Active Reinforcement Learning

Pradipta Biswas

Associate Professor

Indian Institute of Science

pradipta@iisc.ac.in, https://cambum.net/PB/

Difference with Passive RL

- A passive learning agent **has a fixed policy** that determines its behaviour. An active agent must decide what action to take
- An active agent requires outcome probabilities of ALL ACTIONS rather than for a fixed policy as used in passive RL
- An active agent EXPLORES the world
- Trade off between Exploration and Exploitation
  - Sticking to only known world ensures stability but may lead to sub-optimal solution
  - Exploring new opportunities lead to improve the present situation

Problem with Optimal Policy of an ADP Agent

Agent learns a model not the true environment !!

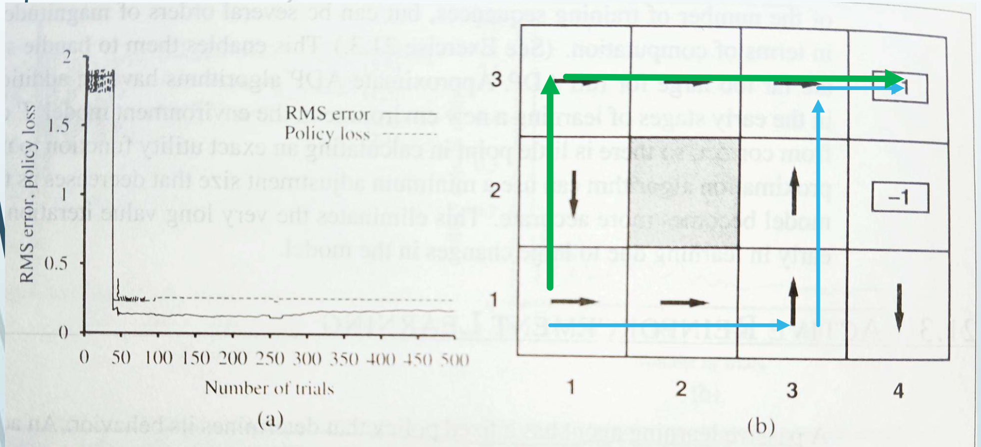


Figure 21.6 Performance of a greedy ADP agent that executes the action recommended by the optimal policy for the learned model. (a) RMS error in the utility estimates averaged over the nine nonterminal squares. (b) The suboptimal policy to which the greedy agent converges in this particular sequence of trials.

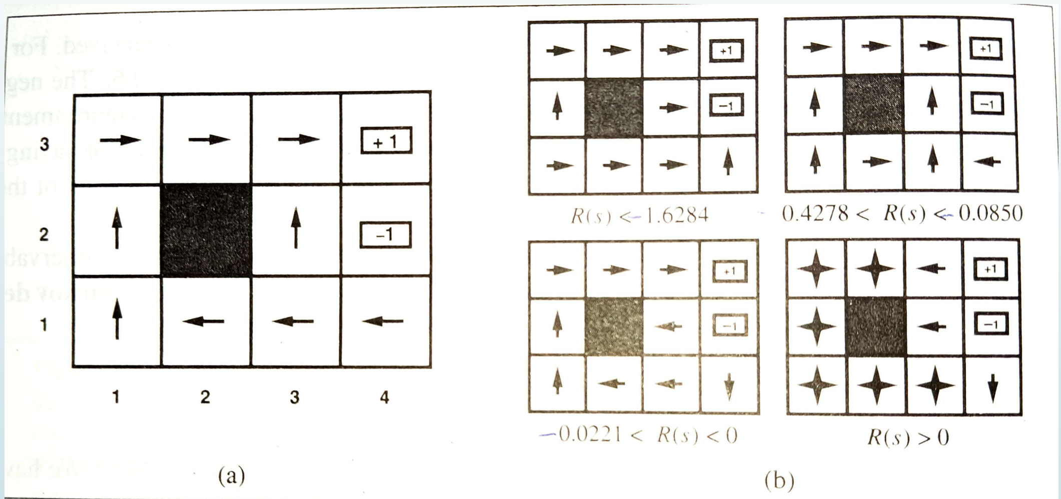


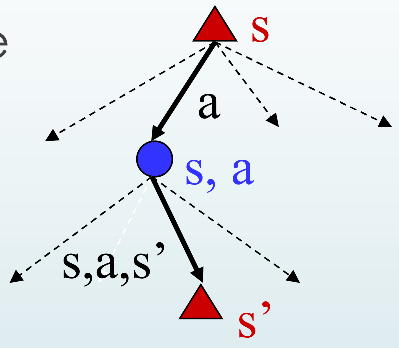
Figure 17.2 (a) An optimal policy for the stochastic environment with $R(s) = -0.04$ in the nonterminal states. (b) Optimal policies for four different ranges of $R(s)$.

Exploration Function

- Two new functions
- N(a, s) = How many times action a is executed at state s
- **Exploration function f (u, n)**
  - Increasing in u and decreasing in n
  - Greed is traded off with curiosity
  - $f(u, n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & \text{otherwise} \end{cases}$
    - where R+ is an optimistic estimate of the best possible reward obtainable in any state and Ne is a fixed parameter
    - Ensures each state-action pair will be tried at least Ne times

Problems with TD Value Learning

- TD learns utility values in local neighborhood instead of the wholw state space
- We want to turn values into a (new) policy



$$\pi(s) = \arg\max_a Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma U^*(s')\right]$$

- Idea: learn Q-values directly
- Makes action selection model-free too!

Q-Learning

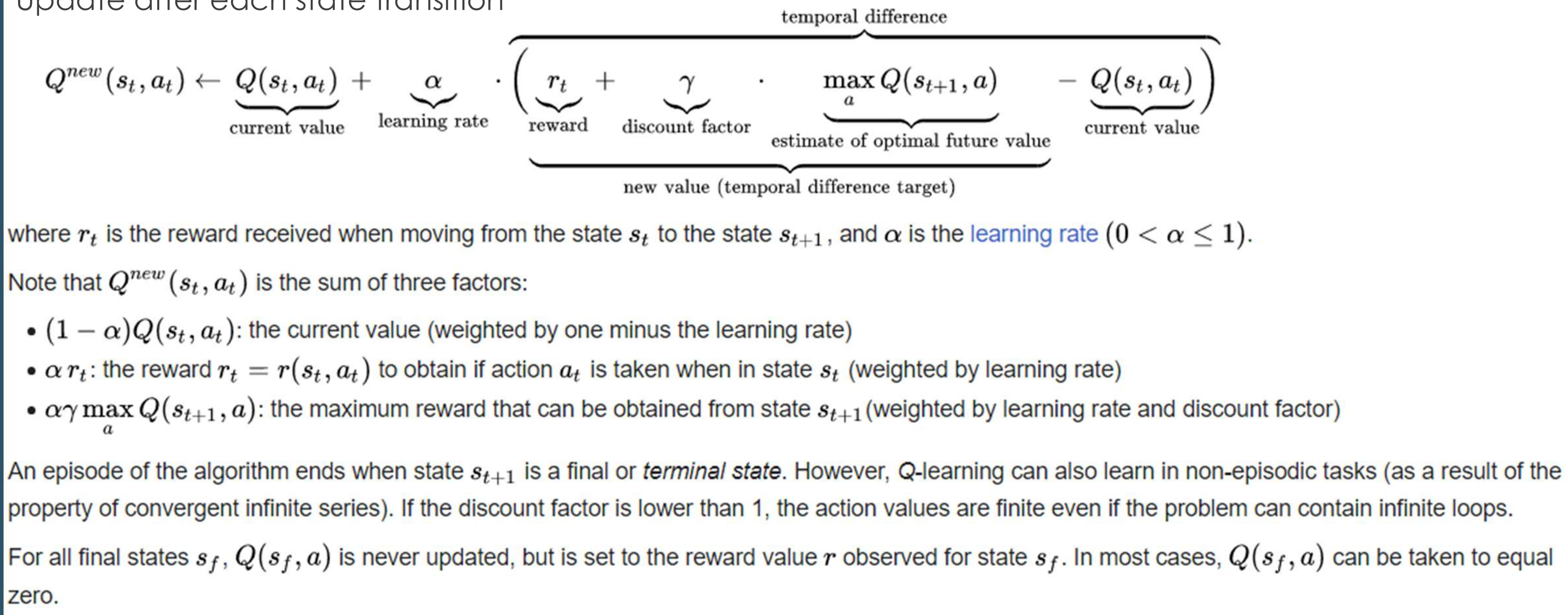
- Define a new function Q (a, s)

Quality of an Action

- Relationship with Utility: $U(s)=\max_a Q(a, s)$
- Constraint Equation for Equilibrium: $Q(a, S) = R(S) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(a', s')$
- TD Update for Utility: $U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$
- Q learning with TD update: $Q(a, S) \leftarrow Q(a, S) + \alpha(R(S) + \gamma \max_{a'} Q(a', s') - Q(a, s)$

Q - Learning

Update after each state transition



$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{current value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\bigg(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t, a_t)}_{\text{current value}} \bigg)}^{\text{temporal difference}}$$

where r_t is the reward received when moving from the state s_t to the state s_{t+1}, and α is the learning rate $(0 < \alpha \leq 1)$.

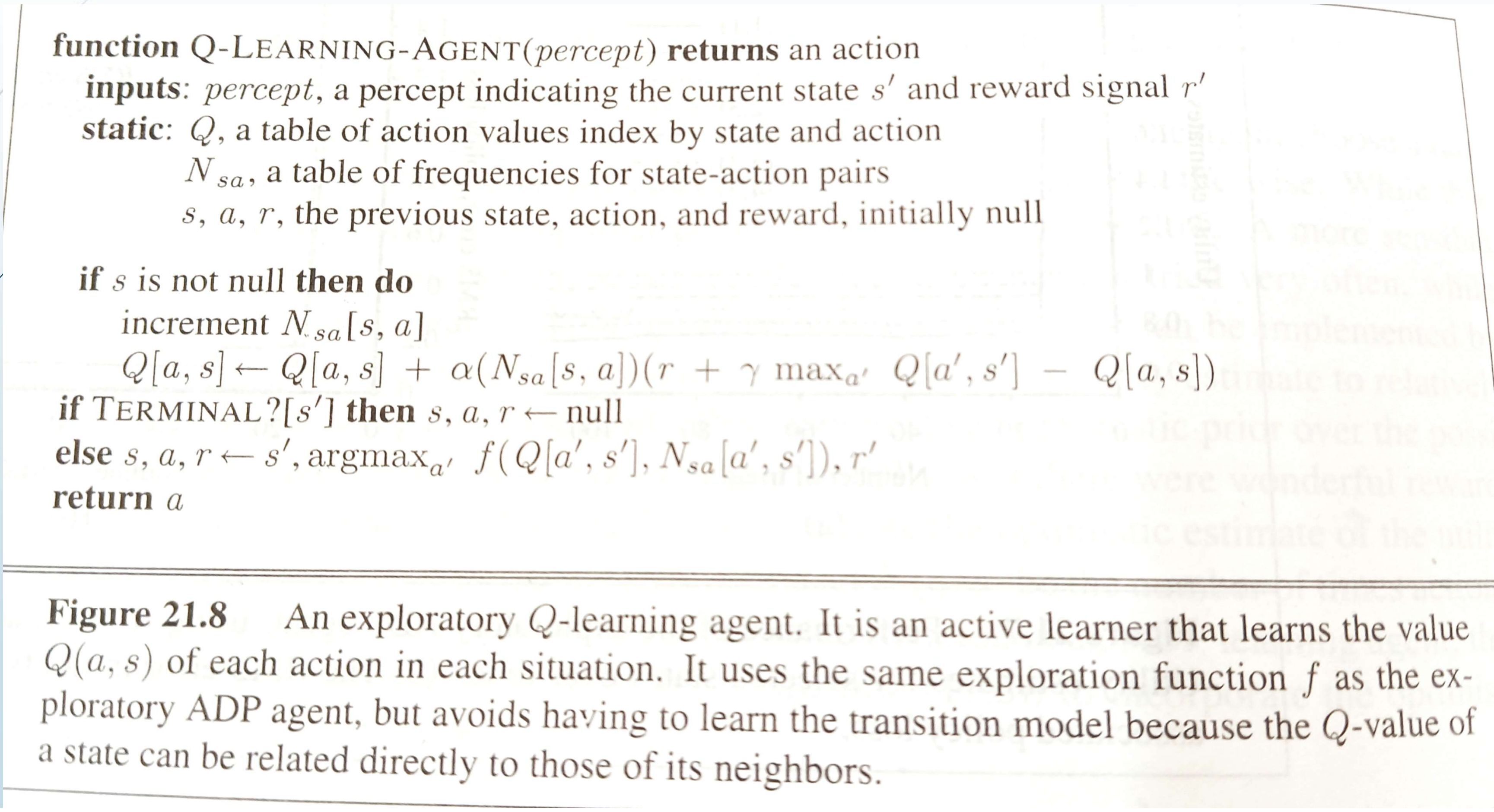
Note that $Q^{new}(s_t, a_t)$ is the sum of three factors:

- $(1 - \alpha)Q(s_t, a_t)$: the current value (weighted by one minus the learning rate)
- $\alpha \, r_t$: the reward $r_t = r(s_t, a_t)$ to obtain if action a_t is taken when in state s_t (weighted by learning rate)
- $\alpha\gamma \max_a Q(s_{t+1}, a)$: the maximum reward that can be obtained from state s_{t+1} (weighted by learning rate and discount factor)

An episode of the algorithm ends when state s_{t+1} is a final or *terminal state*. However, Q-learning can also learn in non-episodic tasks (as a result of the property of convergent infinite series). If the discount factor is lower than 1, the action values are finite even if the problem can contain infinite loops.

For all final states s_f, $Q(s_f, a)$ is never updated, but is set to the reward value r observed for state s_f. In most cases, $Q(s_f, a)$ can be taken to equal zero.

Q – Learning implementation



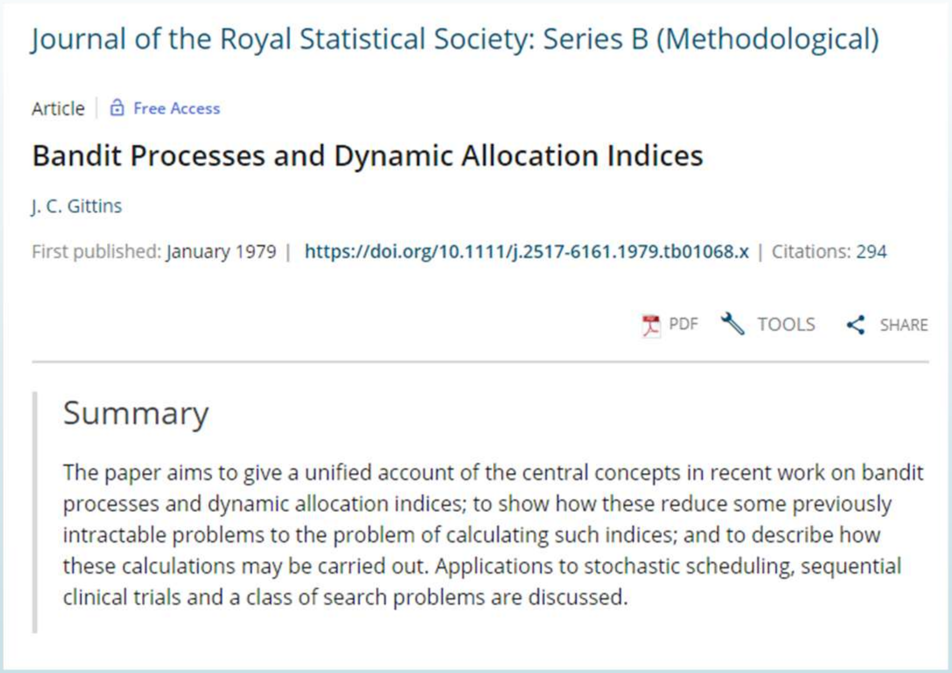
```
function Q-LEARNING-AGENT(percept) returns an action
  inputs: percept, a percept indicating the current state $s'$ and reward signal $r'$
  static: $Q$, a table of action values index by state and action
          $N_{sa}$, a table of frequencies for state-action pairs
          $s$, $a$, $r$, the previous state, action, and reward, initially null

  if $s$ is not null then do
      increment $N_{sa}[s, a]$
      $Q[a, s] \leftarrow Q[a, s] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[a', s'] - Q[a, s])$
  if TERMINAL?[$s'$] then $s, a, r \leftarrow$ null
  else $s, a, r \leftarrow s', \operatorname{argmax}_{a'} f(Q[a', s'], N_{sa}[a', s']), r'$
  return $a$
```

Figure 21.8 An exploratory Q-learning agent. It is an active learner that learns the value $Q(a, s)$ of each action in each situation. It uses the same exploration function f as the exploratory ADP agent, but avoids having to learn the transition model because the Q-value of a state can be related directly to those of its neighbors.

Quest for Optimal Exploration

- GLIE – Greedy in the Limit of Infinite Exploration
- Try each action in each state an unbounded number of times
- **Gittins index** is an effort to find optimal exploration policy
- The multi-armed bandit problem is a problem in which a fixed limited set of resources must be allocated between competing (alternative) choices in a way that maximizes their expected gain, when each choice's properties are only partially known at the time of allocation, and may become better understood as time passes or by allocating resources to the choice.


Journal of the Royal Statistical Society: Series B (Methodological)

Article | Free Access

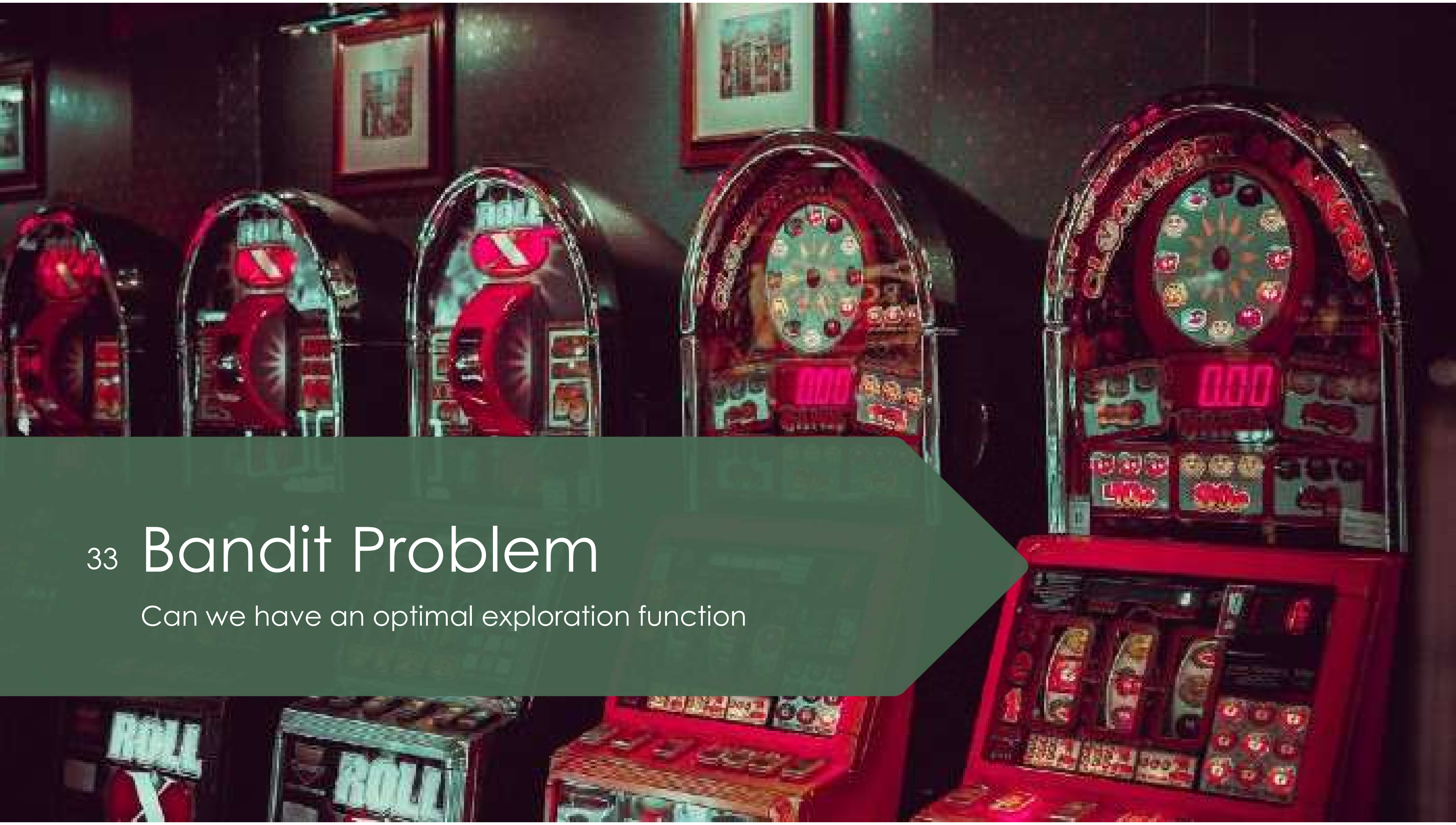
Bandit Processes and Dynamic Allocation Indices

J. C. Gittins

First published: January 1979 | **https://doi.org/10.1111/j.2517-6161.1979.tb01068.x** | Citations: 294

Summary

The paper aims to give a unified account of the central concepts in recent work on bandit processes and dynamic allocation indices; to show how these reduce some previously intractable problems to the problem of calculating such indices; and to describe how these calculations may be carried out. Applications to stochastic scheduling, sequential clinical trials and a class of search problems are discussed.



33 Bandit Problem

Can we have an optimal exploration function

Virtual obstacle avoidance & heterogenous multi-robot coordination in mixed reality with a human in the loop

Ajay Kumar Sandula, Dr Pradipta Biswas, Arushi Khokhar, Dr Debasish Ghose

I3D Lab

RBCCPS Department

Indian Institute of Science, Bangalore

Human-In-The-Loop Multi-Robot Systems

Ajay Kumar Sandula, PhD student, IISC Bangalore

Advisors: Dr Pradipta Biswas, Dr Debasish Ghose

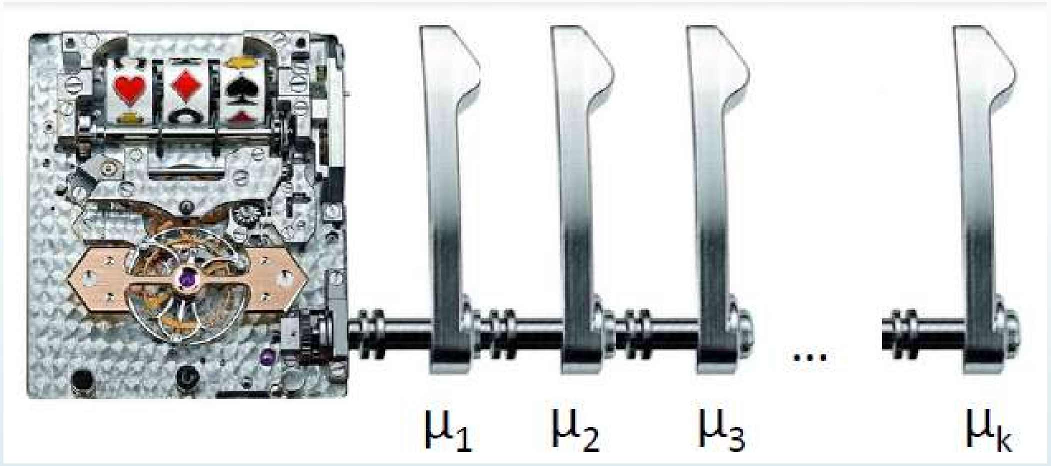
INDIAN INSTITUTE OF SCIENCE

भारतीय विज्ञान संस्थान

RBCCPS

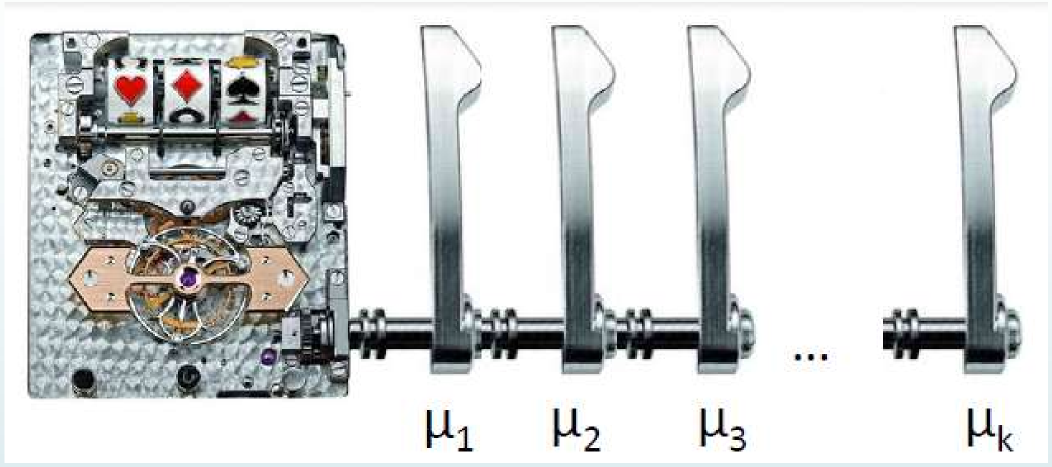
Robert Bosch Centre for Cyber-Physical Systems

k-Armed Bandit



- **Each arm *a***
  - **Wins** (reward=**1**) with fixed (unknown) prob. μ_a
  - **Loses** (reward=**0**) with fixed (unknown) prob. $1-\mu_a$
- All draws are independent given $\mu_1 \ldots \mu_k$
- **How to pull arms to maximize total reward?**

k-Armed Bandit



- Each **fixed base robot / service station** is a **bandit**
- Each **mobile robot** is an **arm**
- **We want to estimate the arm's probability of winning μ_a (task probability)**
- Every time we pull an arm we do an 'experiment'

Exploration vs. Exploitation

- We need to trade off **exploration** (gathering data about arm payoffs) and **exploitation** (making decisions based on data already gathered)

- **Exploration:** Pull an arm we never pulled before
- **Exploitation:** Pull an arm $\boldsymbol{a}$ for which we currently have the highest estimate of $\boldsymbol{\mu_a}$

Stochastic k-Armed Bandit

The setting:

- Set of ***k*** choices (arms)
- Each choice ***a*** is tied to a probability distribution P_a with average reward/payoff μ_a (between [0, 1])
- We play the game for ***T*** rounds
- For each round ***t***:
  - **(1)** We pick some arm ***j***
  - **(2)** We win reward X_t drawn from P_j
    - Note reward is independent of previous draws
- **Our goal is to maximize $\sum_{t=1}^{T} X_t$**
- **We don't know μ_a!** But every time we pull some arm ***a*** we get to learn a bit about μ_a

Epsilon-Greedy Policy

- At each time step, the agent selects an action
- The agent follows the greedy strategy with probability 1 – epsilon
- The agent selects a random action with probability epsilon
- With Q-learning, the greedy strategy is the action a that maximises Q given S_{t+1}

```
For t=1:T
    Set $\varepsilon_t = O(1/t)$
    With prob. $\varepsilon_t$: Explore by picking an arm chosen uniformly at random
    With prob. $1 - \varepsilon_t$: Exploit by picking an arm with highest empirical mean payoff
```

Theorem [Auer et al. '02]

For suitable choice of ε_t it holds that $R_T = O(k \log T) \Rightarrow \frac{R_T}{T} = O\left(\frac{k \log T}{T}\right) \to 0$

$$Q_{new} = \alpha \underline{Q_{new}} + (1-\alpha) Q_{old}$$

Issues with Epsilon Greedy

- **“Not elegant”**: Algorithm explicitly distinguishes between exploration and exploitation
- Exploration makes **suboptimal choices** (since it picks any arm with equal likelihood)

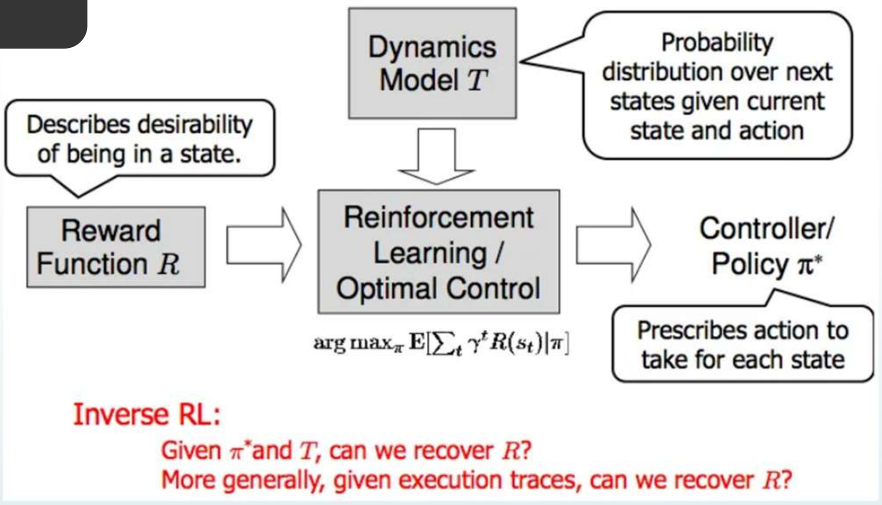
- **Idea:** When exploring/exploiting we need to **compare** arms
- Confidence interval based selection

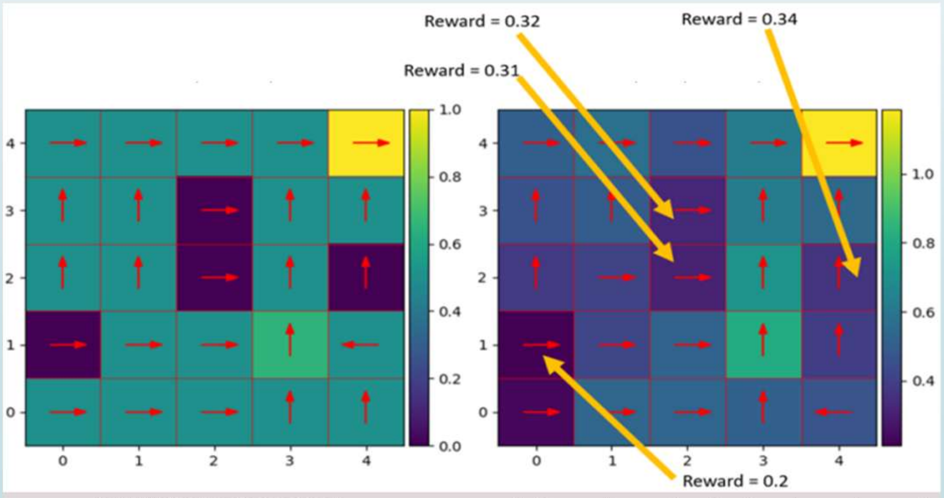
Deep Q-Networks (DQN)

- It is common to use a function approximator Q(s, a; θ) to approximate the action-value function in Q-learning
- Deep Q-Networks is Q-learning with a deep neural network function approximator called the Q-network
- Discrete and finite set of actions A
- Uses epsilon-greedy policy to select actions
- We want the neural network to learn a non-linear hierarchy of features or feature representation that gives accurate Q-value estimates
- The neural network has a separate output unit for each possible action, which gives the Q-value estimate for that action given the input state
- The neural network is trained using mini-batch stochastic gradient updates and experience replay

Inverse Reinforcement Learning

- In RL, it is necessary to manually tweak the rewards until the desired behaviour is observed.
- A better way is to learn the rewards by observing expert demonstrations (or policy).
- Thus IRL learns the underlying reward distribution using expert demonstrations/trajectories (human demonstrations).
- Expert policy π^* and transition probability matrix T is given from expert demonstrations (or can be obtained using supervised learning, etc.)
- IRL is then implemented using π^* and T to learn the reward distribution.
- For a robotic agent (autonomous navigation) or hand motion (for pointing task) it is easier to estimate the initial policy and transition matrix. Challenge is to estimate initial policy for eye gaze movement which has high uncertainty.
- Recently sampling-based approaches being used to obtain reward distribution using IRL for tasks with high uncertainty.

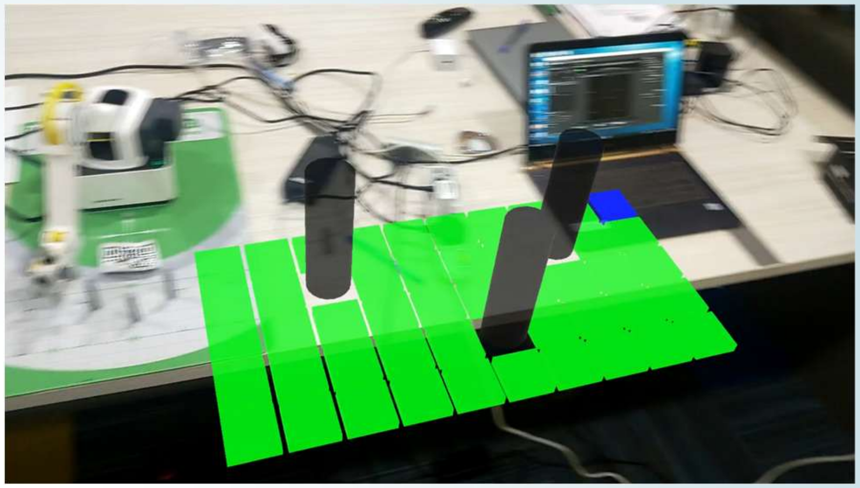

Dynamics Model T
Probability distribution over next states given current state and action
Describes desirability of being in a state.
Reward Function R
Reinforcement Learning / Optimal Control
$\arg\max_\pi \mathbf{E}[\sum_t \gamma^t R(s_t)|\pi]$
Controller/ Policy π^*
Prescribes action to take for each state
Inverse RL:
Given π^* and T, can we recover R?
More generally, given execution traces, can we recover R?

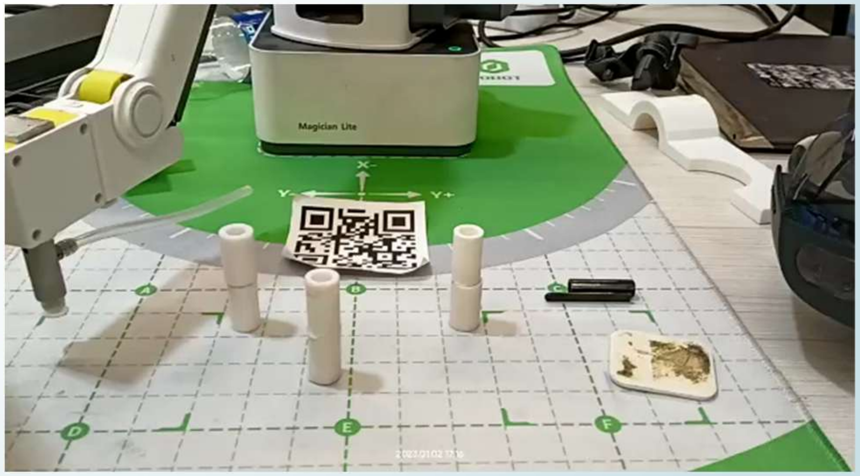


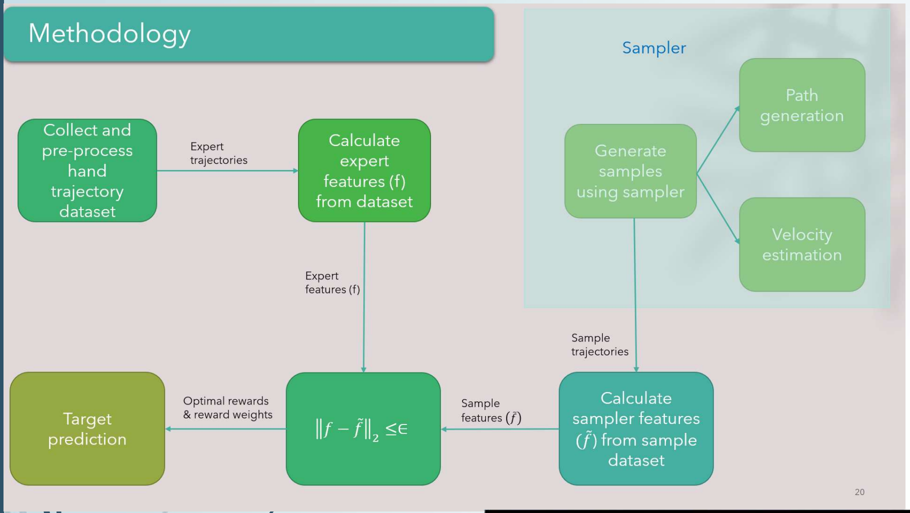
(a) Initial policy from expert (b) learned reward, policy from learned reward

Implementation of SMEIRL in 3D for Fixed Base Robot

- Robot workspace: 100x200x100mm
- 5x10x5 grid
- 150 Expert trajectories were collected, 120 for training, 30 for testing. Manually moving the robotic arm through hand, end-effector tracked by motion capture system.
- Velocity feature was used.
- SMEIRL was implemented.
- Given a start and target location, A-star was implemented using the learned rewards.
- Robot with no sensors, learns from human demonstrations and avoid obstacles.

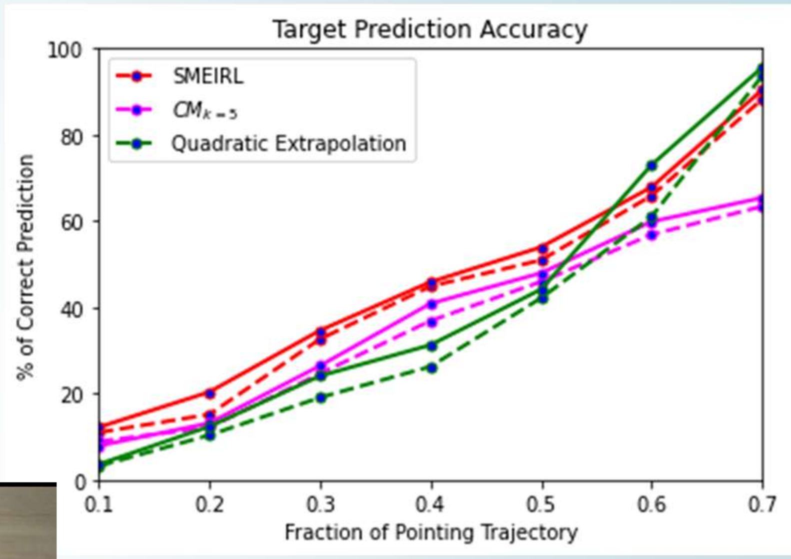


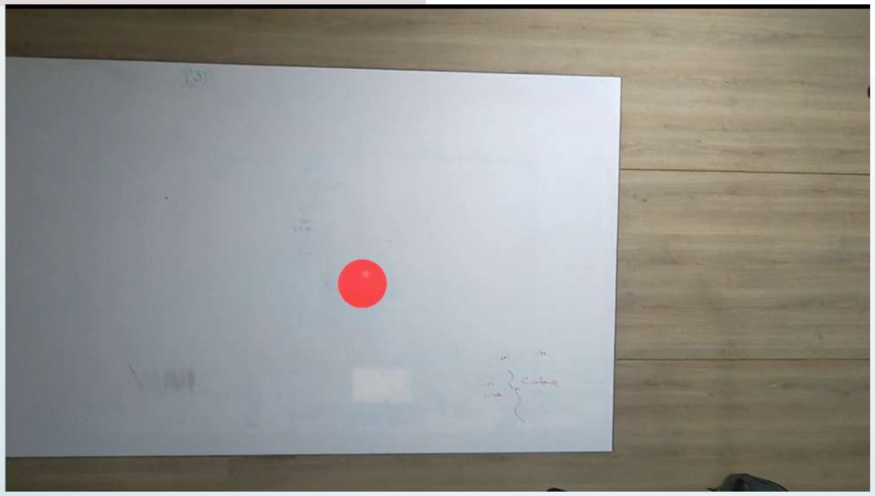




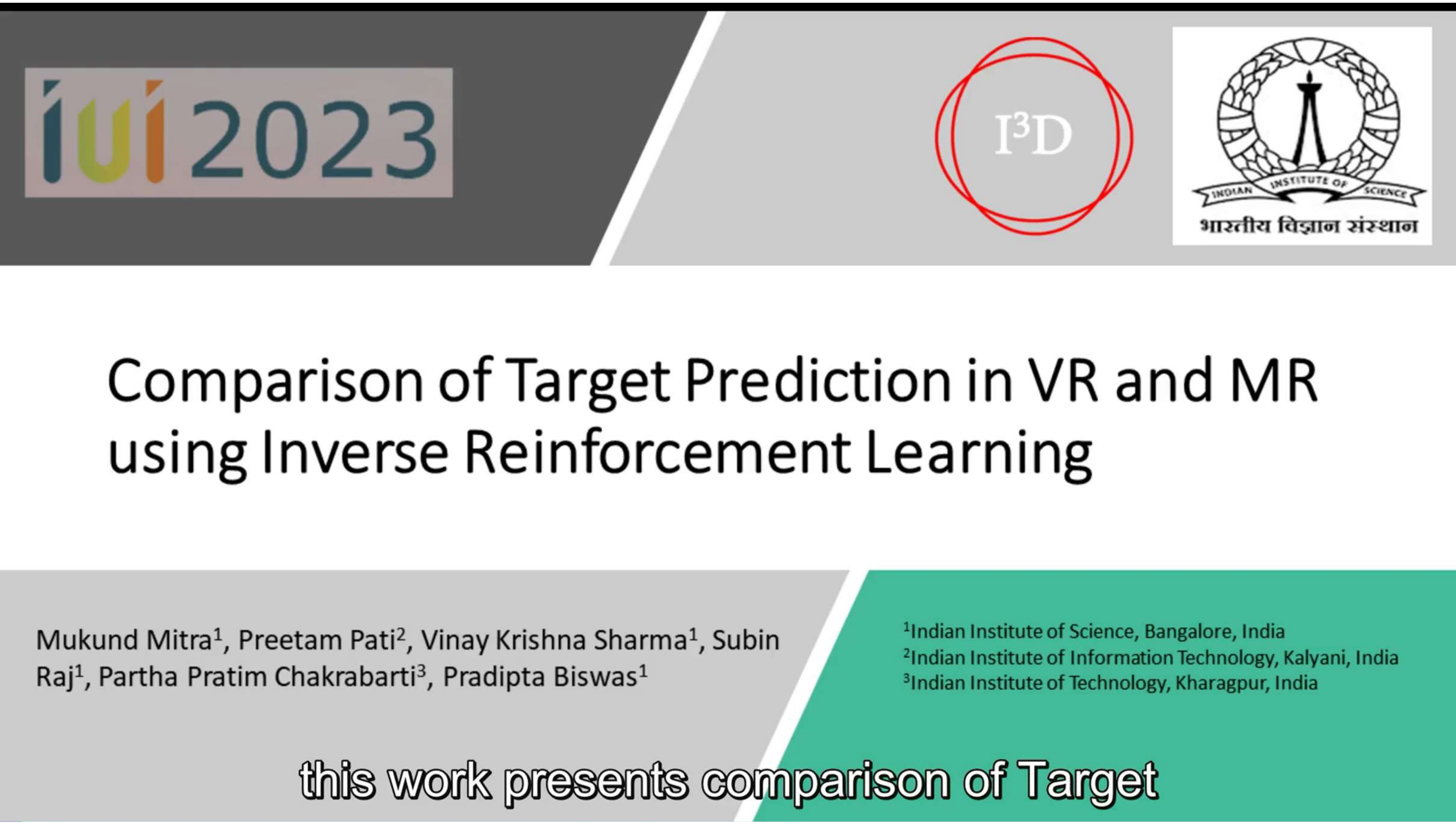
Methodology

Collect and pre-process hand trajectory dataset
Expert trajectories
Calculate expert features (f) from dataset
Sampler
Generate samples using sampler
Path generation
Velocity estimation
Expert features (f)
Sample trajectories
Target prediction
Optimal rewards & reward weights
$\|f - \tilde{f}\|_2 \leq \epsilon$
Sample features ($\tilde{f}$)
Calculate sampler features ($\tilde{f}$) from sample dataset
20





Hand Movement Prediction



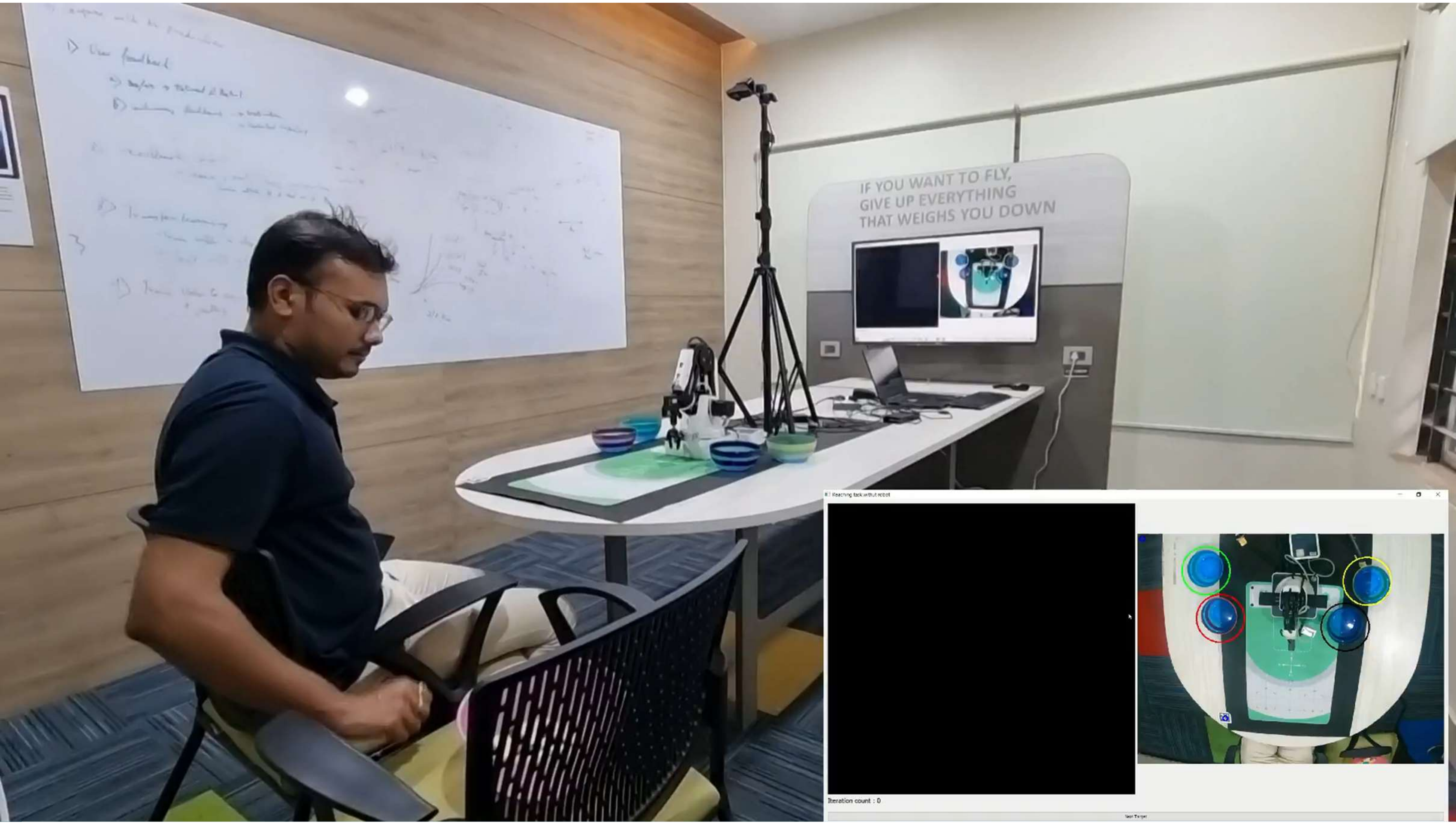
iui 2023

Comparison of Target Prediction in VR and MR using Inverse Reinforcement Learning

Mukund Mitra[1], Preetam Pati[2], Vinay Krishna Sharma[1], Subin Raj[1], Partha Pratim Chakrabarti[3], Pradipta Biswas[1]

[1]Indian Institute of Science, Bangalore, India
[2]Indian Institute of Information Technology, Kalyani, India
[3]Indian Institute of Technology, Kharagpur, India

this work presents comparison of Target


IF YOU WANT TO FLY,
GIVE UP EVERYTHING
THAT WEIGHS YOU DOWN
Reaching task without robot
Iteration count : 0
Next Target

Summary

- Introduction to Reinforcement Learning
- Different Types
  - Passive vs Active
  - Model based vs Model Free
- Concept of Temporal Difference and Q Learning
- Exploration vs Exploitation – Multi Arm Bandit Problem
- Introducing advanced topics – Deep Q-Network and Inverse Reinforcement Learning
- Demonstrations of Multi Arm Bandit in Warehouse Simulation and IRL for Trajectory Prediction