# Machine Learning for Systems and Control

5SC28

Lecture 5B

dr. ir. Maarten Schoukens & dr. ir. Roland Tóth

Control Systems Group

Department of Electrical Engineering

Eindhoven University of Technology

Academic Year: 2020-21 (version 1.01)

EINDHOVEN UNIVERSITY OF TECHNOLOGY

# Reinforcement Learning

Stochastic Environment

Towards Reinforcement Learning:

Value Function, Q-Function, Bellman Equation

Temporal Difference Learning

Q-Learning

# Reinforcement Learning

Stochastic Environment

Towards Reinforcement Learning:

Value Function, Q-Function, Bellman Equation

Temporal Difference Learning

Q-Learning

# Environment (deterministic)

A finite **Markov Decision Process (MDP)** is a tuple $\langle X,\ U,\ g,\ \rho \rangle$ where:

$X$ is the finite state space

$U$ is the finite action space

$g : X \times U \rightarrow X$ is the state transition function

$\rho : X \times U \rightarrow \mathbb{R}$ is the reward function

$x_{k+1} = g(x_k, u_k),\ r_{k+1} = \rho(x_k, u_k)$ with $k$ being the discrete time

A **finite MDP** has finite state, action and reward sets.

# Environment (stochastic)

A finite **Markov Decision Process (MDP)** is given by:

$p(x_{k+1}, r_{k+1} | x_k, u_k)$ the **joint state transition and reward probability**

**Transition probability:** $$p(x_{k+1} | x_k, u_k) = \sum_{r \in R} p(x_{k+1}, r | x_k, u_k)$$

**Expected reward:** $$\rho(x_k, u_k) = \sum_{r \in R} r \sum_{x \in X} p(x, r | x_k, u_k)$$

$X, U, R$ are the finite state, action and reward space respectively

and $k$ is the discrete time

# Environment (stochastic)

A finite **Markov Decision Process (MDP)** is given by:

$p(x_{k+1}, r_{k+1} | x_k, u_k)$ the **joint state transition and reward probability**

**Transition probability:** $$p(x_{k+1} | x_k, u_k) = \sum_{r \in R} p(x_{k+1}, r | x_k, u_k)$$

**Expected reward:** $$\rho(x_k, u_k, x_{k+1}) = \sum_{r \in R} r \frac{p(x_{k+1}, r | x_k, u_k)}{p(x_{k+1} | x_k, u_k)}$$

$X, U, R$ are the finite state, action and reward space respectively

and $k$ is the discrete time

The expected reward can also depend on the next state

# Recycling Robot Example (stochastic)

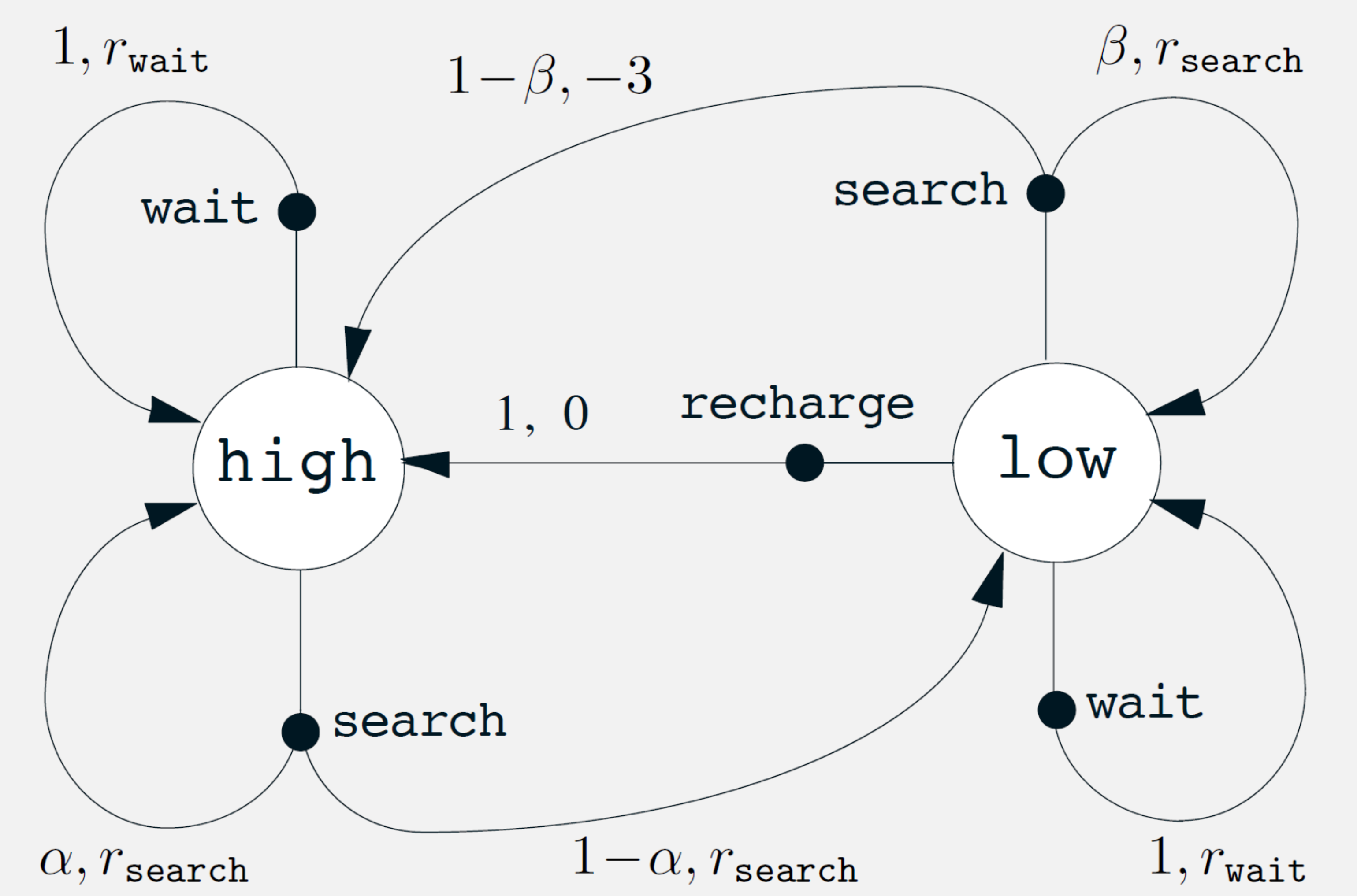
Mobile robot collects empty soda cans in an environment.

Battery: High or Low

Activity: Search, Wait, Recharge

Reward highest when actively searching, but depletes battery

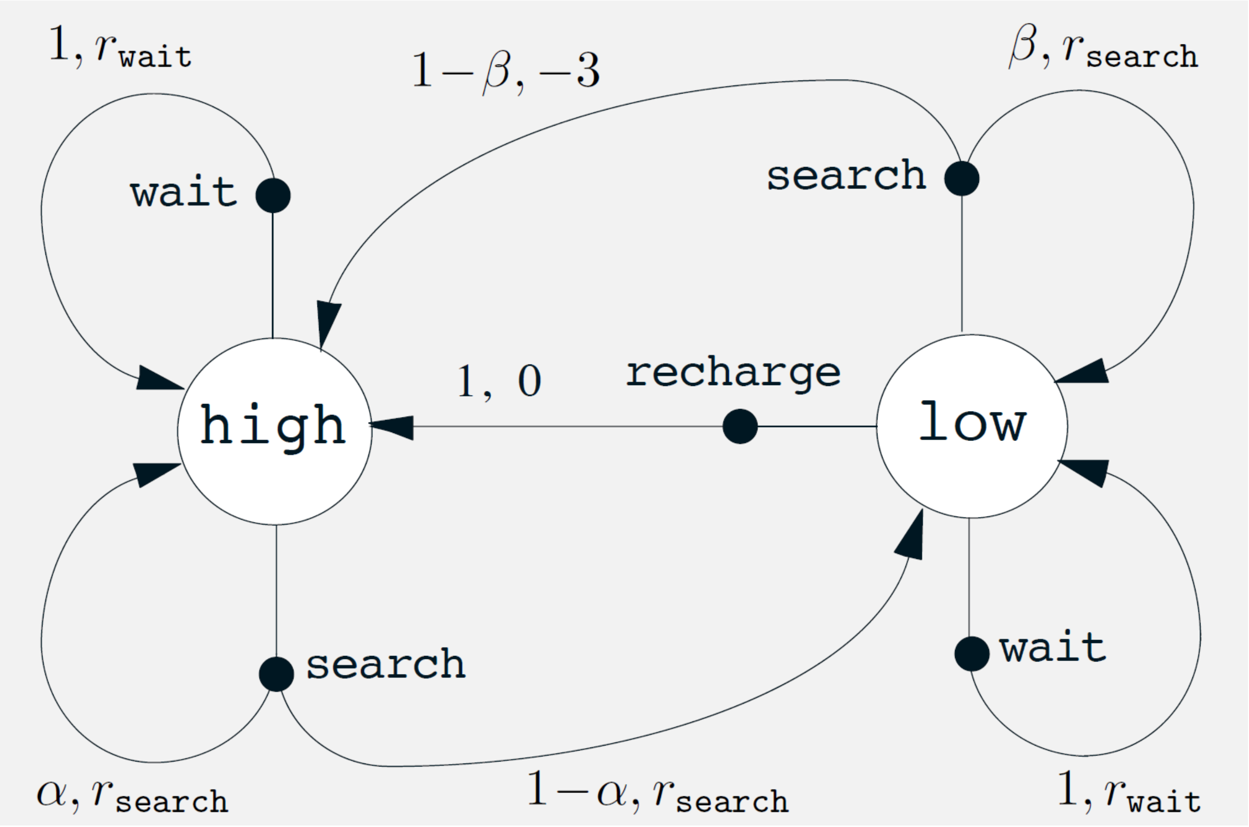
When battery low, there is a risk of running empty, resulting in a large penalty.




Source: R.S. Sutton and A.G. Barto, “Reinforcement Learning: An Introduction”, The MIT Press, 2018

# Recycling Robot Example (stochastic)

| $x_t$ | $u_t$ | $x_{t+1}$ | $p(x_{t+1}\|x_t,u_t)$ | $\rho(x_t,u_t,x_{t+1})$ |
|---|---|---|---|---|
| High | Search | High | α | $r_{search}$ |
| High | Search | Low | 1-α | $r_{search}$ |
| Low | Search | High | 1-β | -3 |
| Low | Search | Low | β | $r_{search}$ |
| High | Wait | High | 1 | $r_{wait}$ |
| High | Wait | Low | 0 | - |
| Low | Wait | High | 0 | - |
| Low | Wait | Low | 1 | $r_{wait}$ |
| Low | Recharge | High | 1 | 0 |
| Low | Recharge | Low | 0 | - |




Source: R.S. Sutton and A.G. Barto, “Reinforcement Learning: An Introduction”, The MIT Press, 2018

# Reinforcement Learning

Stochastic Environment

Towards Reinforcement Learning:

Value Function, Q-Function, Bellman Equation

Temporal Difference Learning

Q-Learning

# Value Function

Discounted Return: $$G_k = \mathbb{E}_\pi \left\{ \sum_{\tau=0}^{\infty} \gamma^\tau r_{k+\tau+1} \right\}$$

Value Function: $$V_\pi(x_k) = \mathbb{E}_\pi \left\{ G_k \middle| x_k \right\}$$

Expected discounted return given the system states at time $k$ for policy $\pi$

# Q-Function

Discounted Return: $$G_k = \mathbb{E}_\pi \left\{ \sum_{\tau=0}^{\infty} \gamma^\tau r_{k+\tau+1} \right\}$$

Value Function: $$V_\pi(x_k) = \mathbb{E}_\pi \left\{ G_k \middle| x_k \right\}$$

Q-Function: $$Q_\pi(x_k, u_k) = \mathbb{E}_\pi \left\{ G_k \middle| x_k, u_k \right\}$$

Expected discounted return given the system states and the action taken at time $k$ for policy π

# Q-Function

Q-Function:

$$Q_\pi(x_k, u_k) = \mathbb{E}_\pi \left\{ G_k \middle| x_k, u_k \right\}$$

Expected discounted return given the system states and the action taken at time $k$ for policy π

Action at time $k$ is independent of the policy, al the consecutive actions are taken following the policy $\pi$

# Bellman Equation

Develop Q-Function one step ahead:

$$
\begin{aligned}
Q_\pi(x_k, u_k) &= \mathbb{E}_\pi \left\{ G_k \middle| \, x_k, u_k \right\} \\
&= \mathbb{E}_\pi \left\{ \sum_{\tau=0}^{\infty} \gamma^\tau r_{k+\tau+1} \middle| \, x_k, u_k \right\} \\
&= \mathbb{E}_\pi \left\{ r_{k+1} + \sum_{\tau=1}^{\infty} \gamma^\tau r_{k+\tau+1} \middle| \, x_k, u_k \right\} \\
&= \mathbb{E}_\pi \left\{ r_{k+1} + \gamma V_\pi(x_{k+1}) \middle| \, x_k, u_k \right\}
\end{aligned}
$$

Link the present with the future

# Optimal Policy

Optimality of a policy is given by the Value Function:

$$\pi \geq \pi' \iff V_\pi(x) \geq V_{\pi'}(x) \; \forall \; x \in X$$

Policy $\pi$ is better than policy $\pi$' if and only if its expected return is greater than or equal to that of $\pi$' for all states.

# Optimal Policy

Optimality of a policy is given by the Value Function:

$$\pi \geq \pi' \iff V_\pi(x) \geq V_{\pi'}(x) \; \forall \; x \in X$$

Policy π is better than policy π' if and only if its expected return is greater than or equal to that of π' for all states.

Multiple optimal policies $\pi_*(x)$ can exist, they share the same optimal Value Function and optimal Q-Function:

$$V_*(x) = \max_\pi V_\pi(x)$$

$$Q_*(x,u) = \max_\pi Q_\pi(x,u)$$

$$\pi_*(x) = \arg\max_{u \in U} \; Q_*(x,u)$$

# Bellman Equation of Optimality

Value Function: $$\begin{aligned} V_*(x_k) &= \max_u Q_*(x_k, u) \\ &= \max_u \mathbb{E}_\pi \left\{ G_k \middle| \, x_k, u_k \right\} \\ &= \max_u \mathbb{E}_\pi \left\{ r_{k+1} + \gamma V_*(x_{k+1}) \middle| \, x_k, u_k \right\} \\ &= \max_u \sum_{x_{k+1}, r} p(x_{k+1}, r | x_k, u) \left( r + \gamma V_*(x_{k+1}) \right) \end{aligned}$$

# Bellman Equation of Optimality

Q-Function:

$$
\begin{aligned}
Q_*(x_k, u_k) &= \mathbb{E}_\pi \left\{ G_k \middle| \, x_k, u_k \right\} \\
&= \mathbb{E}_\pi \left\{ r_{k+1} + \sum_{\tau=1}^{\infty} \gamma^\tau r_{k+\tau+1} \middle| \, x_k, u_k \right\} \\
&= \mathbb{E}_\pi \left\{ r_{k+1} + \gamma V_*(x_{k+1}) \middle| \, x_k, u_k \right\} \\
&= \mathbb{E}_\pi \left\{ r_{k+1} + \gamma \max_u Q_*(x_{k+1}, u) \middle| \, x_k, u_k \right\} \\
&= \sum_{x_{k+1}, r} p(x_{k+1}, r | x_k, u_k) \left( r + \gamma \max_{u_{k+1}} Q_*(x_{k+1}, u_{k+1}) \right)
\end{aligned}
$$

# Reinforcement Learning

Stochastic Environment

Towards Reinforcement Learning:

Value Function, Q-Function, Bellman Equation

Temporal Difference Learning

Q-Learning

# Reinforcement Learning

Monte Carlo

Dynamic Programming

Temporal Difference

# Reinforcement Learning

Monte Carlo

Dynamic Programming

Temporal Difference

# Temporal Difference Learning

$$\text{New Estimate} \leftarrow \text{Old Estimate} + \text{Step Size} \times [\text{Target - Old Estimate}]$$

**Learn directly from raw experience**

**No model** of the system dynamics required

**Update** the knowledge of the agent on **every timestep** (action) rather than on every episode (reaching the goal or end state)
(update estimates based upon other estimates - ***bootstrapping***)

**Incremental** nature (good for online applications)

# The Prediction Problem

Estimate the Value Function $V_\pi(x)$ for a given policy $\pi$.

$$\text{New Estimate} \leftarrow \text{Old Estimate} + \text{Step Size} \times [\text{Target - Old Estimate}]$$

$$V_\pi(x_k) \leftarrow V_\pi(x_k) + \alpha \left[G_k - V_\pi(x_k)\right]$$

$$V_\pi(x_k) \leftarrow V_\pi(x_k) + \alpha \left[\underbrace{r_{k+1} + \gamma V_\pi(x_{k+1}) - V_\pi(x_k)}_{\text{Temporal Difference}}\right]$$

Instead of waiting to see all obtained rewards
required for computing $G_k$
plug in the estimate $V_\pi(x_{k+1})$
→ bootstrapping

# Temporal Difference Learning

**Input:** the policy $\pi$ to be evaluated
**Parameters:** step size $\alpha \in (0, 1]$
**Initialize** $V_\pi(x)$ arbitrarily, $V(\text{terminal}) = 0$.

**for** each episode **do**
    Initialize $x_0$
    **Repeat** for each time step of the episode
        Obtain $u_k$ based on $x_k$ using policy $\pi$

        Take action $u_k$, observe $r_{k+1}$, $x_{k+1}$

        $V_\pi(x_k) \leftarrow V_\pi(x_k) + \alpha \left[ r_{k+1} + \gamma V_\pi(x_{k+1}) - V_\pi(x_k) \right]$

        $k = k + 1$
    **Until** the states are terminal
**end for**

# Reinforcement Learning

Stochastic Environment

Towards Reinforcement Learning:

Value Function, Q-Function, Bellman Equation

Temporal Difference Learning

Q-Learning

# Temporal Difference Learning for Control

**SARSA** (State-Action-Reward-State-Action)

on-policy TD control

**Q-Learning**

off-policy TD Control

An **off-policy** learner learns the value of the optimal policy independently of the agent's actions.
An **on-policy** learner learns the value of the policy being carried out by the agent including the exploration steps.

# Temporal Difference Learning for Control

**SARSA** (State-Action-Reward-State-Action)

on-policy TD control

**Q-Learning**

off-policy TD Control

An **off-policy** learner learns the value of the optimal policy independently of the agent's actions.
An **on-policy** learner learns the value of the policy being carried out by the agent including the exploration steps.

# Q-Learning

Directly learn the optimal Q-Function $Q_*(x_t, u_t)$, use it to compute $\pi_*$

Independent of the policy being followed

Policy still is important:

Determines which state-action pairs are visited and updated

Exploration required

# Q-Learning

Take **Bellman optimality equation** at some state and action

$$Q_*(x_k, u_k) = \sum_{x_{k+1}, r} p(x_{k+1}, r | x_k, u_k) \left( r + \gamma \max_{u_{k+1}} Q_*(x_{k+1}, u_{k+1}) \right)$$

# Q-Learning

Take **Bellman optimality equation** at some state and action

$$Q_*(x_k, u_k) = \sum_{x_{k+1}, r} p(x_{k+1}, r | x_k, u_k) \left( r + \gamma \max_{u_{k+1}} Q_*(x_{k+1}, u_{k+1}) \right)$$

Turn into **iterative update**

$$Q(x_k, u_k) \leftarrow \sum_{x_{k+1}, r} p(x_{k+1}, r | x_k, u_k) \left( r + \gamma \max_{u_{k+1}} Q(x_{k+1}, u_{k+1}) \right)$$

# Q-Learning

Take **Bellman optimality equation** at some state and action

$$Q_*(x_k, u_k) = \sum_{x_{k+1}, r} p(x_{k+1}, r | x_k, u_k) \left( r + \gamma \max_{u_{k+1}} Q_*(x_{k+1}, u_{k+1}) \right)$$

Turn into **iterative update**

$$Q(x_k, u_k) \leftarrow \sum_{x_{k+1}, r} p(x_{k+1}, r | x_k, u_k) \left( r + \gamma \max_{u_{k+1}} Q(x_{k+1}, u_{k+1}) \right)$$

Instead of a transition model, use the **transition sample** at each step

$$Q(x_k, u_k) \leftarrow r_{k+1} + \gamma \max_{u_{k+1}} Q(x_{k+1}, u_{k+1})$$

# Q-Learning

Instead of a transition model, use the **transition sample** at each step

$$Q(x_k, u_k) \leftarrow r_{k+1} + \gamma \max_{u_{k+1}} Q(x_{k+1}, u_{k+1})$$

Make update incremental

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha \left( \underbrace{r_{k+1} + \gamma \max_{u_{k+1}} Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)}_{\text{Temporal Difference}} \right)$$

Learning rate $\alpha \in (0, 1]$

# Q-Learning

**Parameters:** step size $\alpha \in (0, 1]$ and $0 < \epsilon < 1$
**Initialize**
$Q(x, u)$ arbitrarily for all possible states and actions,
$Q(\text{terminal}, .) = 0.$
**for** each episode **do**
Initialize $x_0$
**Repeat** for each time step of the episode
Obtain $u_k$ based on $x_k$ using policy $\pi$ derived from $Q$
(e.g. $\epsilon$-greedy)

Take action $u_k$, observe $r_{k+1}$, $x_{k+1}$

$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha \left[ r_{k+1} + \gamma \max_u Q(x_{k+1}, u) - Q(x_k, u_k) \right]$

$k = k + 1$
**Until** the states are terminal
**end for**

# Q-Learning

**Parameters:** step size $\alpha \in (0, 1]$ and $0 < \epsilon < 1$
**Initialize**
$Q(x, u)$ arbitrarily for all possible states and actions,
$Q(\text{terminal}, .) = 0.$
**for** each episode **do**
Initialize $x_0$
**Repeat** for each time step of the episode
Obtain $u_k$ based on $x_k$ using policy $\pi$ derived from $Q$
(e.g. $\epsilon$-greedy)

Take action $u_k$, observe $r_{k+1}$, $x_{k+1}$

$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha \left[ r_{k+1} + \gamma \max_u Q(x_{k+1}, u) - Q(x_k, u_k) \right]$

$k = k + 1$
**Until** the states are terminal
**end for**

In off-policy learning any policy can be used (exploration recommended), the choice displayed here is one of the more common choices

# Exploration - Exploitation

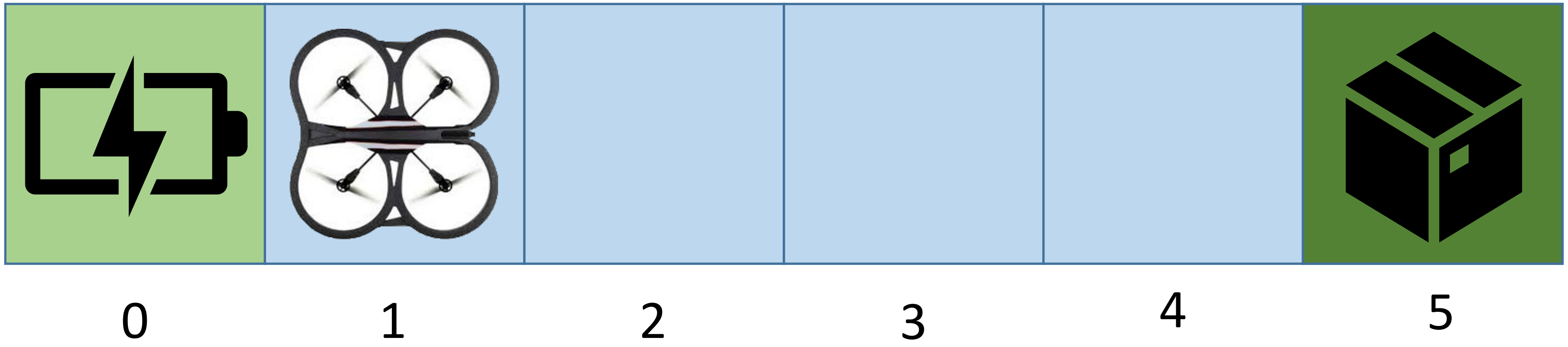
Essential for **convergence** to $Q_*(x_k, u_k)$:

all $(x, u)$ pairs need to be visited infinitely often

**Exploration** necessary, but **exploitation** also necessary:

$\Rightarrow$ $\epsilon$ -greedy approaches are recommended

# Delivery Drone Example




Parameters:

$\alpha = 0.2$ $\varepsilon = 0.3$

$\gamma = 0.5$ $x_0 = 2$ or $3$

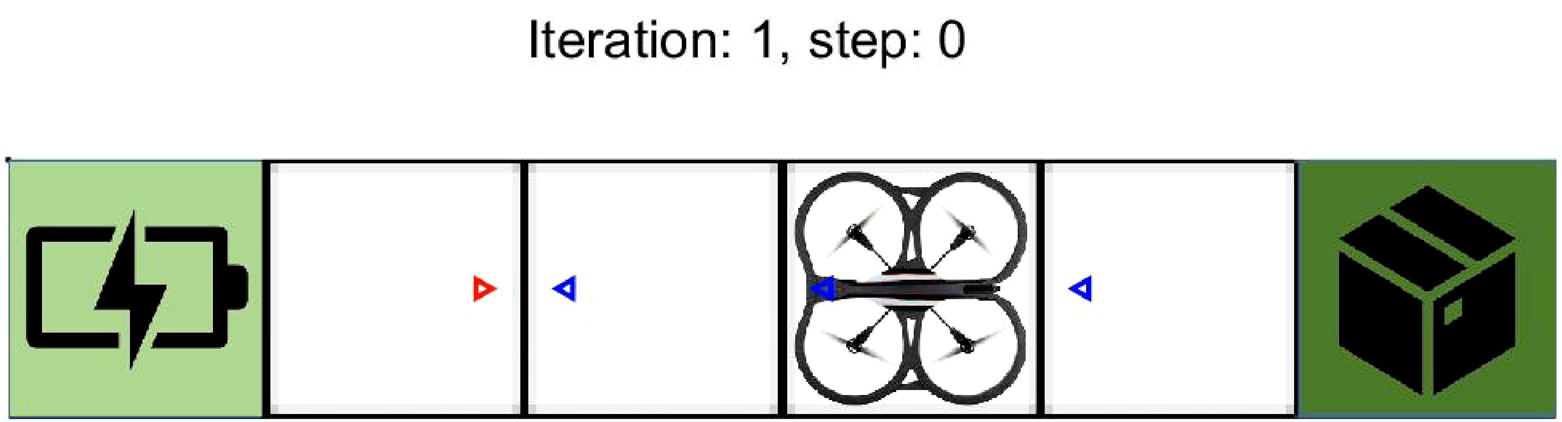
Parameters:

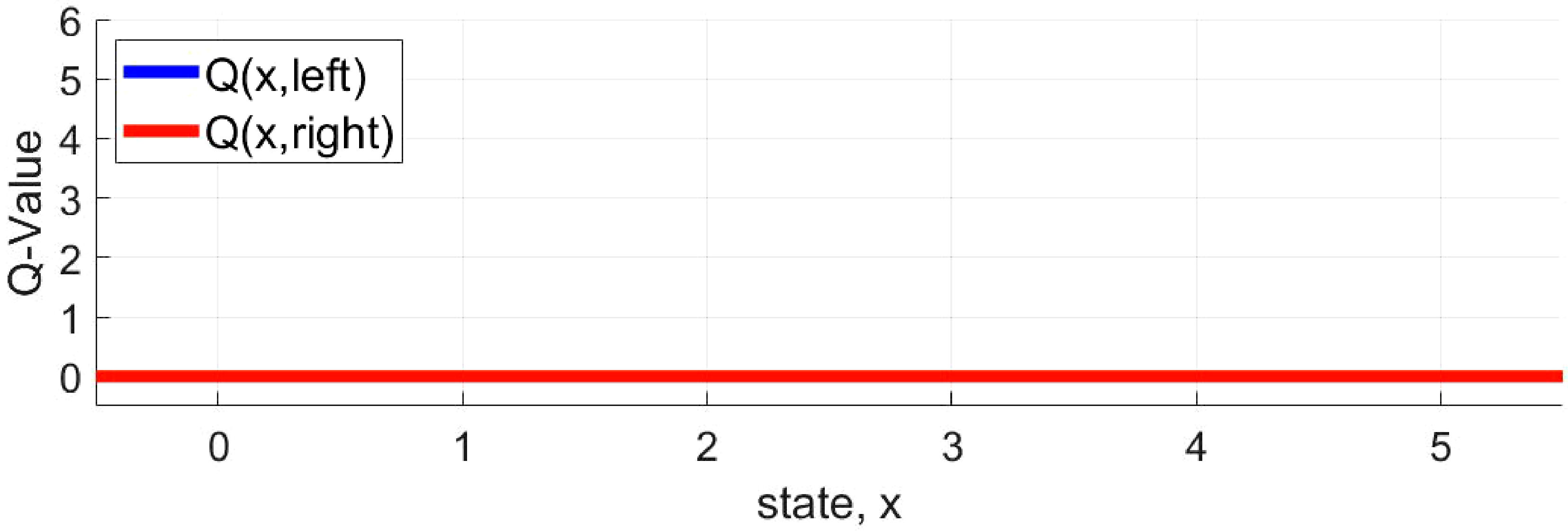
$\alpha = 0.2$

$\varepsilon = 0.3$

$\gamma = 0.5$

$x_0 = 2$ or $3$



# Reinforcement Learning

What is Reinforcement Learning?

Multi-Armed Bandits

Finite Markov Decision Process (Deterministic & Stochastic)

Towards Reinforcement Learning:

Value Function, Q-Function, Bellman Equation

Temporal Difference Learning

Q-Learning

# Remaining Problems

Systems with large state space → large tables!

Continuous state space?