# Anytime Incremental $\rho$POMDP Planning in Continuous Spaces

Presented by: Ron Benchetrit
Advisor: Vadim Indelman

Technion – Israel Institute of Technology

28 April 2025

# Outline

# Autonomous Agents

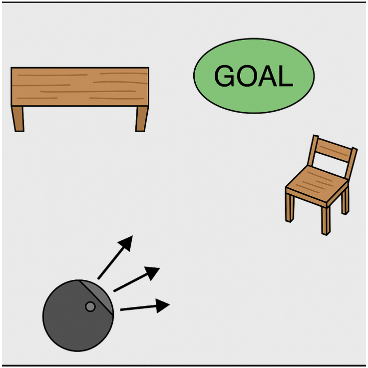
- Autonomous agents appear in a wide range of domains:


# Autonomous Agents

- Autonomous agents must:
  - **Perceive** the world through noisy, partial observations
  - **Act** under uncertainty in outcomes and dynamics
  - **Plan** to achieve long-term objectives
- This talk focuses on **sequential decision-making under uncertainty**.
- Specifically, planning in the **Partially Observable Markov Decision Process (POMDP)** and its extension, the $\rho$**POMDP**.

# MDP – Example

- Imagine a Roomba-like robot operating in a room.
- Its goal is to navigate to a designated target area.
- The robot acts in the environment by actuating its motors.
- However, due to motor errors, slippage, etc., the outcome of an action may not be as expected.
- We want the robot to reach the goal quickly and safely, avoiding obstacles along the way.



# MDP – Definition

A Markov Decision Process (MDP) is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$:

- **Definition:** $\mathcal{S}$ is the state space.

# MDP – Definition

A Markov Decision Process (MDP) is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$:

- **Definition:** $\mathcal{S}$ is the state space.
- *Example:* All possible robot positions and headings in the room.

# MDP – Definition

A Markov Decision Process (MDP) is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$:

- **Definition:** $\mathcal{S}$ is the state space.
- *Example:* All possible robot positions and headings in the room.
- **Definition:** $\mathcal{A}$ is the action space.

# MDP – Definition

A Markov Decision Process (MDP) is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$:

- **Definition:** $\mathcal{S}$ is the state space.
- *Example:* All possible robot positions and headings in the room.
- **Definition:** $\mathcal{A}$ is the action space.
- *Example:* `move forward`, `turn left`, `turn right`.

# MDP – Definition

A Markov Decision Process (MDP) is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$:

- **Definition:** $\mathcal{S}$ is the state space.
- *Example:* All possible robot positions and headings in the room.
- **Definition:** $\mathcal{A}$ is the action space.
- *Example:* `move forward`, `turn left`, `turn right`.
- **Definition:** $\mathcal{T}(s' \mid a, s)$ is the transition model — the probability of reaching $s'$ after taking action $a$ in state $s$.

# MDP – Definition

A Markov Decision Process (MDP) is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$:

- **Definition:** $\mathcal{S}$ is the state space.
- *Example:* All possible robot positions and headings in the room.
- **Definition:** $\mathcal{A}$ is the action space.
- *Example:* `move forward`, `turn left`, `turn right`.
- **Definition:** $\mathcal{T}(s' \mid a, s)$ is the transition model — the probability of reaching $s'$ after taking action $a$ in state $s$.
- *Example:* A linear-Gaussian motion model that accounts for motor noise and slippage.

# MDP – Definition

A Markov Decision Process (MDP) is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$:

- **Definition:** $\mathcal{S}$ is the state space.
- *Example:* All possible robot positions and headings in the room.
- **Definition:** $\mathcal{A}$ is the action space.
- *Example:* `move forward`, `turn left`, `turn right`.
- **Definition:** $\mathcal{T}(s' \mid a, s)$ is the transition model — the probability of reaching $s'$ after taking action $a$ in state $s$.
- *Example:* A linear-Gaussian motion model that accounts for motor noise and slippage.
- **Definition:** $\mathcal{R}(s, a)$ is the reward function.

# MDP – Definition

A Markov Decision Process (MDP) is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$:

- **Definition:** $\mathcal{S}$ is the state space.
- *Example:* All possible robot positions and headings in the room.

- **Definition:** $\mathcal{A}$ is the action space.
- *Example:* `move forward`, `turn left`, `turn right`.

- **Definition:** $\mathcal{T}(s' \mid a, s)$ is the transition model — the probability of reaching $s'$ after taking action $a$ in state $s$.
- *Example:* A linear-Gaussian motion model that accounts for motor noise and slippage.

- **Definition:** $\mathcal{R}(s, a)$ is the reward function.
- *Example:* $+100$ for reaching the goal, $-100$ for hitting an obstacle, and $-1$ for each time step.

# MDP – Objective

The agent aims to choose actions that **maximize expected cumulative reward**:

$$\mathbb{E}\left[\sum_{t=0}^{\infty} r(s_t, a_t)\right]$$

A solution to an MDP is a **policy** $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maps states to actions.

- **Value function:**

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} r(s_t, a_t) \mid s_0 = s, a_t = \pi(s_t)\right]$$

- **Action-value function:**

$$Q^{\pi}(s, a) = \mathbb{E}_{s' \sim T(\cdot|s,a)}\left[r(s, a) + V^{\pi}(s')\right], \quad V^{\pi}(s) = Q^{\pi}(s, \pi(s))$$

# POMDP – Example

- In the MDP setting, the robot always knew its exact position.
- Now, its position is **unknown** — it only starts with a rough guess.
- The robot uses a **laser rangefinder** to measure distances to walls.
- These measurements are **noisy and ambiguous** — multiple locations can produce similar readings.
- To navigate effectively, the robot must estimate its position using both its actions and sensor data.

# POMDP – Definition

A Partially Observable Markov Decision Process (POMDP) extends an MDP to handle partial observability. It is defined by the tuple:

$$(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O}, \mathcal{Z}, b_0)$$

- $\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$, and $\mathcal{R}$ are the same as in MDP.

# POMDP – Definition

A Partially Observable Markov Decision Process (POMDP) extends an MDP to handle partial observability. It is defined by the tuple:

$$(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O}, \mathcal{Z}, b_0)$$

- $\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$, and $\mathcal{R}$ are the same as in MDP.
- **Definition:** $\mathcal{O}$ is the observation space.

# POMDP – Definition

A Partially Observable Markov Decision Process (POMDP) extends an MDP to handle partial observability. It is defined by the tuple:

$$(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O}, \mathcal{Z}, b_0)$$

- $\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$, and $\mathcal{R}$ are the same as in MDP.
- **Definition:** $\mathcal{O}$ is the observation space.
- *Example:* All possible laser rangefinder readings the robot can receive.

# POMDP – Definition

A Partially Observable Markov Decision Process (POMDP) extends an MDP to handle partial observability. It is defined by the tuple:

$$(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O}, \mathcal{Z}, b_0)$$

- $\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$, and $\mathcal{R}$ are the same as in MDP.
- **Definition:** $\mathcal{O}$ is the observation space.
- *Example:* All possible laser rangefinder readings the robot can receive.
- **Definition:** $\mathcal{Z}(o \mid s)$ is the observation model — the probability of observing $o$ in state $s$.

# POMDP – Definition

A Partially Observable Markov Decision Process (POMDP) extends an MDP to handle partial observability. It is defined by the tuple:

$$(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O}, \mathcal{Z}, b_0)$$

- $\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$, and $\mathcal{R}$ are the same as in MDP.
- **Definition:** $\mathcal{O}$ is the observation space.
- *Example:* All possible laser rangefinder readings the robot can receive.
- **Definition:** $\mathcal{Z}(o \mid s)$ is the observation model — the probability of observing $o$ in state $s$.
- *Example:* Noisy distance measurements that may be consistent with several positions.

# POMDP – Definition

A Partially Observable Markov Decision Process (POMDP) extends an MDP to handle partial observability. It is defined by the tuple:

$$(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O}, \mathcal{Z}, b_0)$$

- $\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$, and $\mathcal{R}$ are the same as in MDP.
- **Definition:** $\mathcal{O}$ is the observation space.
- *Example:* All possible laser rangefinder readings the robot can receive.
- **Definition:** $\mathcal{Z}(o \mid s)$ is the observation model — the probability of observing $o$ in state $s$.
- *Example:* Noisy distance measurements that may be consistent with several positions.
- **Definition:** $b_0$ is the initial belief - a distribution over states.

# POMDP – Definition

A Partially Observable Markov Decision Process (POMDP) extends an MDP to handle partial observability. It is defined by the tuple:

$$(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O}, \mathcal{Z}, b_0)$$

- $\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$, and $\mathcal{R}$ are the same as in MDP.
- **Definition:** $\mathcal{O}$ is the observation space.
- *Example:* All possible laser rangefinder readings the robot can receive.
- **Definition:** $\mathcal{Z}(o \mid s)$ is the observation model — the probability of observing $o$ in state $s$.
- *Example:* Noisy distance measurements that may be consistent with several positions.
- **Definition:** $b_0$ is the initial belief - a distribution over states.
- *Example:* The robot's initial guess about its position, e.g., a uniform distribution over the room.

# Belief – Definition

- In MDPs, the agent chooses actions based on the fully observable state.
- In POMDPs, the state is hidden, so the agent must act based on what it knows so far.
- To handle this, the agent maintains a **belief** $b_t$: a probability distribution over possible states at time $t$.
- The belief is updated over time using the agent's history of actions and observations:

$$h_t = (b_0, a_0, o_1, \ldots, a_{t-1}, o_t)$$

- The belief at time $t$ is defined as:

$$b_t(s) = \mathbb{P}(s \mid h_t)$$

# POMDP – Objective

As in MDPs, the agent aims to **maximize expected cumulative reward**:

$$\mathbb{E}\left[\sum_{t=0}^{\infty} r(s_t, a_t)\right]$$

Since the state is hidden, actions are chosen based on the agent's **belief**. A solution is a **policy** $\pi : \mathcal{B} \to \mathcal{A}$ mapping beliefs to actions.

- **Value function:**

$$V^\pi(b) = \mathbb{E}\left[\sum_{t=0}^{\infty} r(s_t, a_t) \,\middle|\, b_0 = b,\ a_t = \pi(b_t)\right]$$

- **Action-value function:**

$$Q^\pi(b, a) = \mathbb{E}_{s \sim b,\, s' \sim T(\cdot|s,a),\, o \sim O(\cdot|s')}\left[r(s, a) + V^\pi(b')\right]$$

where $b'$ is the updated belief after taking $a$ and observing $o$.

# Belief – Update

The belief is updated recursively using the Bayes filter:

$$b_t(s_t) = \eta \cdot \mathcal{Z}(o_t \mid s_t) \int_S \mathcal{T}(s_t \mid s_{t-1}, a_{t-1})\, b_{t-1}(s_{t-1})\, ds_{t-1}$$

where $\eta$ is a normalization constant.

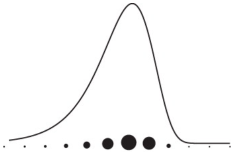
Special cases of the Bayes filter include:

- **Kalman Filter** — for linear-Gaussian systems.
- **Particle Filter** — for non-linear, non-Gaussian systems.

# Particle Filter

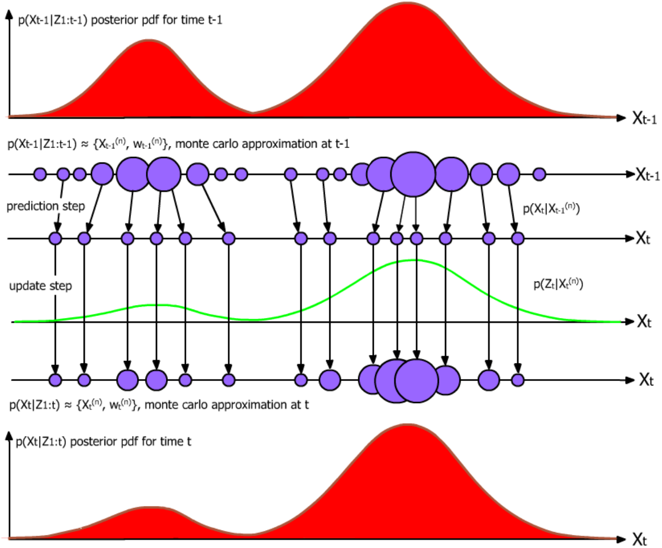
- A particle filter approximates the belief using a **set of weighted samples** (particles).
- Each particle represents a possible state and has an associated weight.

$$b(s) \approx \sum_{i=1}^{N} w_i \cdot \delta(s - s_i)$$



- This representation is highly flexible and well-suited for capturing complex, multimodal beliefs.
- As the number of particles $N$ **increases**, the approximation becomes more **accurate**.

# Particle Filter – Update



# $\rho$POMDP – Example

- Now, suppose the robot's task is to **localize itself** in the room.
- There is no designated goal location — the objective is to **reduce uncertainty**.
- Examples: active localization, informative path planning, active learning, and active SLAM.
- Such tasks cannot be naturally expressed using the standard $R(s, a)$.
- Instead, we define a **belief-dependent reward function** $\rho(b, a)$ — for example, negative entropy.

# $\rho$POMDP – Definition

A $\rho$POMDP is defined by the tuple:

$$(\mathcal{S}, \mathcal{A}, \mathcal{T}, \cancel{\mathcal{R}}, \mathcal{O}, \mathcal{Z}, b_0, \rho)$$

- $\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$, $\mathcal{O}$, $\mathcal{Z}$, and $b_0$ are defined as in a standard POMDP.

# $\rho$POMDP – Definition

A $\rho$POMDP is defined by the tuple:

$$(\mathcal{S}, \mathcal{A}, \mathcal{T}, \cancel{\mathcal{R}}, \mathcal{O}, \mathcal{Z}, b_0, \rho)$$

- $\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$, $\mathcal{O}$, $\mathcal{Z}$, and $b_0$ are defined as in a standard POMDP.
- **Definition:** $\rho$ is a belief dependent reward, i.e., $\rho(b, a)$.

# $\rho$POMDP – Definition

A $\rho$POMDP is defined by the tuple:

$$(\mathcal{S}, \mathcal{A}, \mathcal{T}, \cancel{\mathcal{R}}, \mathcal{O}, \mathcal{Z}, b_0, \rho)$$

- $\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$, $\mathcal{O}$, $\mathcal{Z}$, and $b_0$ are defined as in a standard POMDP.
- **Definition:** $\rho$ is a belief dependent reward, i.e., $\rho(b, a)$.
- *Example:* Information-theoretic rewards such as negative entropy or information gain.

# $\rho$POMDP – Definition

A $\rho$POMDP is defined by the tuple:

$$(\mathcal{S}, \mathcal{A}, \mathcal{T}, \cancel{\mathcal{R}}, \mathcal{O}, \mathcal{Z}, b_0, \rho)$$

- $\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$, $\mathcal{O}$, $\mathcal{Z}$, and $b_0$ are defined as in a standard POMDP.
- **Definition:** $\rho$ is a belief dependent reward, i.e., $\rho(b, a)$.
- *Example:* Information-theoretic rewards such as negative entropy or information gain.
- This formulation naturally captures tasks centered on **active information gathering**.

# $\rho$POMDP – Definition

A $\rho$POMDP is defined by the tuple:

$$(\mathcal{S}, \mathcal{A}, \mathcal{T}, \cancel{\mathcal{R}}, \mathcal{O}, \mathcal{Z}, b_0, \rho)$$

- $\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$, $\mathcal{O}$, $\mathcal{Z}$, and $b_0$ are defined as in a standard POMDP.
- **Definition:** $\rho$ is a belief dependent reward, i.e., $\rho(b, a)$.
- *Example:* Information-theoretic rewards such as negative entropy or information gain.
- This formulation naturally captures tasks centered on **active information gathering**.
- In practice, $\rho$ can combine both belief-dependent and state-dependent rewards:
  $$\rho(b, a) = \mathbb{E}_{s \sim b}[r(s, a)] + \mathcal{I}(b)$$

# $\rho$POMDP – Objective

In $\rho$POMDPs, the agent aims to **maximize expected cumulative belief-dependent reward**:

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \rho(b_t, a_t)\right]$$

Here, rewards depend directly on the **belief** and **action**. A solution is a **policy** $\pi : \mathcal{B} \to \mathcal{A}$ mapping beliefs to actions.

# Solving POMDPs – Computation Complexity

Solving POMDPs is a challenging task due to the following reasons:

- Curse of dimensionality
- Curse of history
- Continuous state space
- Continuous action space
- Continuous observation space

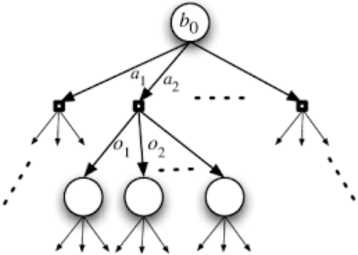
# Offline vs. Online Planning

- **Offline methods** aim to compute a policy $\pi : \mathcal{B} \rightarrow \mathcal{A}$ in advance.
  - Require solving for the entire belief space.
  - Computationally intractable for large or continuous domains.
  - Examples: value iteration, point-based solvers (PBVI).

- **Online methods** compute the next action at each time step, based on the current belief.
  - Avoid planning for unreachable beliefs.
  - More scalable in high-dimensional or continuous settings.
  - Examples: POMCP [6], DESPOT [7], POMCPOW [8].

# Online Tree Search

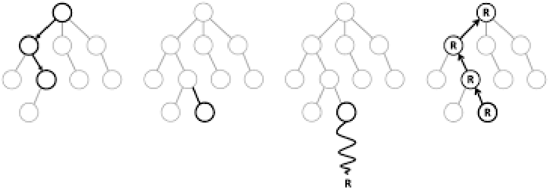
- Online methods plan by building a search tree rooted at the current belief — the **belief tree**.
- Nodes represent beliefs; edges correspond to actions and observations.
- The tree is expanded by simulating action-observation trajectories.
- The best action is selected based on estimated values at the root's children.
- The chosen action is executed in the real environment, an observation is received, the belief is updated, and the process repeats.



# Monte Carlo Tree Search

- Monte Carlo Tree Search (MCTS) is a widely used planning algorithm.
- It uses random trajectory simulations to estimate the value of actions from the current belief.
- MCTS builds the tree incrementally using four key steps:

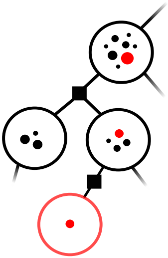
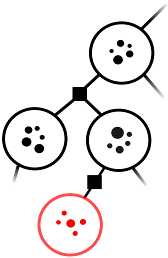
- **Selection:** Traverse the tree to select a promising node.
- **Expansion:** Add one or more child nodes.
- **Simulation:** Simulate a rollout from the new node to estimate return.
- **Backpropagation:** Propagate the return up the tree to update value estimates.




(a) Selection (b) Expansion (c) Simulation (d) Backpropagation

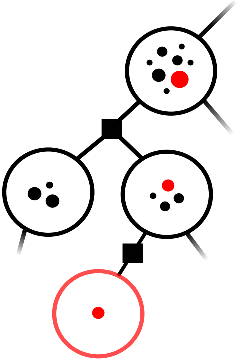
# State Simulators vs. Belief Simulators

- SOTA online POMDP solvers use a **particle-based** approach to represent the belief.
- These solvers can be broadly categorized as **state simulators** or **belief simulators**.

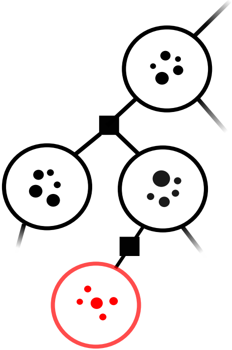
# State Simulators

- Simulate state trajectories directly.
- Each time a belief node is visited, a new particle is added.
- Particles accumulated across visits.
- Representation improves at frequently visited (and promising) nodes.
- Belief-dependent rewards are more challenging to compute.
- Examples: POMCP [6], POMCPOW [8], DESPOT [7], LABECOP [5].



# Belief Simulators

- Treat belief states as explicit nodes in a belief-MDP.
- Fixed set of particles created once upon expansion.
- Representation is static, wasting computation resources.
- Easy belief-dependent rewards computation.
- Examples: PFT-DPW [8], AdaOPS [10].



# Online $\rho$POMDP Solvers

| Setting | Belief Simulator | State Simulator |
|---|---|---|
| **Discrete** | $\rho$beliefUCT [9] | $\rho$POMCP [9] |
| **Continuous** | PFT-DPW [8]<br>IPFT [4]<br>AI-FSSS [1]<br>SITH-PFT [11] | |

# Online $\rho$POMDP Solvers

| Setting | Belief Simulator | State Simulator |
|---|---|---|
| **Discrete** | $\rho$beliefUCT [9] | $\rho$POMCP [9] |
| **Continuous** | PFT-DPW [8]<br>IPFT [4]<br>AI-FSSS [1]<br>SITH-PFT [11] | $\rho$POMCPOW (ours) |

# $\rho$POMCPOW – Challenges

- We aim to extend POMCPOW — a state-of-the-art state-simulator for POMDPs — to the $\rho$POMDP setting.
- This introduces several key challenges:
  - **Value estimation:** Running averages used in POMCPOW are incompatible with belief-dependent rewards.
  - **Belief representation:** Particle accumulation during visits does not ensure sufficient belief quality across the tree.
  - **Reward recomputation:** Belief updates require frequent and costly re-evaluation of the reward function.

In the next slides, we address each of these challenges in turn.

# Challenge: Value Estimation

- In POMCPOW, node values are estimated using a running average of sampled rewards:

$$\hat{Q}(h, a) = \frac{1}{N(ha)} \sum_{i=1}^{N(ha)} R_i$$

- This is well-suited for state-based rewards, where the expected reward depends only on the state.
- In $\rho$POMDPs, rewards are **non-linear functions of the belief**, which evolves as new particles are added.
- Averaging rewards across outdated beliefs leads to **biased and inconsistent** value estimates.

# Solution: Last-Value Update (LVU)

- To avoid mixing outdated rewards, we adopt the **Last-Value Update (LVU)** framework proposed by [9].
- In LVU, each node stores a value estimate based only on the most recent rewards from its children.

**Value:**

$$\hat{V}(h) = \frac{1}{N(h)} \left[ \text{Rollout}(h) + \sum_{a \in Ch(h)} N(ha)\hat{Q}(ha) \right]$$

**Action-value:**

$$\hat{Q}(ha) = \frac{1}{N(ha)} \sum_{o \in Ch(ha)} N(hao) \left[ \hat{\rho}(hao) + \gamma \hat{V}(hao) \right]$$

# Our Contribution: Incremental LVU

- Standard LVU updates require summing over all children: $O(n)$ time.
- This is costly in large trees, especially in continuous domains.
- We propose an **incremental update** that maintains correctness with just $O(1)$ computation.

**Incremental value update:**

$$\hat{V}(h) \leftarrow \hat{V}(h) + \frac{1}{N(h)}\Big[N(ha')\hat{Q}(ha') - (N(ha')-1)\hat{Q}^{\text{prev}}(ha') - \hat{V}(h)\Big]$$

**Incremental action-value update:**

$$\hat{Q}(ha) \leftarrow \hat{Q}(ha) + \frac{1}{N(ha)}\Big[N(hao')\big(\hat{\rho}(hao') + \gamma\hat{V}(hao')\big)$$
$$-(N(hao')-1)\big(\hat{\rho}^{\text{prev}}(hao') + \gamma\hat{V}^{\text{prev}}(hao')\big) - \hat{Q}(ha)\Big]$$

# Challenge: Belief Representation

- Beliefs are typically represented using a particle-based approximation.
- In state simulators, particles are added incrementally during node visitation.
- This improves belief representation at frequently visited nodes.
- However, less-visited nodes may remain poorly represented.
- For belief-dependent rewards, we need good belief quality **throughout the tree**, not just locally.

# Solution: Anytime Belief Refinement

- We ensure that belief quality improves **over time and throughout the tree**.
- Our approach provides a theoretical guarantee: under consistent selection strategies, **every node** is visited increasingly often.
- This enables belief refinement even in deep or rarely explored parts of the tree.
- Further details and formal results are available in the paper.

# Challenge: Reward Re-computation

- In $\rho$POMDPs, rewards depend on the belief — not just the state.
- As the belief evolves (e.g., when new particles are added), the reward must be updated accordingly.
- However, belief-dependent rewards are often:
  - **Non-linear** (e.g., entropy-based), making updates non-trivial.
  - **Expensive** to recompute from scratch.
- Major computational bottleneck in continuous domains.
- Want to update rewards **incrementally**, without full recomputation.
- We will focus in information-theoretic rewards.

# Solution: Incremental Shannon Entropy

- Shannon entropy is a common information-theoretic reward used to quantify belief uncertainty:

$$\hat{H}(b) = -\sum_{i=1}^{N} w_i \log w_i$$

- Recomputing it after every belief update is linear in the number of particles: $O(N)$.
- We derive an equivalent form that enables fast incremental updates:

$$\hat{H}(b) = -\frac{1}{\sum w_i} \sum w_i \log w_i + \log \sum w_i$$

- This formulation allows updating entropy in $O(1)$ time after adding a new particle — by reusing cached terms.

# Boers Entropy Estimator

- Shannon entropy is simple but not well-suited for particle-based beliefs — especially in continuous spaces.
- The Boers estimator [2] is specifically designed for such settings, capturing local density and the shape of the belief.
- It is defined as:

$$\hat{H}(b') = \log \sum_{i=1}^{N} \mathcal{Z}(o \mid s'_i)\, w'_i - \sum_{i=1}^{N} w'_i \log \mathcal{Z}(o \mid s'_i) - \sum_{i=1}^{N} w'_i \log \left[ \sum_{j=1}^{N} \mathcal{T}(s'_i \mid s_j, a)\, w_j \right]$$

- The final term — a nested sum over particles — is the computational bottleneck: $\mathcal{O}(N^2)$.
- This term can be updated incrementally and efficiently.

# Incremental Update for Boers Estimator

- Recall the final (expensive) term in the Boers estimator:

$$\sum_{i=1}^{N} w_i' \log \underbrace{\left[ \sum_{j=1}^{N} \mathcal{T}(s_i' \mid s_j, a) \cdot w_j \right]}_{c_i}$$

- Adding a new particle $s_{N+1}$ affects all $c_i$. We incrementally update them as:

$$\tilde{c}_i = \frac{\sum_{j=1}^{N} w_j}{\sum_{j=1}^{N+1} w_j} \cdot c_i + \mathcal{T}(s_i \mid s_{N+1}, a) \cdot \tilde{w}_{N+1}$$

- This reuses cached values and only requires computing one new transition column.
- Total cost drops from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$.

# Experimental Setup

- We implemented $\rho$POMCPOW in Julia using the `POMDPs.jl` framework [3].
- Source code: `github.com/ronbenc/rhoPOMCPOW`
- Compared against state-of-the-art solvers on two benchmark domains:
    - **Continuous 2D Light-Dark**
    - **Active Localization**
- We also evaluate the benefit of **incremental reward computation**.

# Benchmark Domains

- Both domains are set in a 2D continuous environment with:
  - Uncertain initial belief over the agent's state
  - Beacons that provide noisy observations (more accurate when closer)
  - Stochastic dynamics with 8 movement directions (unit circle) and a stay action that terminates the episode
- **Light-Dark:**
  - Task: navigate to a goal region from an uncertain start.
  - Sparse rewards — large bonus on success, heavy penalty on failure.
  - *Information gain* is used as reward shaping to encourage localization.
- **Active Localization:**
  - Task: actively reduce uncertainty about position.
  - Reward: pure *information gain*.
  - Environment includes obstacles that penalize collisions.
- *Results are averaged over 1000 runs with different seeds.* We report mean return $\pm$ standard error.

# Results – Continuous 2D Light-Dark

- **Task:** Navigate to a goal region using beacon-based observations.
- $\rho$POMCPOW outperforms due to its effective belief representation.
- Despite the added cost, incorporating *information gain* boosts planning effectiveness in $\rho$POMCPOW.

| **Algorithm** | **0.1s** | **0.2s** | **1.0s** |
|---|---|---|---|
| $\rho$POMCPOW$^{\dagger}$ | **22.3** $\pm$ 1.2 | **25.9** $\pm$ 1.1 | **26.2** $\pm$ 1.1 |
| POMCPOW | 17.2 $\pm$ 1.4 | 17.5 $\pm$ 1.4 | 18.5 $\pm$ 0.9 |
| IPFT$^{\dagger}$ | -2.3 $\pm$ 1.8 | 6.4 $\pm$ 1.7 | 17.2 $\pm$ 1.2 |
| PFT-DPW$^{\dagger}$ | 5.3 $\pm$ 1.6 | 13.4 $\pm$ 1.4 | 20.5 $\pm$ 1.0 |

$^{\dagger}$ Uses information gain as reward shaping.

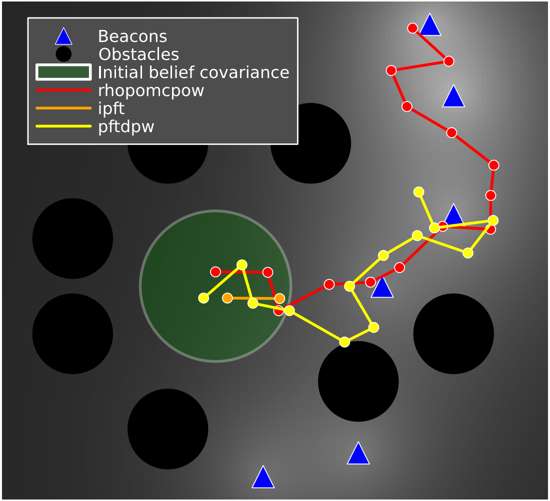
# Results – Active Localization

- **Task:** Actively localize using beacon observations while avoiding obstacles.
- $\rho$POMCPOW **steadily improves** and outperforms as the planning time increases — thanks to its anytime refinement of beliefs.
- IPFT underperforms, likely due to its expensive particle reinvigoration mechanism.

| **Algorithm** | **0.1s** | **0.2s** | **1.0s** |
|---|---|---|---|
| $\rho$POMCPOW | 29.0 $\pm$ 0.5 | **38.1** $\pm$ 0.7 | **45.9** $\pm$ 0.8 |
| IPFT | 27.1 $\pm$ 0.4 | 27.7 $\pm$ 0.4 | 27.0 $\pm$ 0.4 |
| PFT-DPW | **36.7** $\pm$ 0.7 | 37.6 $\pm$ 0.7 | 38.7 $\pm$ 0.7 |

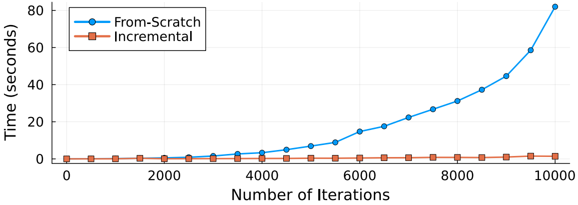
# Example Trajectory – Active Localization

- The agent actively reduces uncertainty about its position while avoiding obstacles.



# Effect of Incremental Reward Computation

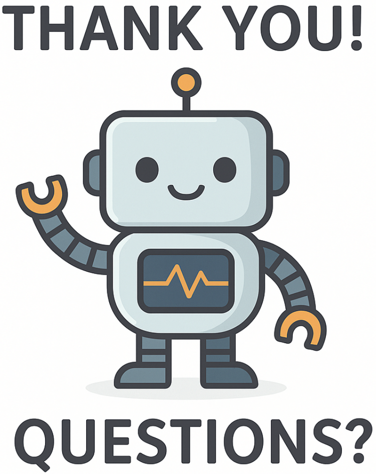
- We compare $\rho$POMCPOW **with and without** incremental reward updates on the Light-Dark domain.
- Both variants use the same random seed to ensure identical search tree expansion.
- **Result:** Full recomputation becomes increasingly expensive as planning progresses.
- **Incremental updates significantly reduce planning time** — essential for $\rho$POMCPOW to be applicable in real-time scenarios.



# Conclusions

- We introduced $\rho$**POMCPOW** — an anytime, online tree search algorithm for solving $\rho$POMDPs in continuous spaces.
- Our contributions include:
  - **Incremental LVU** for accurate and efficient value estimation.
  - **Anytime belief refinement** with theoretical guarantees on belief improvement.
  - **Incremental reward computation**, enabling scalable use of belief-dependent rewards.
- Empirical results demonstrate improved solution quality and planning efficiency over state-of-the-art methods.
- **Limitations:** convergence remains an open question; belief-dependent reward computation is still costly.

# Questions?



M. Barenboim and V. Indelman.
Adaptive information belief space planning.
In *the 31st International Joint Conference on Artificial Intelligence and the 25th European Conference on Artificial Intelligence (IJCAI-ECAI)*, July 2022.

Y. Boers, H. Driessen, A. Bagchi, and P. Mandal.
Particle filter based entropy.
In *2010 13th International Conference on Information Fusion*, pages 1–8, 2010.

Maxim Egorov, Zachary N Sunberg, Edward Balaban, Tim A Wheeler, Jayesh K Gupta, and Mykel J Kochenderfer.
Pomdps. jl: A framework for sequential decision making under uncertainty.
*The Journal of Machine Learning Research*, 18(1):831–835, 2017.

Johannes Fischer and Omer Sahin Tas.
Information particle filter tree: An online algorithm for pomdps with belief-based rewards on continuous domains.

In *Intl. Conf. on Machine Learning (ICML)*, Vienna, Austria, 2020.

Marcus Hoerger and Hanna Kurniawati.
An on-line pomdp solver for continuous observation spaces.
In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 7643–7649. IEEE, 2021.

David Silver and Joel Veness.
Monte-carlo planning in large pomdps.
In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2164–2172, 2010.

Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee.
Despot: Online pomdp planning with regularization.
In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 13, pages 1772–1780, 2013.

Zachary Sunberg and Mykel Kochenderfer.
Online algorithms for pomdps with continuous state, action, and observation spaces.

In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, 2018.

Vincent Thomas, Geremy Hutin, and Olivier Buffet.
Monte carlo information-oriented planning.
*arXiv preprint arXiv:2103.11345*, 2021.

Chenyang Wu, Guoyu Yang, Zongzhang Zhang, Yang Yu, Dong Li, Wulong Liu, and Jianye Hao.
Adaptive online packing-guided search for pomdps.
*Advances in Neural Information Processing Systems*, 34:28419–28430, 2021.

Andrey Zhitnikov, Ori Sztyglic, and Vadim Indelman.
No compromise in solution quality: Speeding up belief-dependent continuous pomdps via adaptive multilevel simplification.
*Intl. J. of Robotics Research*, 2024.