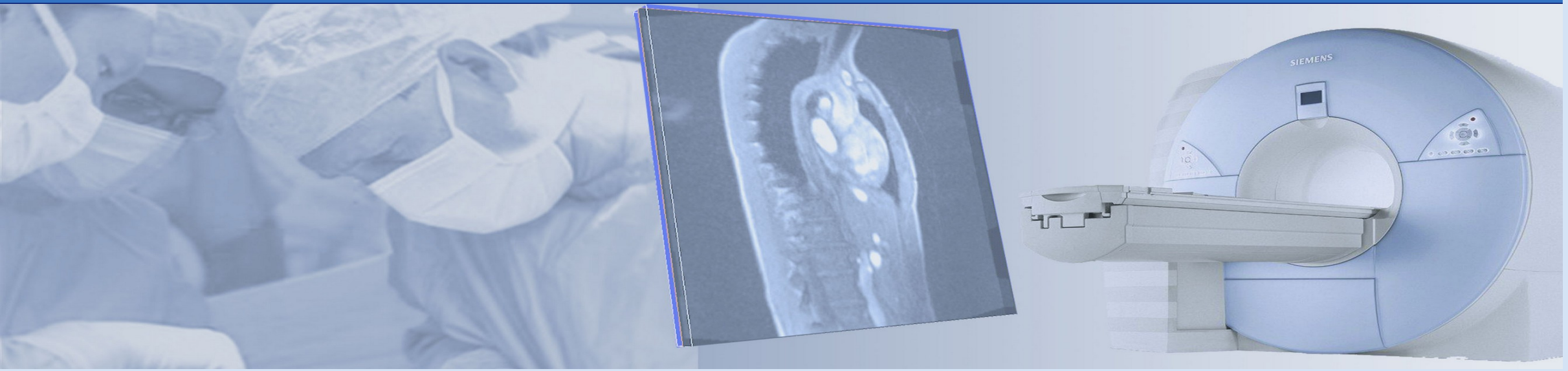
Computer- and robot-assisted Surgery



NCT

NATIONAL CENTER
FOR TUMOR DISEASES
PARTNER SITE DRESDEN
UNIVERSITY CANCER CENTER UCC

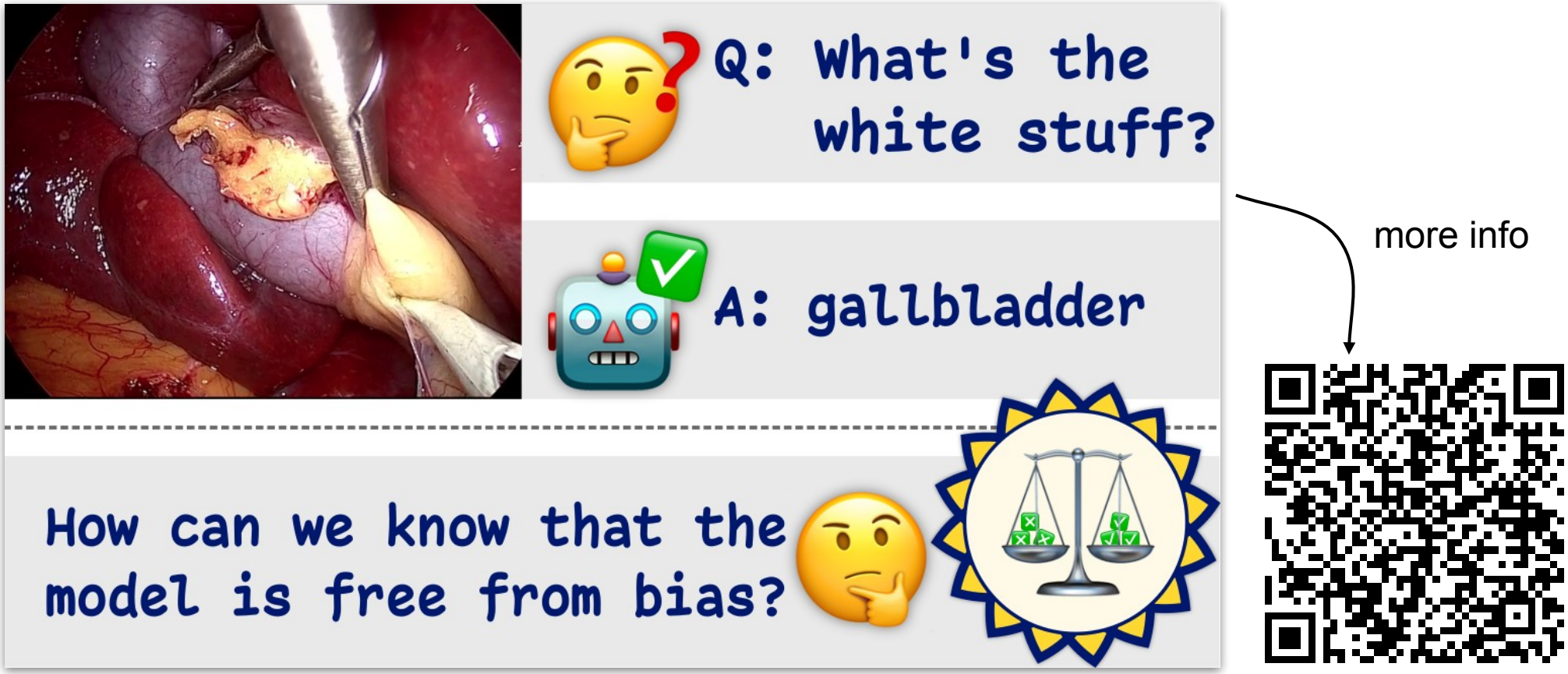
Supported by:
German Cancer Research Center
University Hospital Carl Gustav Carus Dresden
Carl Gustav Carus Faculty of Medicine, TU Dresden
Helmholtz-Zentrum Dresden-Rossendorf

Lecture 14

Machine Learning Basics 3: Reinforcement Learning

Research Project Topic: Minimally Invasive: Benchmarking Tiny VLMs 👈 for Surgical VQA

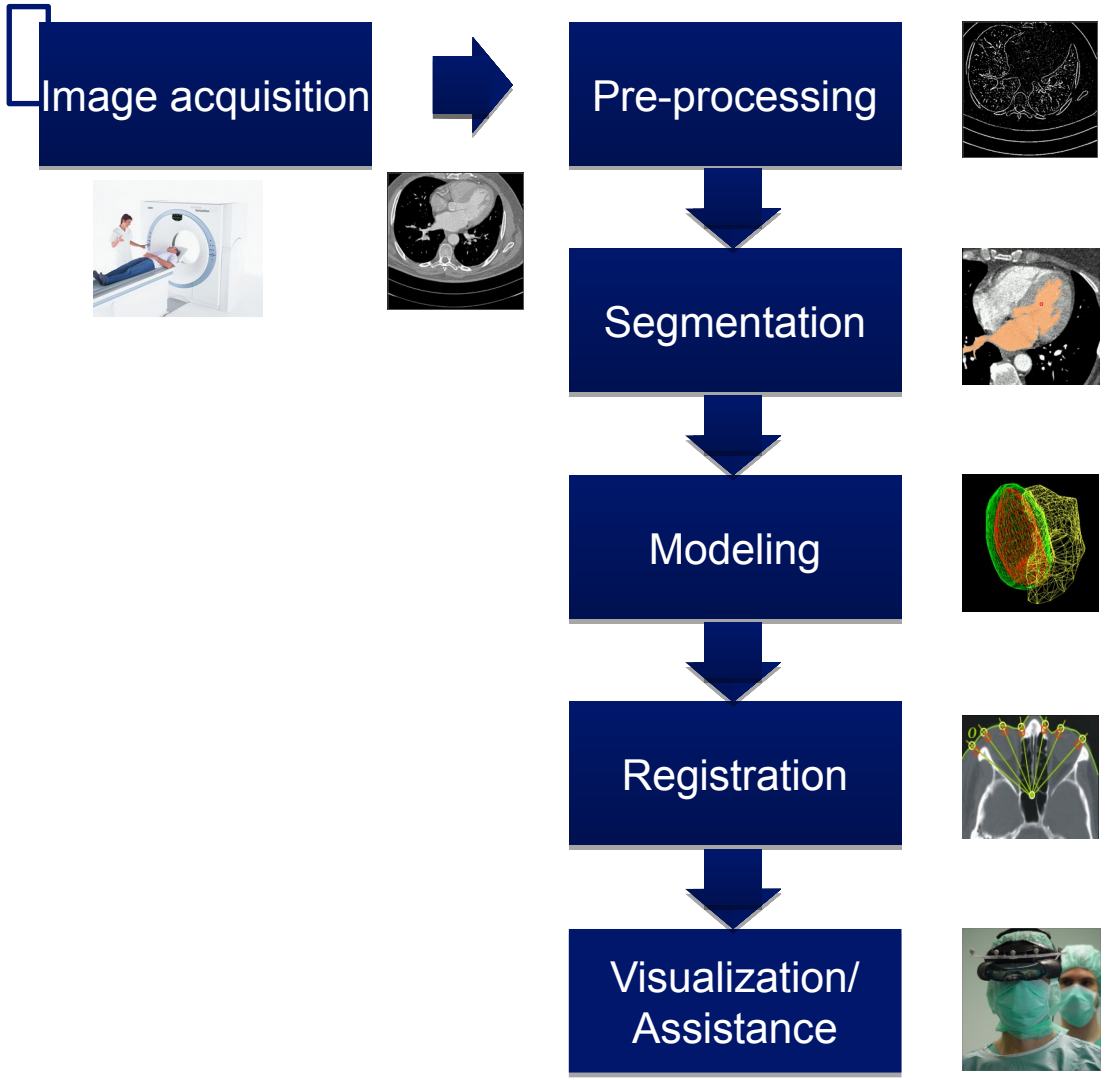
- Supervisor: Claas de Boer (claas.deboer@nct-dresden.de)



Exam

- If you don’t have a date etc yet, contact Ricarda Abdel Bary (ricarda.abdel-bary@nct-dresden.de)

Process chain computer-assisted surgery

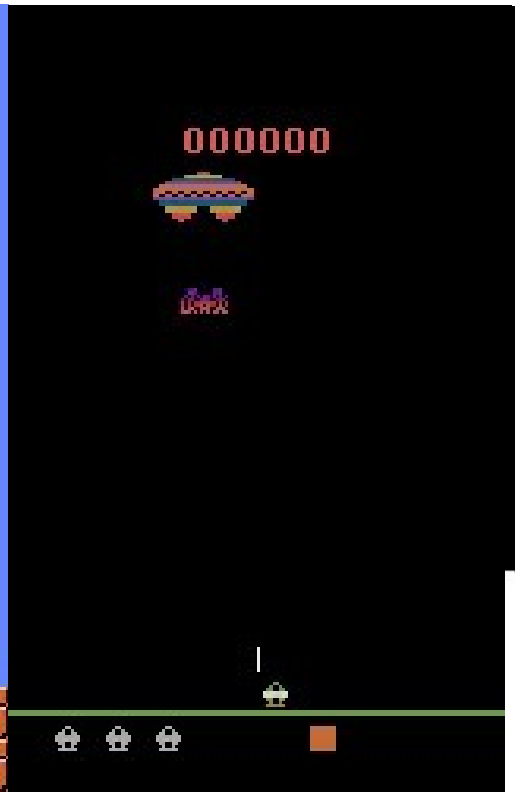


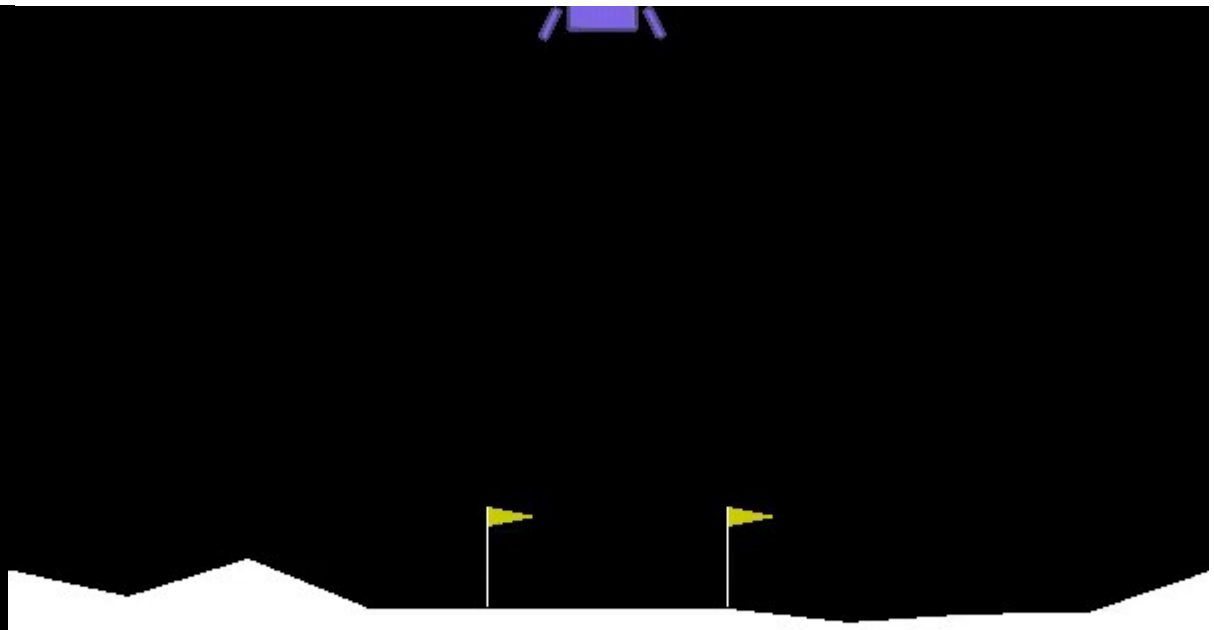
Overview

- Basic idea reinforcement learning
- Markov Decision Process
  - How to optimize
  - Value and policy iteration
- Model-free approaches
  - TD Learning
  - Q-Learning
- Overview advanced topics

Reinforcement Learning: Applications

- Video games



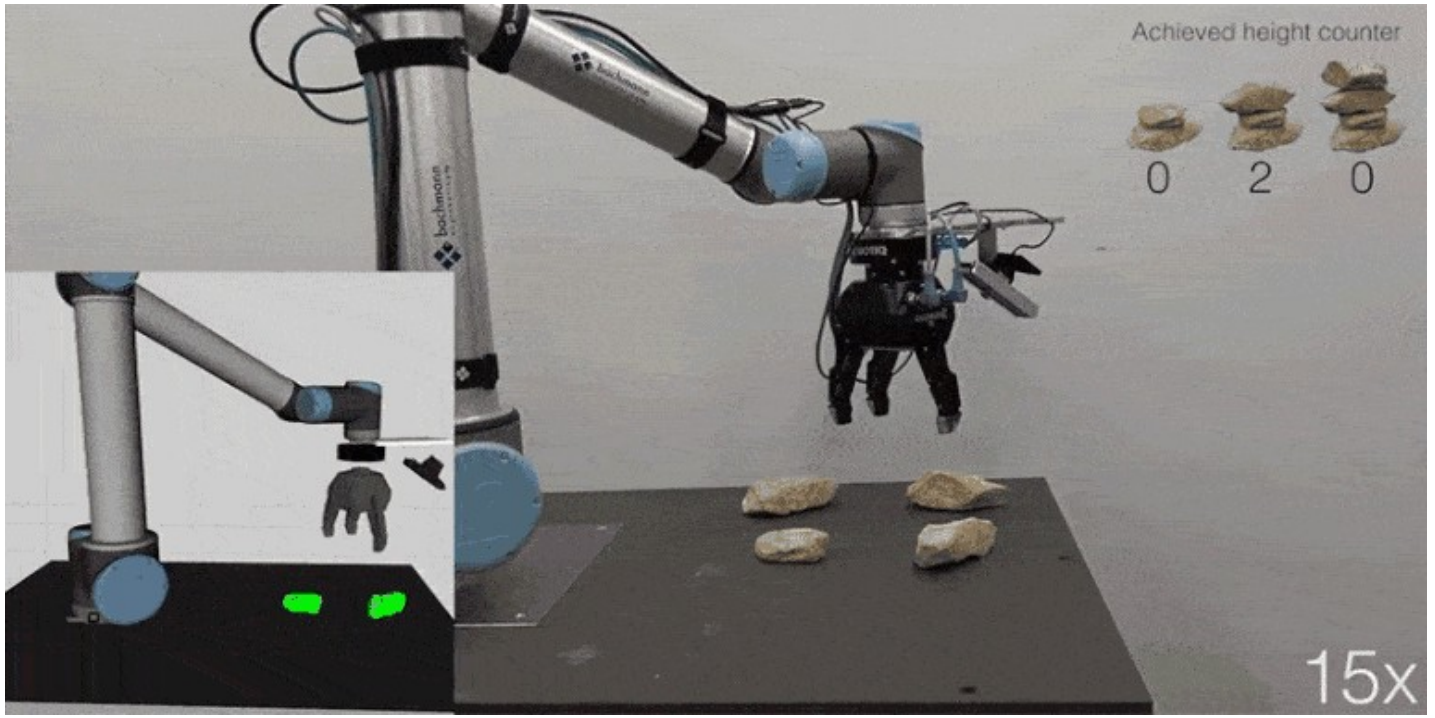


Reinforcement Learning: Applications

- Autonomous vehicle

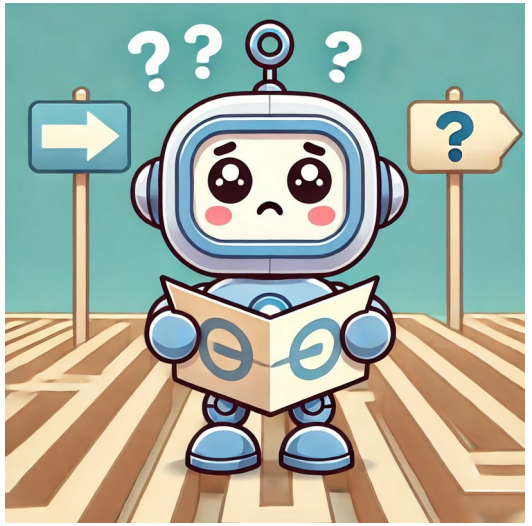
Reinforcement Learning: Applications

- Robotics



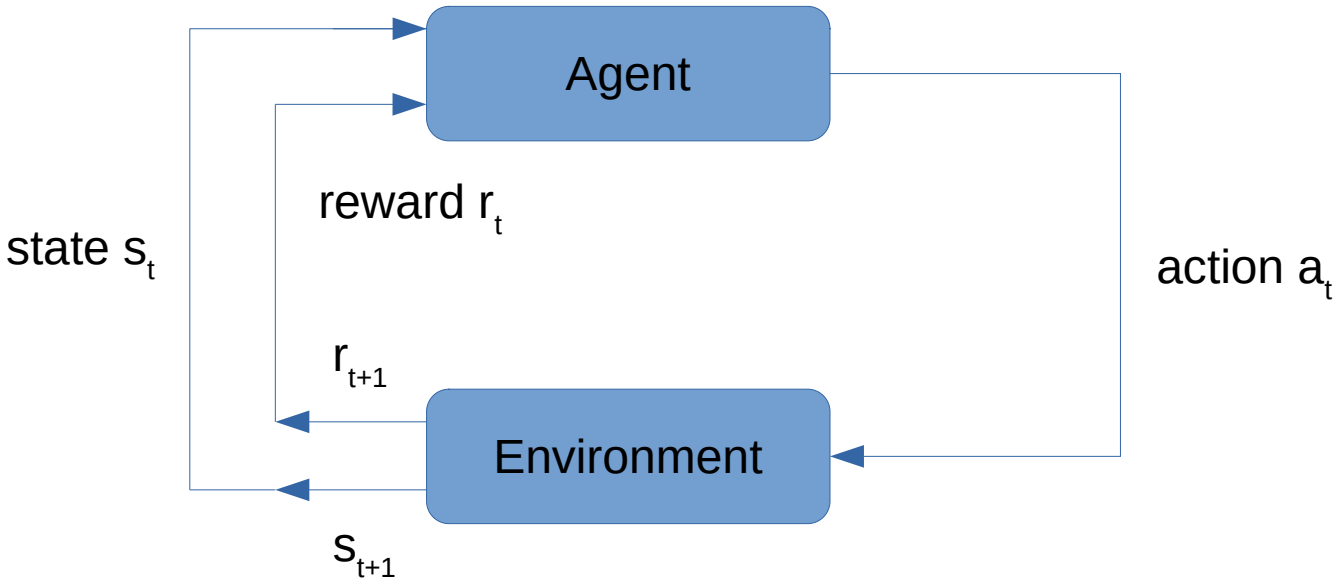
Motivation

- Agent navigating environment
  - E.g. (semi-)autonomous surgical robot, chess playing computer
- Teaching the agent how to act?
  - What action is best?
- Idea: Chess via supervised learning
  - Database with 1 million games => 10^8 examples
  - All possible chess positions => ~10^{111}-10^{123} (about ~10^{40} legal)
  - => Agent is bound to fail



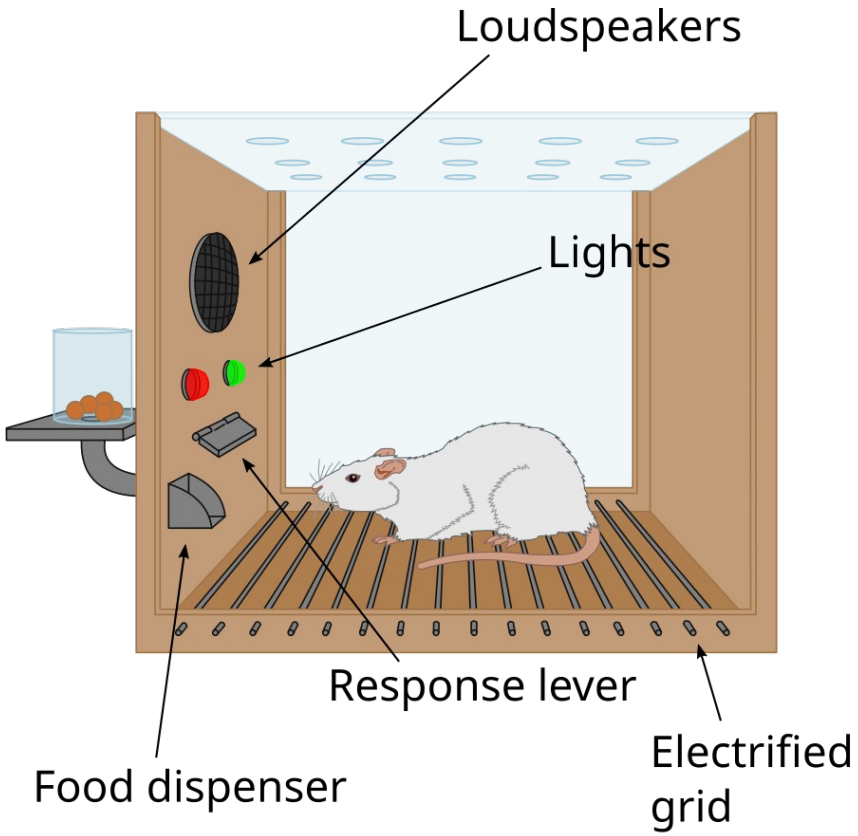
Reinforcement Learning

- Alternative



- An agent in an environment performs an action a_t in state s_t and receives reward r_t.
  - How to maximize the expected reward?

Skinner Box



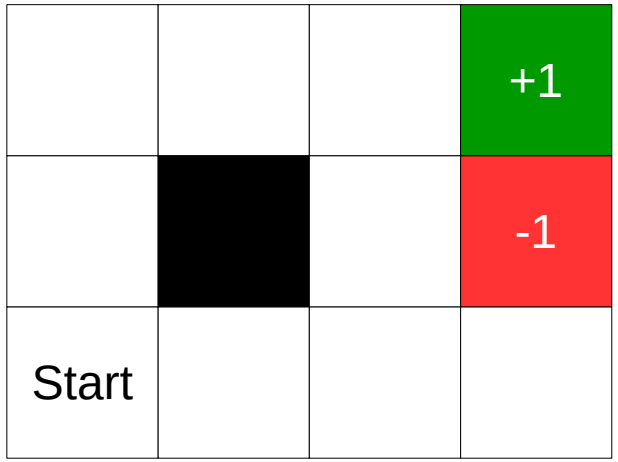
https://commons.wikimedia.org/wiki/File:Skinner_box_scheme_01.svg

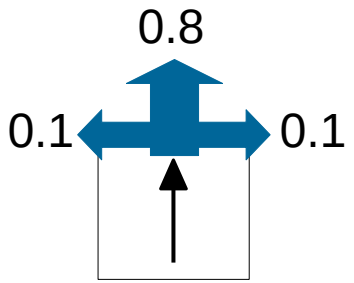
Markov Decision Process (MDP)

- Markov property
  - The probability of a future state only depends on current state

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_1, \ldots, s_t)$$

- MDP
  - State changes: $P(s'|s,a)$
  - Rewards: $R(s,a,s')$
- Example





Markov Decision Process (MDP): Policies

- Policy π: Specifies what the agent does in any state

$$\pi(s) = a$$

- Utility function U^{π}: expected reward when executing π

$$U^{\pi}(s) = \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1})], S_0 = s, \gamma \in [0,1]$$

  - Discount factor γ:
    - $\gamma = 0$: Disregard future rewards, maximize reward for next state
    - $0 \leqslant \gamma < 1$: Future rewards count, but are discounted
    - $\gamma = 1$: All rewards count equal, no matter when received
- Optimal policy π*: Yields the highest expected utility

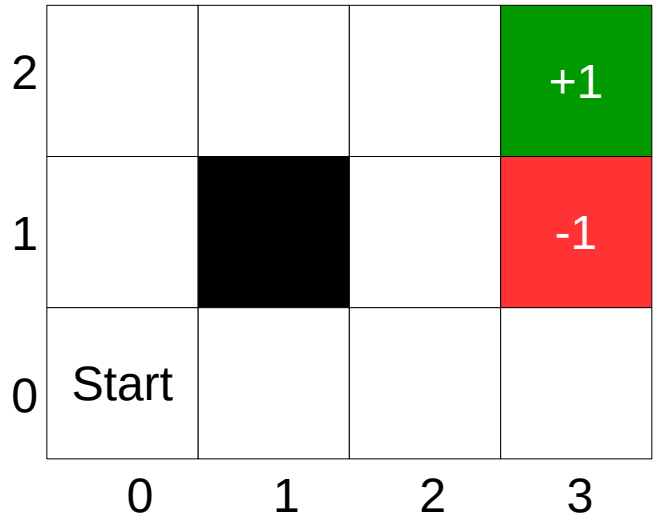
$$\pi^*(s) = \underset{\pi}{\operatorname{argmax}}\, U^{\pi}(s)$$

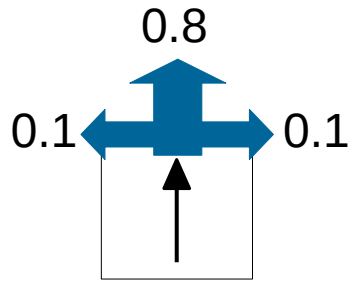
NCT

Bellman equation

- To find the optimal policy π*, we need to know U^{π^*}

$$U^{\pi}(s)=U(s)=\max_{a\in A(s)}\sum_{s'}P(s'|s,a)[R(s,a,s')+\gamma U(s')]$$





$$U(0,0)=\max\{$$

up $[0.8(\gamma U(0,1))+0.1(\gamma U(1,0))+0.1(\gamma U(0,0))]$,

right $[0.8(\gamma U(1,0))+0.1(\gamma U(0,1))+0.1(\gamma U(0,0))]$,

left $[0.9(\gamma U(0,0))+0.1(\gamma U(0,1))]$,

down $[0.9(\gamma U(0,0))+0.1(\gamma U(1,0))]]$

Q-function

- Action-utility (or Q)-function Q(s,a):
  - expected utility of action a in state s

$$U(s) = \max_a Q(s,a)$$

  - Can be used to calculate the optimal policy

$$\pi^*(s) = \operatorname{argmax}_a Q(s,a)$$

  - We can also develop a Bellman equation for U

$$Q(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U(s')]$$
$$= \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q(s',a')]$$

  - Difference U(s) and Q(s, a)
    - U tells us how good a state is, not how to get there
    - Q tells us the expected return if we take action a in state s
      - Picking the action with the highest reward results in an optimal policy

NCT

Policy iteration

- Goal: Find the optimal policy π* by iteratively improving an initial policy.
- 2 steps
  - Evaluate a given policy π_i, by calculating its utility function

$$U^{\pi_i}(s) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U^{\pi_i}(s')]$$

  - Policy improvement by greedy selection of action with highest utility

$$\pi_{i+1}(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U^{\pi_i}(s')]$$

  - Iterate until convergence
- Convergence to optimal policy π*
- Can be solved via dynamic programming

Value iteration

- Skips explicit policy evaluation and improves values directly
  - Initialize U(s) randomly for all states s
  - Loop
    - Loop for each $s \in S$
      - $U(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U(s')]$
    - Iterate until convergence
  - Extract policy

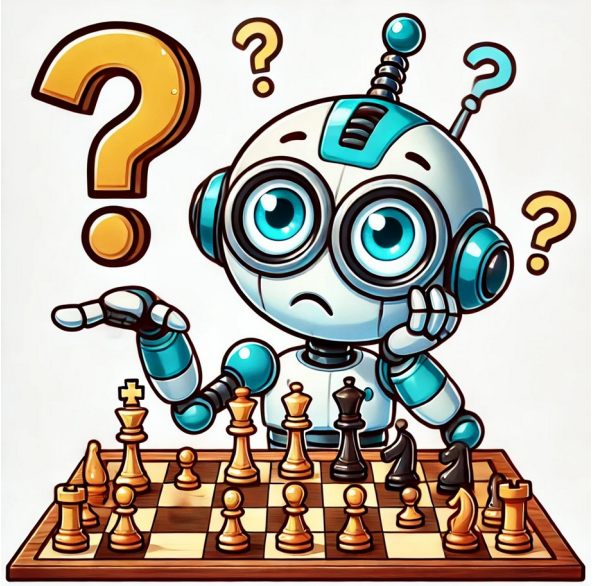
$$\pi^*(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U(s')]$$

- Also converges to optimal policy π*
- Can also be solved via dynamic programming
- Policy Iteration: stable and interpretable policy update in each step
- Value Iteration: computational efficiency is key and/or large state spaces

NCT

Where to use them?

- Value or policy iteration for games like chess?
  - Requires full knowledge of environment, i.e. transition probabilities
    - Chess has a deterministic transition model
  - Requires storing values for all states
    - Chess: ~10^{111}-10^{123} (about ~10^{40} legal)
- => work well for simpler environments, e.g.
  - Network routing
  - Elevator control system
  - Traffic light control
- But more complex environments, like chess, are infeasible



Temporal Difference (TD) Learning

- Instead of computing values for all states, passively learn from experience
  - Use observed transitions to adjust utilities
- Input current state s', previous state s, observed reward r
  - If s' is new: $U[s'] \leftarrow 0$
  - $N_s = N_s + 1$: count how often state s was visited
  - $U[s] \leftarrow U[s] + \alpha(N_s)(r + \gamma U[s'] - U[s])$: update previous estimate
    - α: learning rate, decreases with number of visits to s
- Is learned passively (e.g. watching 2 chess agents play)
- Model free: no knowledge of MDP transitions required!
  - Though transition model is needed to extract a policy!
- A function approximator, e.g. neural network, can be used

NCT

Less structured environments

- TD-Learning in less structured environments, e.g. video games
  - Generally high-dimensional and not discrete states!
    - ATARI games: 210x160x3 image (RGB)
  - States not always directly observable
  - Transition function not necessarily known
    - How to calculate next step?

Q-Learning

- Modified TD-Learning approach for Q-function Q(s, a)
  - Allows agent to choose action
- Input current state s', previous state s and action a, observed reward r
  - $N_{sa} = N_{sa} + 1$: count how often action a was performed in state s
  - $Q[s,a] \leftarrow Q[s,a] + \alpha(N_{sa})(r + \gamma \max_{a'} Q[s',a'] - Q[s,a])$: update previous estimate
    - α: learning rate, decreases with number of visits to s
- Learns actively, i.e. provides action
- Policy extraction: $\pi(s) = \operatorname{argmax}_a Q(s,a)$
- Can be used to train function approximator, e.g. neural network

Q-Learning: Exploitation vs Exploration

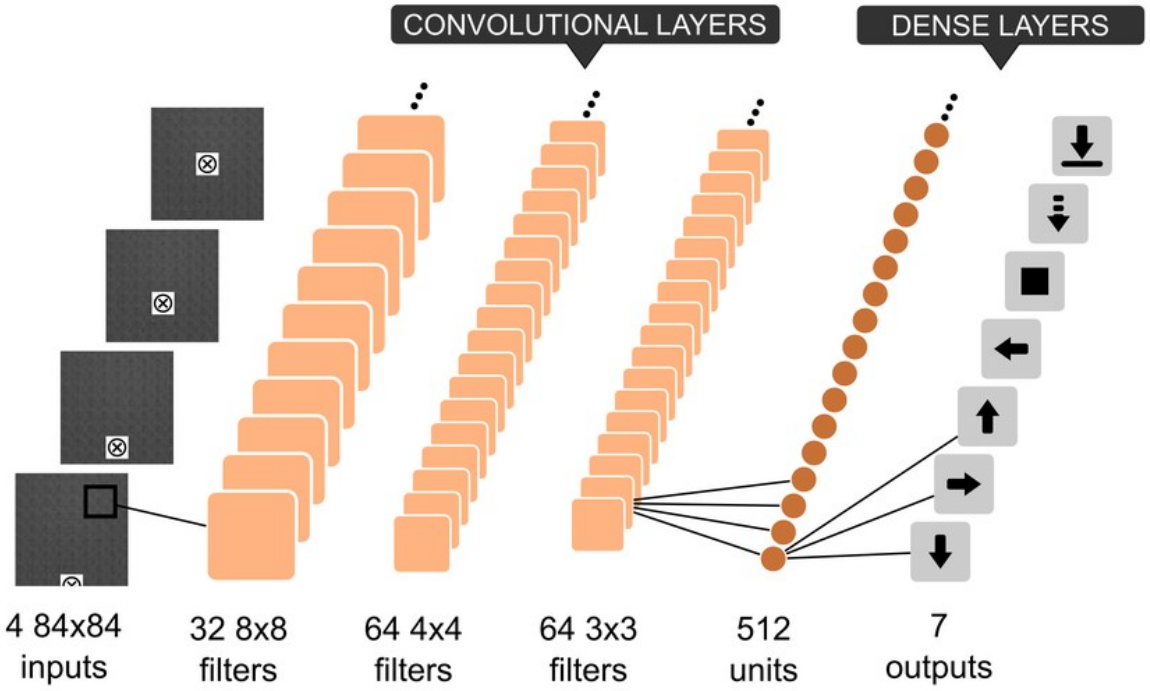
- Which action a to select for the current state s?
  - Action which maximizes Q(s,a): local learning (Exploitation)
    - Known actions are examined, leads to high rewards
    - actions that were not selected before, might not be selected
  - Random action: global learning (Exploration)
    - Discover better actions, not tried before
    - Can lead to suboptimal moves in the short term
- ε-greedy strategy

$$a = \begin{cases} \text{argmax}_a Q(s,a) & \text{with probability } 1-\epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$$

Decay ε over time (e.g. start with ε = 1 and gradually reduce it)

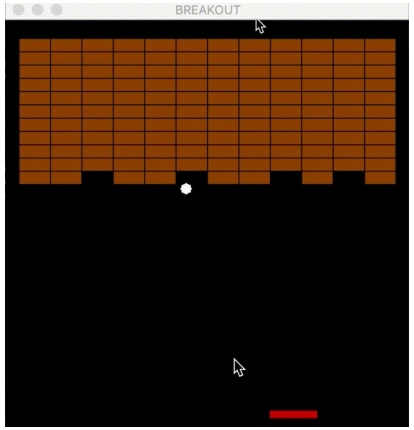
NCT

Q-Learning for video games (Deep Q Networks)



Algorithm 1 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 $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for



Mnih, Volodymyr. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

NCT

Policy Learning

- What if our actions are not discrete or high dimensional?
  - e.g. controlling a car where the steering operates in degrees
- Learn policy π(a|s) directly, e.g. via neural network
  - Perform policy gradient to maximize the expected reward J(θ)

$$J(\theta) = E_{\pi}\left[\sum_{t=0}^{T} r_t\right], \theta = \text{network parameters}$$

  - Optimize by using the gradient as cost function

$$\nabla_{\theta} J(\theta) = E_{\pi}[\nabla_{\theta} \log \pi_{\theta}(a|s) r]$$

- Directly output expected reward per action
  - Optimize policy parameters to maximize future rewards

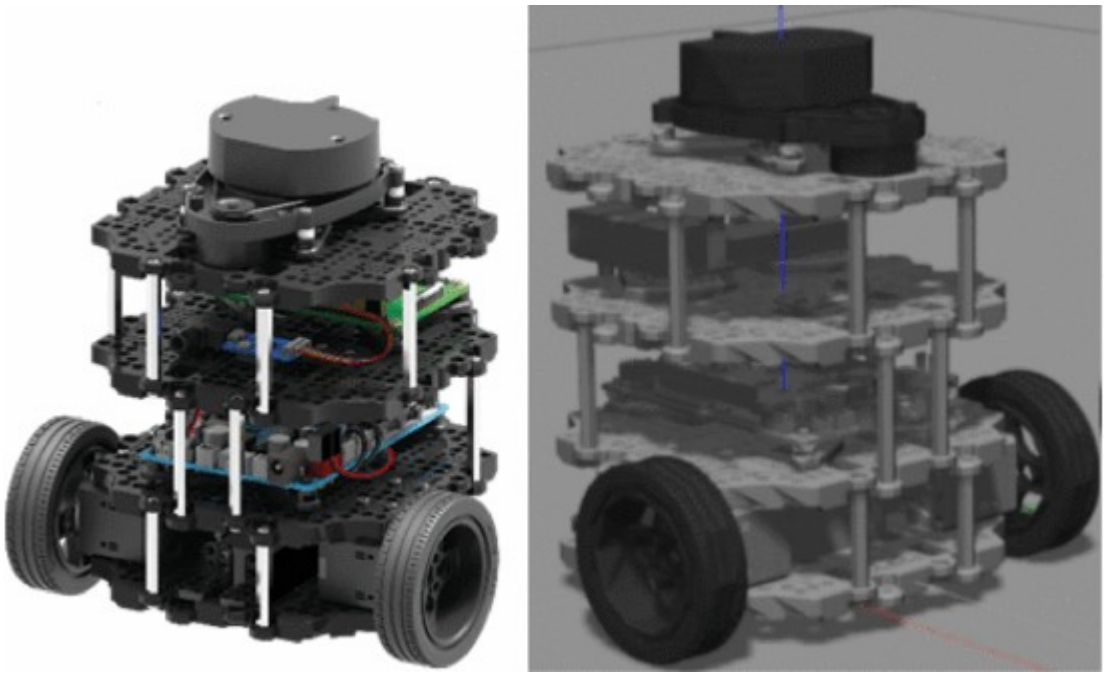
NCT

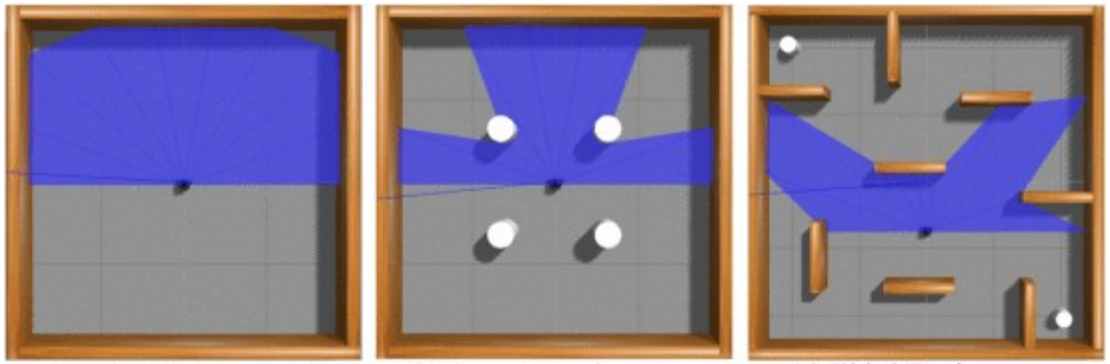
Policy Learning: REINFORCE

- How to implement
  - Initialize $\pi_\theta(a|s)$
  - Loop
    - Collect episode by sampling actions from $\pi_\theta(a|s)$
    - Compute rewards r_t for each step t
    - Update policy parameters: $\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) r_t$

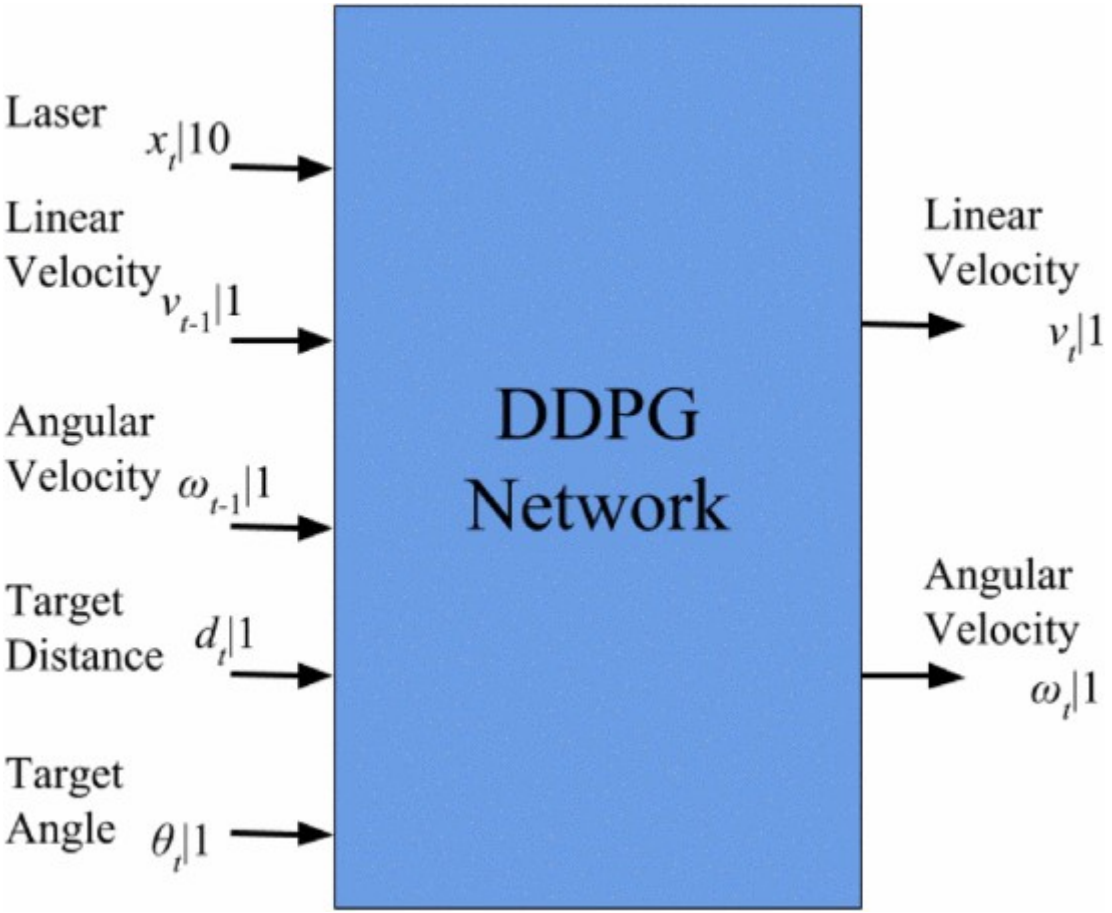
NCT

Application robot navigation





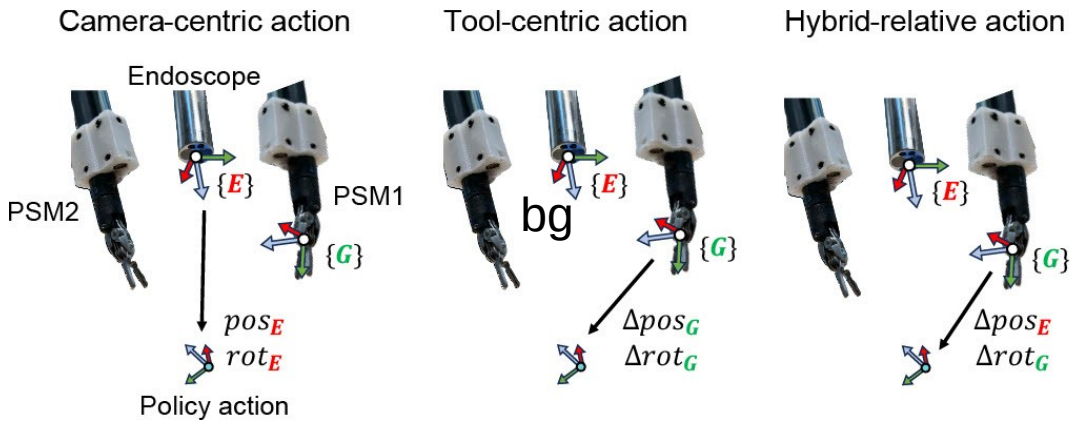
(a) First Environment (b) Second Environment (c) Third Environment



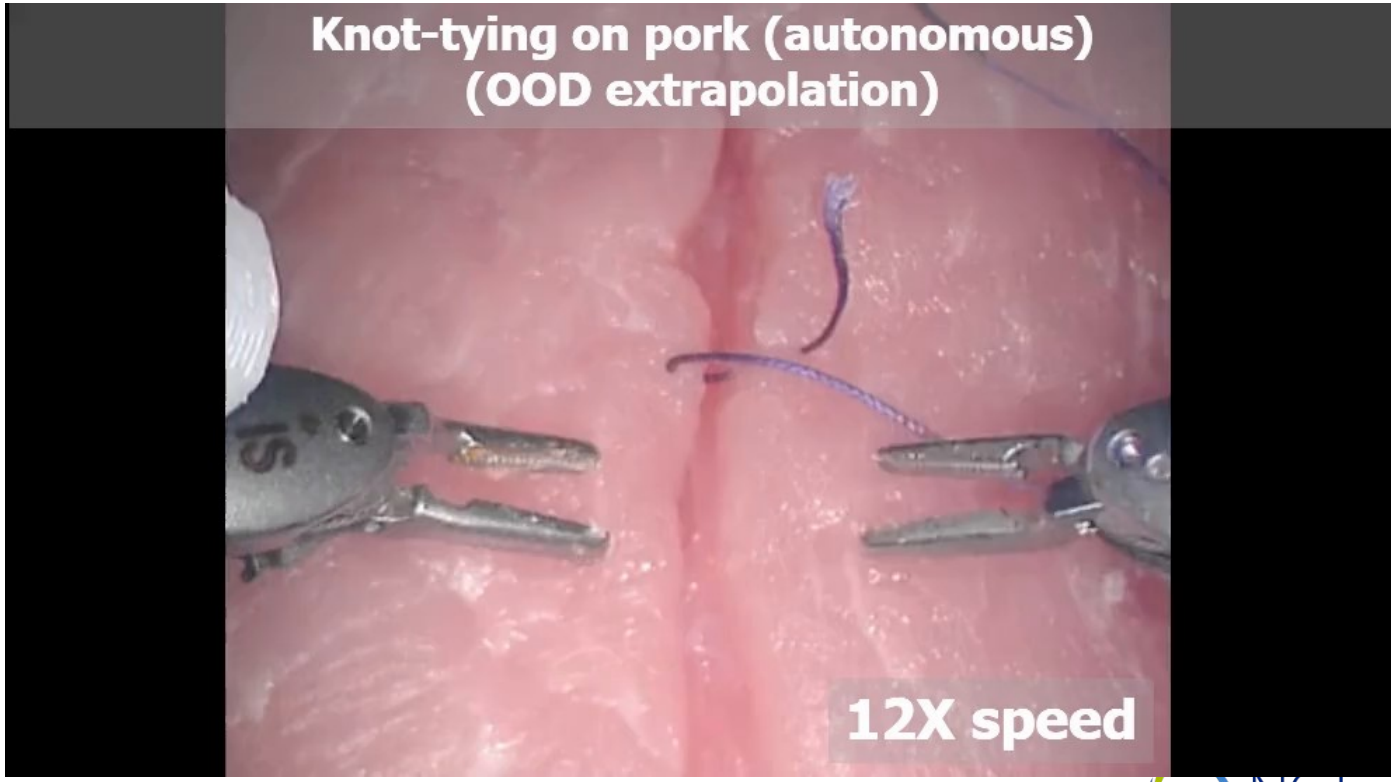
Advanced topics: Imitation learning

- Can we incorporate expert knowledge?
  - Training from scratch is slow
- Imitation Learning: the agent learns directly from demonstrations
  - Agent observes state-action pairs [s_t, a_t]
  - Agent learns a policy $\pi(s|\ a)$ that mimics expert
- Avoids exploration
- Widely used, e.g. self-driving cars
- Doesn’t generalize well
  - Can fail in unseen states

Imitation Learning for surgery



https://surgical-robot-transformer.github.io/



Imitation Learning for surgery @NCT

Advanced topics: Inverse Reinforcement Learning

- Traditional RL requires reward function
  - Difficult to define in many real-world scenarios
- Inverse RL aims to infer a reward function from demonstrations
  - What goal is the expert trying to optimize?
- Approach
  - Observe state-action pairs $[s_t, a_t]$
  - Infer reward function r(s, a)
  - Optimize a policy given the reward function