient actuation $\Rightarrow$ need to swing back & forth

Inverted Pendulum: Near-optimal Solution

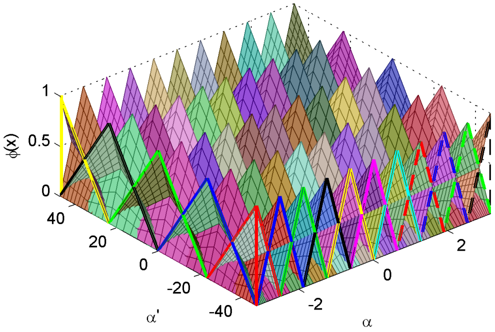
Left: Q-function for $u = 0$ Right: policy

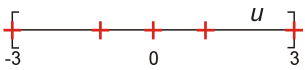




Inverted Pendulum: Fuzzy Q-iteration Demo

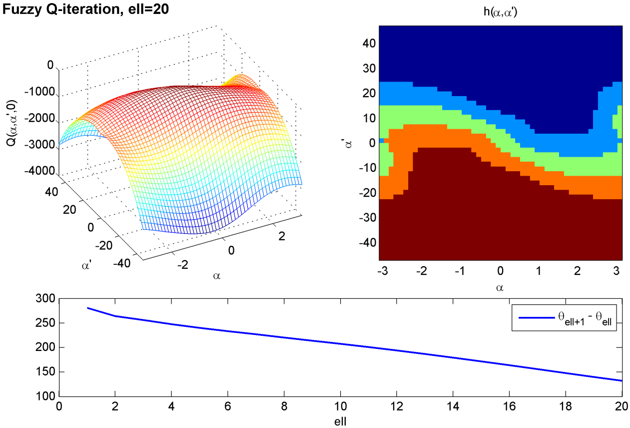
MFs: 41 $\times$ 21 equidistant grid
Discretization: 5 actions, logarithmically spaced around 0





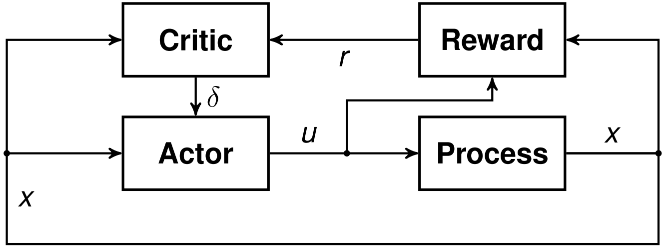
Inverted Pendulum: Fuzzy Q-iteration Demo

Demo



TUDelft

Ingredients



- Explicitly separated value function and policy
- **Actor** = control policy $\pi(x)$
- **Critic** = state value function $V(x)$

Continuous Action/State Space

To deal with continuity:

- Actor parameterized in φ: $\hat{\pi}(x, \varphi)$
- Critic parameterized in θ: $\hat{V}(x, \theta)$

Parameters φ and θ have finite size, but approximate functions on continuous (infinitely large) spaces!

Remarks on Lecture Notes

Paragraph 9.4.6 has some peculiarities:

- Slightly different notation
- Equation (9.49) should not contain u_k
- Reward not necessarily $r_k \in \{0, -1\}$, but can be any value

Terminology:

- "Performance Evaluation Unit" = reward function
- "Control Unit" = actor (i.e. the policy)
- "Stochastic Action Modifier" = exploration

Algorithm

On-policy: **find Q^π**, improve π, repeat

1. Take Bellman equation for V^π, at some x_k:

$$V^\pi(x) = \rho(x, \pi(x)) + \gamma V^\pi(f(x, \pi(x)))$$

2. Take temporal difference Δ:

$$\Delta = \rho(x, \pi(x)) + \gamma V^\pi(f(x, \pi(x))) - V^\pi(x)$$

3. Use sample $(x_k, u_k, x_{k+1}, r_{k+1})$ at each step k and parameterized V:

$$\Delta_k = r_{k+1} + \gamma \hat{V}^\pi(x_{k+1}, \theta_k) - \hat{V}^\pi(x_k, \theta_k)$$

Note: $u_k = \hat{\pi}(x_k, \varphi_k) + \tilde{u}_k$, $\hat{\pi}$ = actor, $\tilde{u}_k$ = **exploration**

TU Delft

Algorithm (cont'd)

4. Use Δ_k for critic update:

$$\theta_{k+1} = \theta_k + \alpha_c \Delta_k \left. \frac{\partial \hat{V}(x, \theta)}{\partial \theta} \right|_{\substack{x=x_k \\ \theta=\theta_k}}$$

   $\alpha_c > 0$: learning rate of critic

- $\Delta_k > 0$, i.e., $r_{k+1} + \gamma \hat{V}^\pi(x_{k+1}, \theta_k) > \hat{V}^\pi(x_k, \theta_k)$
  $\Rightarrow$ old estimate too low, increase $\hat{V}$.
- $\Delta_k < 0$, i.e., $r_{k+1} + \gamma \hat{V}^\pi(x_{k+1}, \theta_k) < \hat{V}^\pi(x_k, \theta_k)$
  $\Rightarrow$ old estimate too high, decrease $\hat{V}$.

Algorithm (cont'd)

Recall: $u_k = \hat{\pi}(x_k, \varphi_k) + \tilde{u}_k$, $\hat{\pi}$ = actor, $\tilde{u}_k$ = **exploration**

5. Use Δ_k *and* exploration term $\tilde{u}_k$ for actor update:

$$\varphi_{k+1} = \varphi_k + \alpha_a \Delta_k \tilde{u}_k \left. \frac{\partial \hat{\pi}(x, \varphi)}{\partial \varphi} \right|_{\substack{x = x_k \\ \varphi = \varphi_k}}$$

$\alpha_a \in (0, 1]$: learning rate of actor

- Product $\Delta_k \tilde{u}_k$ determines sign in update
- $\Delta_k > 0$, i.e., $r_{k+1} + \gamma \hat{V}^\pi(x_{k+1}, \theta_k) > \hat{V}^\pi(x_k, \theta_k)$
  $\Rightarrow \tilde{u}_k$ had positive effect. Move in direction of u_k.
- $\Delta_k < 0$, i.e., $r_{k+1} + \gamma \hat{V}^\pi(x_{k+1}, \theta_k) < \hat{V}^\pi(x_k, \theta_k)$
  $\Rightarrow \tilde{u}_k$ had negative effect. Move away from u_k.

Complete Actor-Critic Algorithm

Actor-critic

for every trial **do**
- initialize x_0, choose initial action $u_0 = \tilde{u}_0$
- **repeat** for each step k
  - apply u_k, measure x_{k+1}, receive r_{k+1}
  - choose **next** action $u_{k+1} = \hat{\pi}(x_{k+1}, \varphi_k) + \tilde{u}_{k+1}$
  - $\Delta_k = r_{k+1} + \hat{V}(x_{k+1}, \theta_k) - \hat{V}(x_k, \theta_k)$
  - $\theta_{k+1} = \theta_k + \alpha_c \Delta_k \left. \frac{\partial \hat{V}(x,\theta)}{\partial \theta} \right|_{\substack{x=x_k \\ \theta=\theta_k}}$
  - $\varphi_{k+1} = \varphi_k + \alpha_a \Delta_k \tilde{u}_k \left. \frac{\partial \hat{\pi}(x,\varphi)}{\partial \varphi} \right|_{\substack{x=x_k \\ \varphi=\varphi_k}}$
- **until** terminal state

end for

Pendulum Swing-up Learning

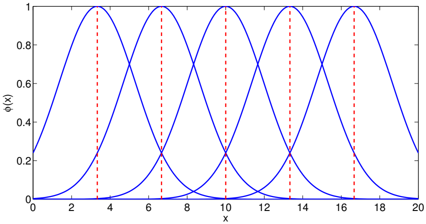
Figure : Solution to pendulum swing-up problem.

Radial Basis Functions

$$\widehat{f}(x) = \theta^{\mathrm{T}}\widetilde{\phi}(x)$$

where $\widetilde{\phi}(x)$ is a column vector with the value of normalized RBFs:

$$\widetilde{\phi}_i(x) = \frac{\phi_i(x)}{\sum_j \phi_j(x)} \quad \text{with} \quad \phi_i(x) = e^{-\frac{1}{2}(x-c_i)^{\mathrm{T}}B^{-1}(x-c_i)}$$



Evolution of a Policy

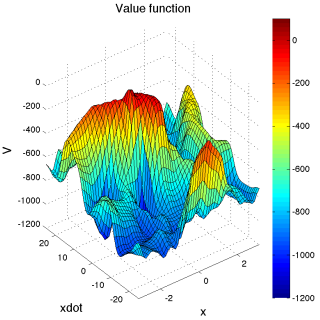


Figure : Value function and policy in learning phase.

Policy After Saturation

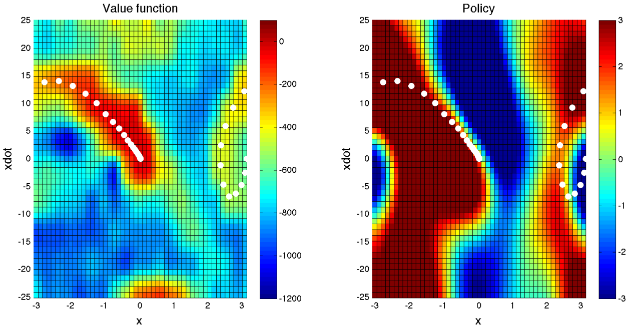
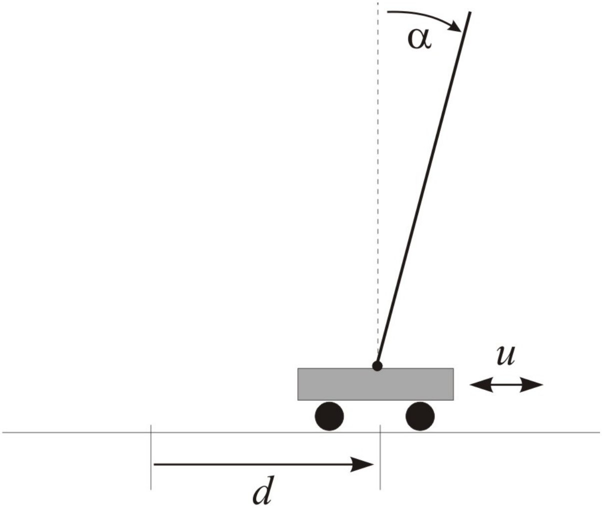
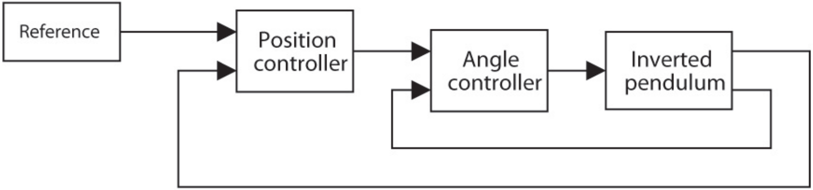


Figure : Trajectory of pendulum.

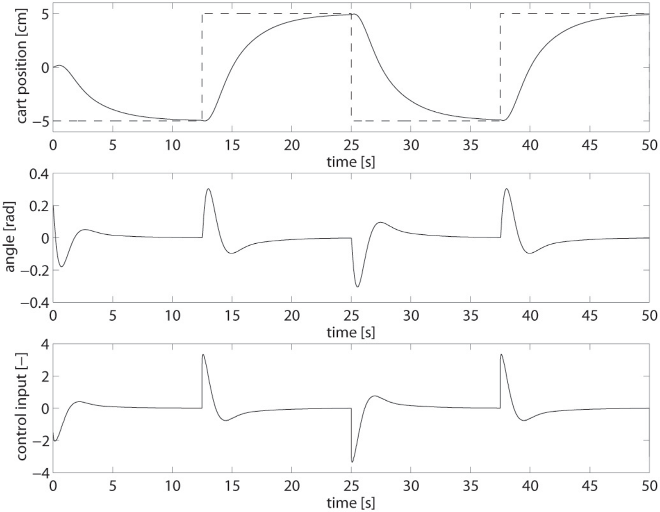
Example: Inverted Pendulum



Cascade Control Scheme

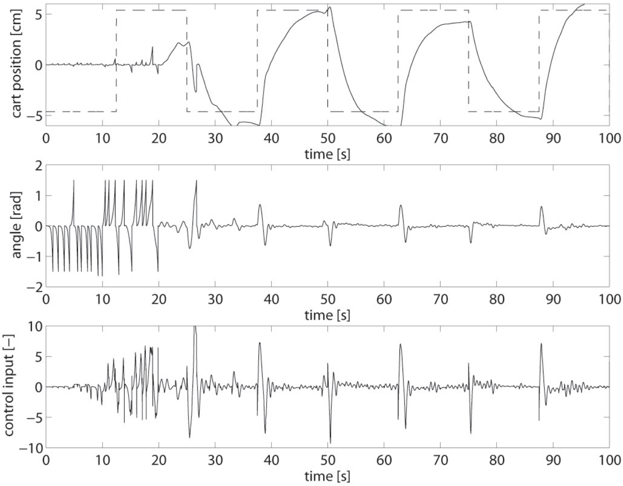


PD Control

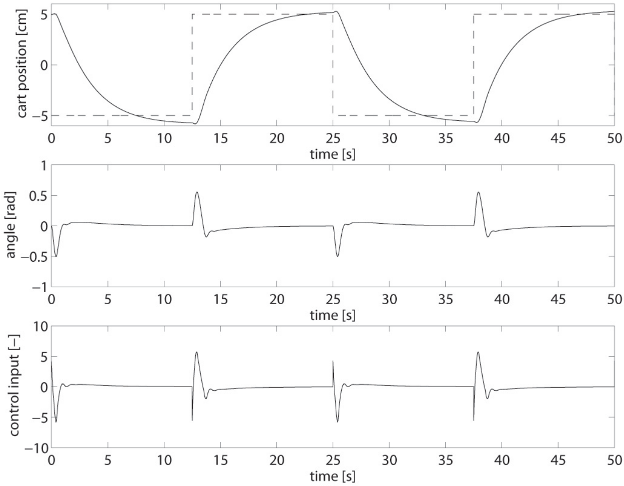


cart position [cm]
5
0
−5
0
5
10
15
20
25
30
35
40
45
50
time [s]
angle [rad]
0.4
0.2
0
−0.2
−0.4
time [s]
control input [−]
4
2
0
−2
−4
time [s]

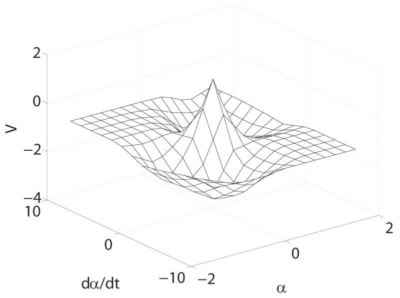
Reinforcement Learning



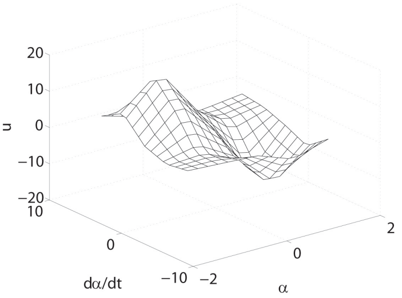
Reinforcement Learning: Final Performance



Critic and Actor Surfaces

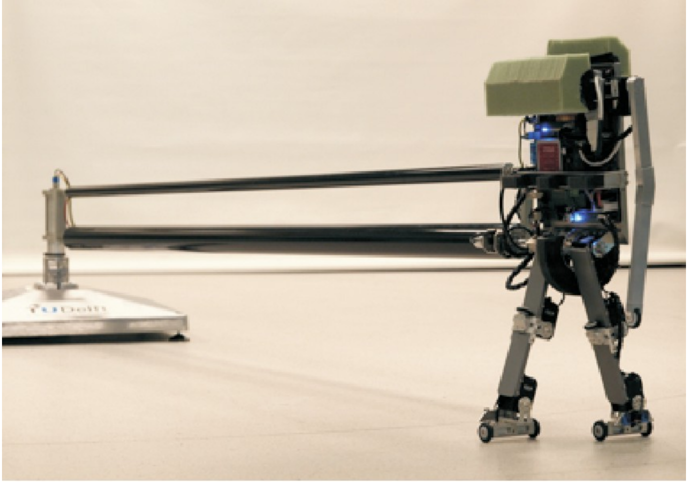


critic



actor

Example: Walking Robot Leo (Erik Schuitema)



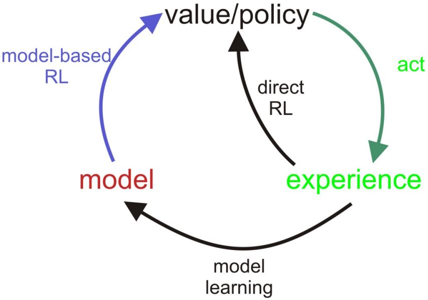
https://youtu.be/SBf5-eF-EIw

Example: Autonomous Helicopter

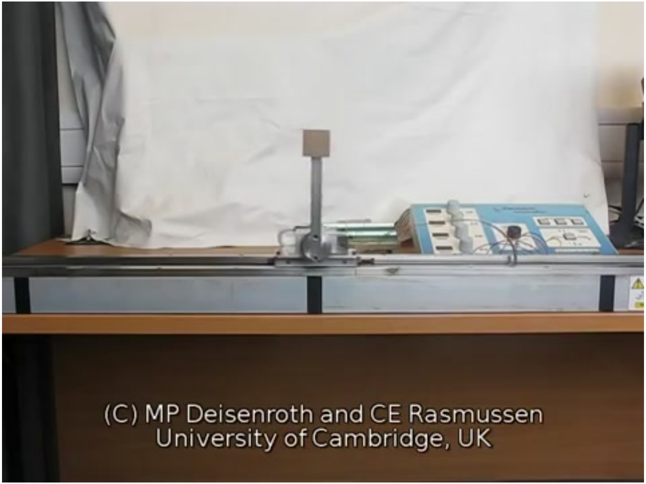
https://youtu.be/VCdxqn0fcnE

Mixed Model-Based and Model-Free: Dyna

- **Experience** is usually **costly** to obtain.
- Sometimes, **a priori information** on the environment is available (though perhaps uncertain).
- Use experience, but also **learn from the model**.



Example: Cart-Pole Swing-up (Marc P. Deisenroth)



https://youtu.be/XiigTGKZfks

Types of RL Algorithms

By path to optimal solution

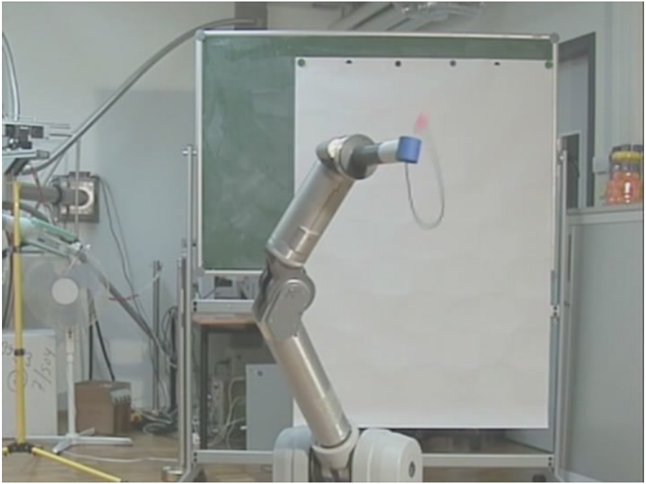
By level of interaction with the process

By model knowledge

By what is learned

1. Actor-critic – learn value function and policy
2. Critic-only – learn value function
3. Actor-only – learn policy

Example: Ball-in-a-Cup



https://youtu.be/qtqubguikMk

Summary

- **Reinforcement learning** =
  optimal, adaptive, model-free control
- Real-life RL: continuous states and actions
  – **approximation** required
- Effective algorithms for approximate RL,
  able to solve complex tasks from scratch

More Videos

- https://youtu.be/SH3bADiB7uQ
- https://youtu.be/2NLN-6fMWXI
- https://youtu.be/C63avx1YCF4
- https://youtu.be/W_gxLKSsSIE
- https://youtu.be/6ovzs1KSkJE
- https://youtu.be/8Thdf_7j4dI
- https://youtu.be/nM1HTp_P3lY
- http://www.cs.utexas.edu/~AustinVilla/?p=research/learned_walk