Lecture 10
ROB-GY 7863 / CSCI-GA 3033 7863: Planning, Learning, and Control for Space Robotics

Benjamin Riviere

November 10, 2025

Logistics

- Midterm feedback reports:
  - Thank you for filling these out! I'll do my best to incorporate the feedback.
- Project 2 Deadlines:
  - Final presentations: December 8th
- 4 lectures (including this one) left! Reinforcement Learning

Space Culture: StarCloud I

- StarCloud (partner with NVIDIA) builds data centers for AI in space.
  - For power, 24/7 high intensity solar power
  - For cooling, radiate into darkness of space
  - Dawn dusk sun synchronous orbit
- White Paper
- Alphabet announced Project Suncatcher
- SpaceX announced Starling v3 datacenters

Space Culture: Robot Exploration 3.0 I

- Robotics Exploration 3.0
- Hiro Ono: Research Technologist and the Group Supervisor of the Robotic Surface Mobility Group. PI of EELs robot.
- Robotics Exploration
  - 1.0:
    - Era: 1958-1968.
    - Goal: explore the Earth's moon.
    - Failed vs Successful missions: 12 vs 8.
    - Vibe: Fast paced trial and error.
    - Will this approach work for Mars? Probably not because launch windows are sparse.
  - 2.0:
    - Era: 1992-2020.
    - Goal: explore Mars.
    - Failed vs Successful Missions: 3 vs 11.
    - Vibe: incremental complexity improvement over series of missions.

Space Culture: Robot Exploration 3.0 II

- Will this approach work for outer solar system like Saturn's Moon Enceladeus? Probably not because (i) orbital reconnaissance of subsurface ocean is not possible (ii) cruise time is 10 years (iii) scaling to many planets.

- 3.0:
  - Era: 2025-.
  - Goal: explore outer solar system
  - Vibe: one-shot exploration. Whereas mars rover behaviors are fixed before launch, we want robotics that can adapt on the planet based on its own experiences. Start simple/safe, explore to more complexity.
  - Example: Exobiology Extant Live Surveyor (EELS (start at 5:35) Robot.

Recap

- Local Minima
- A*, RRT, RRT*
- Software tools
- Example: Rover path planning
- Break
- Seminar on MCTS for Robotics

Agenda

- MDP and Value
- Value Iteration and Policy Iteration
- Break
- Overview of Reinforcement Learning
- Value-Based Reinforcement Learning
- Problem Extensions: POMDPs, MGs
- Reference for today (Murphy, 2024)

Markov Decision Process (MDP) I

- A Markov Decision Process is a tuple: $\text{MDP} = \langle S, A, T, R, \gamma \rangle$ where
  - S is the state space
  - A is the action space
  - T is the transition probability function: $T(s, a, s') = p(s'|s, a)$
  - R is the reward function
  - γ is the discount factor
- **Example:** LQR is an MDP with a continuous state/action space, linear dynamics model and quadratic cost function. Exercise: specify the MDP for the LQR problem.
- **Example:** Gridworld is an MDP with a discrete state/action space, Markov kernel dynamics model, and goal-conditioned reward. Exercise: specify the MDP for Gridworld.

Value Functions I

- A policy π is a (stochastic) function from state to action, $\pi : S \to \Delta A$.
- The value of a policy is the expected discounted return of the policy, and its exact definition depends on notion of time:
  - Finite horizon $K < \infty$, $\gamma = 1$:

$$V^{\pi}(s) = \mathbb{E}_{s_{k+1} \sim T(.|s_k, a_k), a_k \sim \pi(.|s_k)} \left[\sum_{k=1}^{K} R(s_k, a_k) | s_0 = s \right] \quad (1)$$

  - Discounted infinite horizon $K \to \infty$, $\gamma < 1$:

$$V^{\pi}(s) = \mathbb{E}_{s_{k+1} \sim T(.|s_k, a_k), a_k \sim \pi(.|s_k)} \left[\sum_{k=1}^{\infty} \gamma^k R(s_k, a_k) | s_0 = s \right] \quad (2)$$

Value Functions II

- Average cost $K \to \infty$, $\gamma = 1$:

$$V^{\pi}(s) = \lim_{K \to \infty} \frac{1}{K} \mathbb{E}_{s_{k+1} \sim T(.|s_k, a_k), a_k \sim \pi(.|s_k)} \left[\sum_{k=1}^{K} R(s_k, a_k) | s_0 = s \right] \tag{3}$$

- Value functions have a recursive structure:

$$V^{\pi}(s) = \mathbb{E}_{s_{k+1} \sim T(.|s_k, a_k), a_k \sim \pi(.|s_k)} \left[R(s_0, a_0) + \gamma V^{\pi}(s_1) | s_0 = s \right] \tag{4}$$

  If you use recursive structure in your algorithm, you are doing **dynamic programming**. We will have examples later.

- The action-value function is another important function to introduce:

$$Q^{\pi}(s, a) = R(s, a) + \gamma \mathbb{E}_{s' \sim T(.|s,a)} V^{\pi}(s') \tag{5}$$

Value Functions III

- The solution of an MDP is the optimal value and policy pair:

$$\pi^*(s) = \arg\max_{\pi} V^{\pi}(s) \tag{6}$$

$$V^*(s) = \max_{\pi} V^{\pi}(s) = V^{\pi^*}(s) \tag{7}$$

- The principal of optimality says "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision." - Bellman 1957.
- Applying this principal, we write the optimal value function in the **Bellman Equation**:

$$V^*(s) = \max_{a} R(s, a) + \gamma \mathbb{E}_{s'} V^*(s') \tag{8}$$

Value Iteration I

- How to solve an MDP and get V^*?
- Assumptions: Discounted ($\gamma < 1$), finite state space ($|S| < \infty$).
- Algorithm:
  - Initialize $V(s) = 0 \; \forall s \in S$
  - Iterate:

$$V_{k+1} = \mathcal{T} V_k \tag{9}$$

$$(\mathcal{T} V)(s) = \max_{a \in A} R(s, a) + \gamma \mathbb{E}_{s' \sim T(.|s,a)} V(s') \tag{10}$$

where $\mathcal{T}$ is called the **Bellman Value Operator**.

- **TODO:** 1d grid example

Value Iteration Analysis I

- Goal is to show that:

$$\|V^* - V_k\|_\infty \leq \gamma^k \|V^* - V_0\|_\infty \tag{11}$$

- Strategy: Show that V^* is a fixed point of $\mathcal{T}$ and that $\mathcal{T}$ is contracting:

$$\|\mathcal{T}V - \mathcal{T}W\|_\infty = \gamma \|V - W\|_\infty \tag{12}$$

- By definition, V^* is fixed point of $\mathcal{T}$.

Value Iteration Analysis II

- It remains to show the operator $\mathcal{T}$ is contracting. For all V, W, and states $s \in S$:

$$\mathcal{T}V(s) - \mathcal{T}W(s) = \left[\max_a R(s,a) + \gamma \mathbb{E}_{s'} V(s')\right] \tag{13}$$

$$- \left[\max_a R(s,a) + \gamma \mathbb{E}_{s'} W(s')\right] \tag{14}$$

$$\leq \gamma \max_a \sum_{s'} T(s'|s,a)(V(s') - W(s')) \tag{15}$$

$$\leq \gamma \max_a \sum_{s'} T(s'|s,a)\|V - W\|_\infty \tag{16}$$

$$= \gamma \|V - W\|_\infty \tag{17}$$

$$\tag{18}$$

where we use identity:
$\max_x f(x) - \max_x g(x) \leq \max_x f(x) - g(x)$.

Value Iteration Analysis III

- Because this relation holds for all states s, it also holds at the $\max_{s \in S}(\mathcal{T}V)(s) - (\mathcal{T}W)(s) = \|\mathcal{T}V - \mathcal{T}W\|_\infty$, so we arrive at desired result.

Notes on Value Iteration

- Computational Complexity: Each iteration has $O(|A||S|^2)$ computations and, to reach some specified error, we require $k = \log_\gamma \frac{\epsilon}{\epsilon_0}$ iterations.
- Value iteration requires probabilities $T(s'|s, a)$, which is considered "access to the model" and "model-based".

Policy Iteration I

- Goal: compute an optimal policy π^* and V^* for a discounted, finite MDP ($\gamma < 1$, $|S| < \infty$).
- Algorithm:
  - Policy Evaluation, given π, solve for V^π:

$$V^\pi(s) = \sum_a \pi(a|s) \left[R(s,a) + \gamma \mathbb{E}_{s' \sim T(\cdot|s,a)} V^\pi(s') \right] \tag{19}$$

  - Policy Improvement:

$$\mathcal{T}^\pi(s) = \arg\max_{a \in A} \left[R(s,a) + \gamma \, \mathbb{E}_{s' \sim T(\cdot|s,a)} V(s') \right] \tag{20}$$

Motivating (and defining) Value-Based Reinforcement Learning

- Value and Policy iteration require "model" $T(., s, a)$.
- Instead, we might want to approximate this iteration with only samples $a \sim \pi_b$ and $s' \sim T(.|s, a)$. In the ML community, this is called "model free", even though we sample from the model. This is the ML distinction between "learning" and "planning".
- The fact that we use only samples in learning algorithms, motivates the distinction between two policies:
  - a behavioral policy generates data π^b
  - a target policy is what is learned π^t
- In off policy algorithms, $\pi^b \neq \pi^t$
- In on-policy algorithms, $\pi^b = \pi^t$
- We will see why this distinction is important later.

Break I

- Break

Overview of Reinforcement Learning I

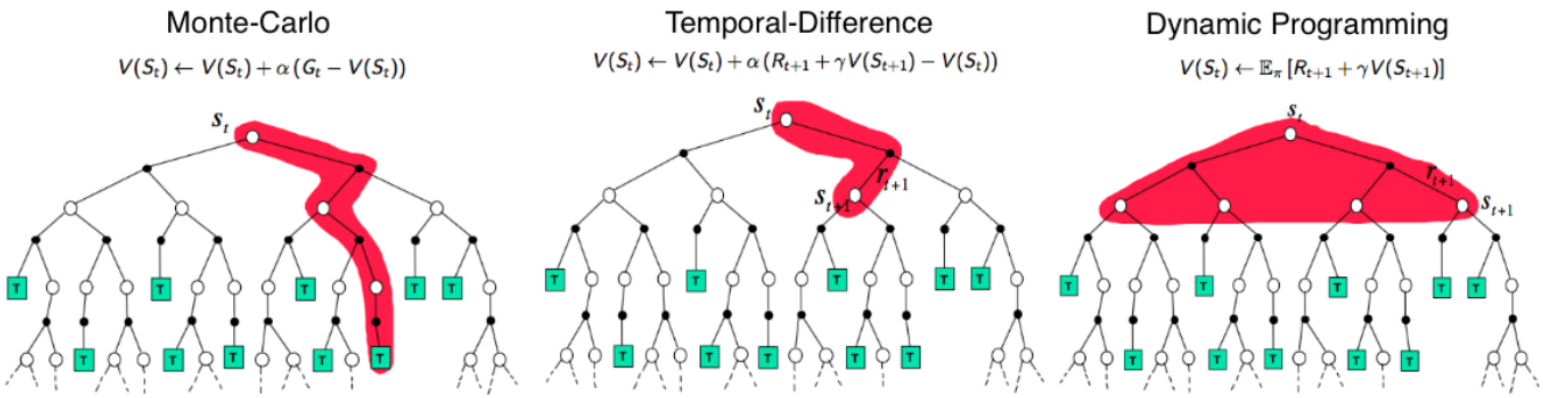
Approach	Method	Functions learned	On/Off	Section
Value-based	SARSA	$Q(s,a)$	On	Section 2.4
Value-based	Q-learning	$Q(s,a)$	Off	Section 2.5
Policy-based	REINFORCE	$\pi(a\|s)$	On	Section 3.1.3
Policy-based	A2C	$\pi(a\|s)$, $V(s)$	On	Section 3.2.1
Policy-based	TRPO/PPO	$\pi(a\|s)$, $\text{Adv}(s,a)$	On	Section 3.3.3
Policy-based	DDPG	$a = \pi(s)$, $Q(s,a)$	Off	Section 3.2.6.2
Policy-based	Soft actor-critic	$\pi(a\|s)$, $Q(s,a)$	Off	Section 3.6.8
Model-based	MBRL	$p(s'\|s,a)$	Off	Chapter 4

Overview of Reinforcement Learning II

URL	Language	Comments
Stoix	Jax	Mini-library with many methods (including MBRL)
PureJaxRL	Jax	Single files with DQN; PPO, DPO
JaxRL	Jax	Single files with AWAC, DDPG, SAC, SAC+REDQ
Stable Baselines Jax	Jax	Library with DQN, CrossQ, TQC; PPO, DDPG, TD3, SAC
Jax Baselines	Jax	Library with many methods
Rejax	Jax	Library with DDQN, PPO, (discrete) SAC, DDPG
Dopamine	Jax/TF	Library with many methods
Rlax	Jax	Library of RL utility functions (used by Acme)
Acme	Jax/TF	Library with many methods (uses rlax)
CleanRL	PyTorch	Single files with many methods
Stable Baselines 3	PyTorch	Library with DQN; A2C, PPO, DDPG, TD3, SAC, HER
TianShou	PyTorch	Library with many methods (including offline RL)

Table 1.3: Some open source RL software.

Value-Based Reinforcement Learning I



To approximate policy evaluation $\mathcal{T}^\pi$, we can do:

- Monte Carlo: Unbiased, high variance, does not use dynamic programming

$$G_t = r_t + \gamma r_{t+1} + \dots \tag{21}$$

$$V_{k+1}(s_t) = V_k(s_t) + \alpha(G_t - V_k(s_t)) \tag{22}$$

Value-Based Reinforcement Learning II

- Temporal Difference: Biased, low variance, uses dynamic programming

$$V_{k+1}(s) = V_k(s) + \alpha(R + \gamma V_k(s') - V_k(s)) \qquad (23)$$

SARSA Algorithm I

- SARSA (**XX**) is on-policy TD learning. Its called SARSA because we use data of type (s, a, r, s', a') .
- Define ϵ-greedy policy:

$$\pi_k^{\epsilon}(s) = \begin{cases} \arg\max_a Q_k(s,a) & \text{w.p } 1-\epsilon \\ \sim \mathbb{U}(A) & \text{w.p } \epsilon \end{cases} \tag{24}$$

- Algorithm:
  - Initialize: $Q_0 = 0$, $\pi_0(s) = \arg\max_a Q_k(s,a)$
  - Iterate: If done, reset state and action (s, a). Then:

$$\begin{aligned} a' &\sim \pi_k^{\epsilon}(s) \\ s' &\sim T(.|s,a') \\ Q_{k+1}(s,a) &= Q_k(s,a) + \eta\left[r + \gamma Q_k(s',a') - Q_k(s,a)\right] \\ s &\leftarrow s' \\ a &\leftarrow a' \end{aligned}$$

Q Learning I

- Q Learning (**XX**) is off-policy TD learning.
- Algorithm:
  - Initialize: $Q_0 = 0$
  - Iterate: If done, reset state s. Then:

$$a \sim \pi_k^{\epsilon}(s) \tag{25}$$

$$Q_{k+1}(s,a) = Q_k(s,a) + \eta \left[r + \gamma \max_{a'} Q_k(s',a') - Q_k(s,a) \right] \tag{26}$$

$$s \leftarrow s' \tag{27}$$

On-Policy vs Off-Policy RL I

- On-Policy (SARSA): Learn about the policy that is actually executed. Updates reflect the value of the exploratory policy.
- Off-Policy (Q Learning): Learn about a policy different from the one that generates data. Updates reflect value of a greedy policy, while behaving ϵ-greedy.
- Exploration affects:
  - On-policy: the value backups themselves.
  - Off-policy: only data collection, not the update target.
- Key consequence:
  - SARSA learns a safe policy under exploration.
  - Q-learning learns the optimal greedy policy even from exploratory samples.
- Exercise: How does Q-learning and SARSA learn cliff walking?

SARSA vs Q-Learning: Bellman Operator View I

- Both methods are 1-step temporal-difference algorithms, but differ in their **backup operators**.
- SARSA approximates Policy Iteration. It uses data to simultaneously estimate the Bellman Expectation Operator for policy π:

$$(\mathcal{T}^{\pi} Q)(s,a) = R(s,a) + \gamma \mathbb{E}_{s' \sim T(\cdot|s,a), a' \sim \pi(\cdot|s')} Q(s',a').$$

- Q Learning approximates Value Iteration. IT uses data to estimate the Bellman Optimality Operator:

$$(\mathcal{T}^{*} Q)(s,a) = R(s,a) + \gamma \mathbb{E}_{s' \sim T(\cdot|s,a)} \max_{a'} Q(s',a').$$

- You can tell by looking at their backup information.

Q Learning with Function Approximation

- Tabular methods will not scale because $|A|$ and $|S|$ get big.
- Parameterize Q with parameters w, $Q_w(s, a)$ (e.g. feed forward neural network).
- Q learning with Function Approximation:
- Not perfect, we will discuss limitations next time.

Algorithm 2: Q learning with function approximation and replay buffers

Initialize environment state $\boldsymbol{s}$, network parameters $\boldsymbol{w}_0$, replay buffer $\mathcal{D} = \emptyset$, discount factor γ, step size η, policy $\pi_0(a|s) = \epsilon \text{Unif}(a) + (1-\epsilon)\delta(a = \text{argmax}_a\, Q_{\boldsymbol{w}_0}(s, a))$
for *iteration* $k = 0, 1, 2, \ldots$ **do**
  for *environment step* $s = 0, 1, \ldots, S-1$ **do**
    Sample action: $a \sim \pi_k(a|s)$
    Interact with environment: $(s', r) = \text{env.step}(a)$
    Update buffer: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s, a, s', r)\}$
  $\boldsymbol{w}_{k,0} \leftarrow \boldsymbol{w}_k$
  for *gradient step* $g = 0, 1, \ldots, G-1$ **do**
    Sample batch: $B \subset \mathcal{D}$
    Compute error: $\mathcal{L}(B, \boldsymbol{w}_{k,g}) = \frac{1}{|B|} \sum_{(s,a,r,s') \in B} \left[Q_{\boldsymbol{w}_{k,g}}(s, a) - (r + \gamma \max_{a'} Q_{\boldsymbol{w}_k}(s', a')) \right]^2$
    Update parameters: $\boldsymbol{w}_{k,g} \leftarrow \boldsymbol{w}_{k,g} - \eta \nabla_{\boldsymbol{w}_{k,g}} \mathcal{L}(B, \boldsymbol{w}_{k,g})$
  $\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_{k,G}$

Partially Observable MDPs (POMDP) I

- Sometime we do not have access to the state and we only have access to measurements or observations from a sensor with known likelihood.
- A POMDP is an MDP with two additional components:
  - O observation space
  - Z likelihood function: $Z(s, o) = p(o|s)$
- Introduce history $h \in H$:

$$h_k = (o_1, a_1, \ldots, o_k, a_k) = (o_{1:k}, a_{1:k}) \tag{28}$$

- Same policy/value setup, except history replaces the state:

$$\pi : H \to \Delta A \tag{29}$$

$$V^{\pi}(h) = \mathbb{E}_{s_{k+1} \sim T(.|s_k, a_k), a_k \sim \pi(.|h_k), o_k \sim Z(.|s_k)} \left[\sum_{k=1}^{K} R(s_k, a_k) | h_0 = h \right] \tag{30}$$

Partially Observable MDPs (POMDP) II

- An alternative and equivalent formulation is to define the belief state (exactly the same object in the Bayes filter and Kalman filter, recall Lecture 7)

$$b_k(s_k) = p(s_k|h_k) \tag{31}$$

$$= \eta \underbrace{p(o_k|s_k)}_{Z} \int \underbrace{p(s_k|s_{k-1}, a_k)}_{T} \underbrace{p(s_{k-1}|h_{k-1})}_{b_{k-1}} ds_{k-1} \tag{32}$$

$$= \text{BayesFilter}(b_{k-1}, a_k, o_k) \tag{33}$$

Partially Observable MDPs (POMDP) III

- From any POMDP, we can construct a Belief-State MDP:

$$\text{BMDP} = \langle \mathcal{B}, A, T^b, R^b, \gamma \rangle \tag{34}$$

$$T^b(b', b, a) = \delta(b' = \int_o \text{BayesFiter}(b, a, o)) \tag{35}$$

$$R^b(b, a) = \int_s b(s) R(s, a) \tag{36}$$

- Belief-based policy and value:

$$\pi : \mathcal{B} \rightarrow \Delta U \tag{37}$$

$$V^\pi(b) = \mathbb{E}_{b_{k+1} \sim T^b(.|b_k, a_k), a_k \sim \pi(.|b_k)} \left[\sum_{k=1}^{K} \gamma^k R^b(b_k, a_k) | b_0 = b \right] \tag{38}$$

Partially Observable MDPs (POMDP) IV

- **Example:** LQG is an POMDP with a continuous state/action space, Gaussian kernel for dynamics and likelihood and prior, and quadratic cost function. Exercise: specify the POMDP for the LQG problem.
- **Example:** RockSample is an POMDP with a discrete observation/state/action space, Exercise: specify the POMDP for the RockSample problem.

Markov Games I

- Multi-agent MDP extension:

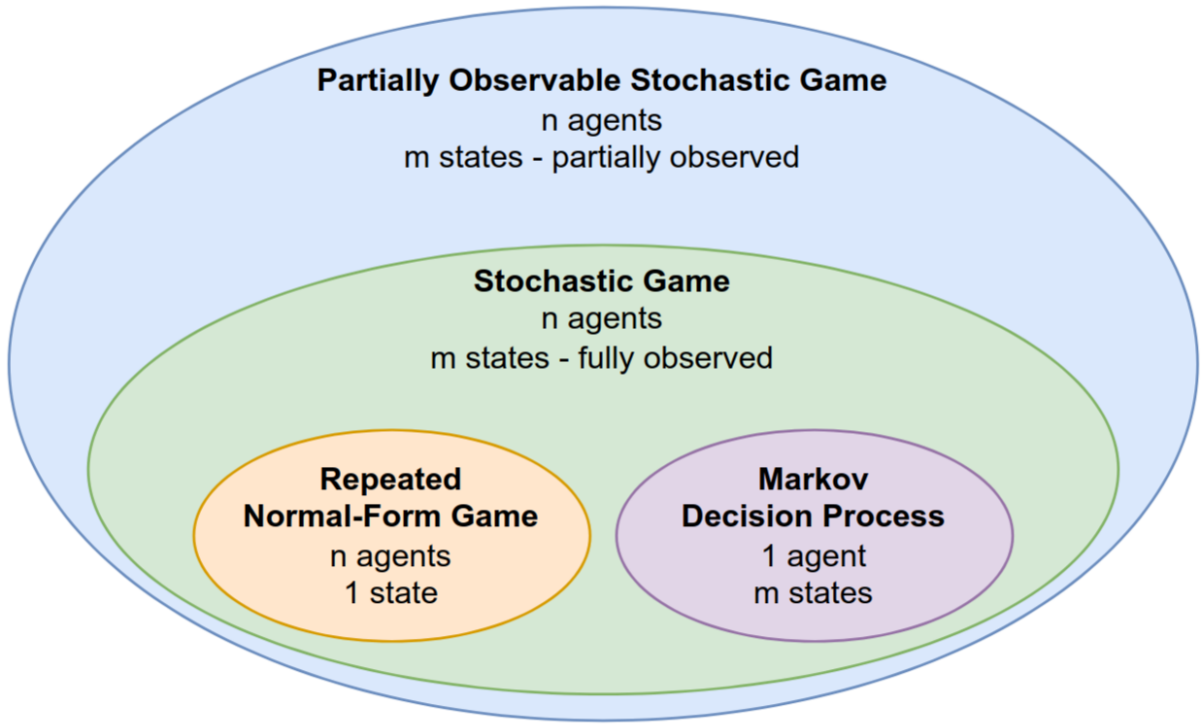
$$\text{MG} = \langle S, \{A^i\}_{i=1}^N, T, \{R^i\}_{i=1}^N, \gamma \rangle \tag{39}$$

- The agents each have their own actions and rewards and share a common state and transition function.
- The set of policies $\pi = \{\pi^i\}_{i=1}^N$ is at a **Nash Equillibrium** if no agent i can improve its expected returns by changing its policy π^i, assuming the other agents policies remain fixed:

$$\forall i, \pi^i, \quad V^i(\pi^i, \pi^{-i}) \leq V^i(\pi) \tag{40}$$

  where π and π^{-i} are fixed policies.
- In general, this is a challenging optimization problem and in practice, people try to avoid game theoretic settings. It is often easier to treat a team of robots as a single robot and use an MDP formulation.

Markov Games II



Markov Games III

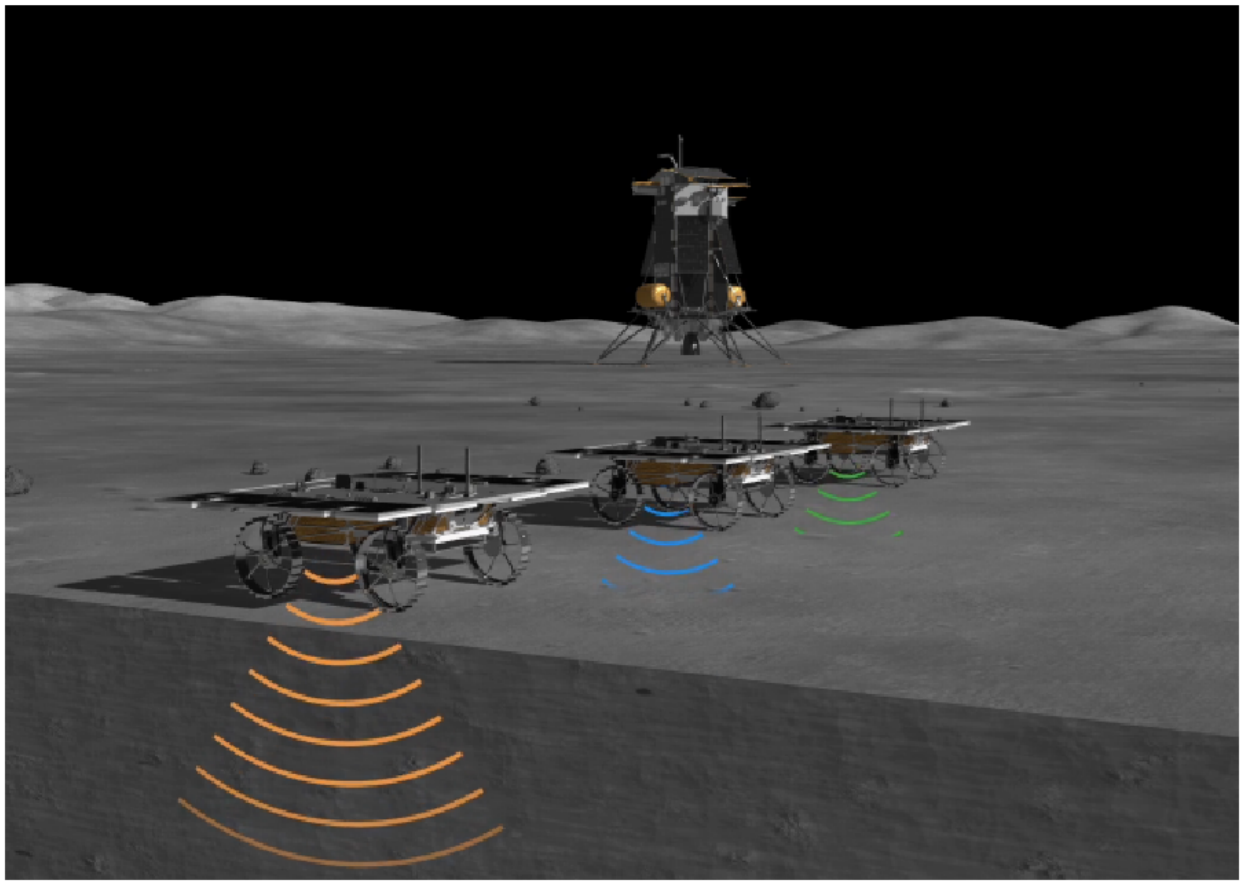


Figure: Example: CADRE mission

 Murphy, Kevin (2024). “Reinforcement learning: an overview”. In: arXiv preprint arXiv:2412.05265 (cit. on p. 7).