Sequential Decision Making and Reinforcement Learning

(SDMRL)

Decision-Making Under Partial Information

Sarah Keren

The Taub Faculty of Computer Science
Technion - Israel Institute of Technology

Acknowledgments

- David Sliver's course on RL: *https://www.deepmind.com/learning-resources/introduction-to-reinforcement-learning-with-david-silver*
- Slides by Malte Helmert, Carmel Domshlak, Erez Karpas and Alexander Shleyfman.

Planning With Partial Observability

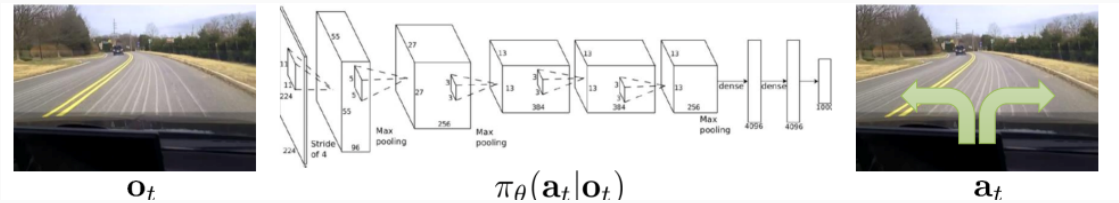
Accounting for Partial Information

Accounting for Partial Information

- Can we use a **Markov Decision Process**(MDP) $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ to account for partially observable environments ?

Accounting for Partial Information

Sometimes, yes.



Sometimes, an MDP is not enough.

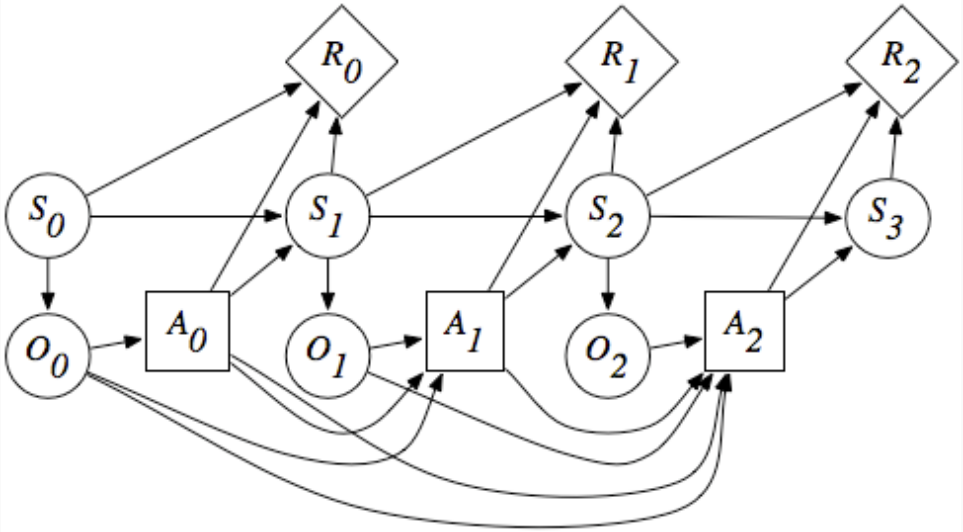
Remidner: Partially Observable Markov Decision Process (POMDP)

A **Partially Observable Markov Decision Process**(POMDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \Omega, \mathcal{O}, \beta_0 \rangle$ where

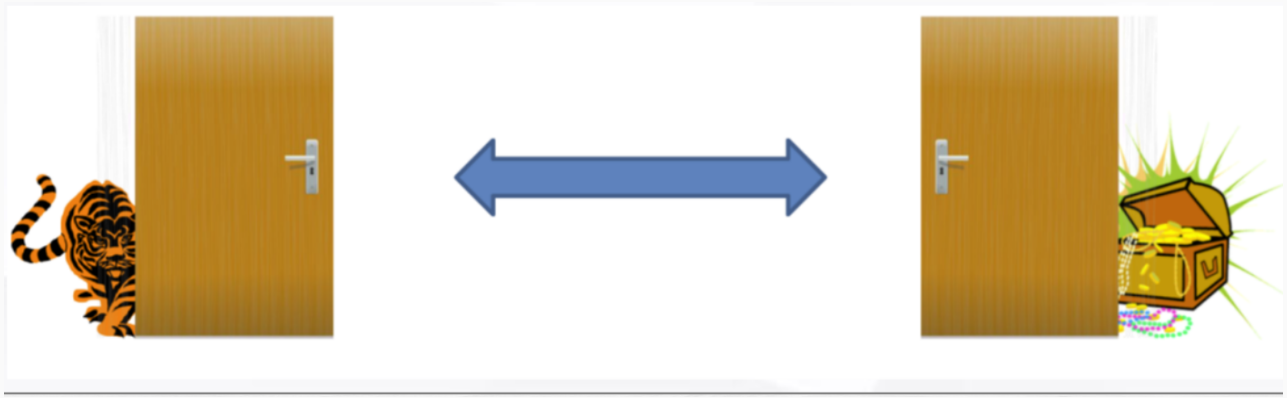
- $\mathcal{S}$, $\mathcal{A}$, $\mathcal{P}$, $\mathcal{R}$ and γ are as for an MDP.
- Ω is a set of observations (observation tokens),
- $\mathcal{O}$ is a sensor function specifying the conditional observation probabilities $\mathcal{O}^o_{s,a} = \mathcal{P}[O_{t+1} = o | S_t = s, A_t = a]$ of receiving observation token $o \in \Omega$ in state s after applying action a [1].
- β_0 the initial belief: a probability distribution over the states such that $\beta_0(s)$ stands for the probability of s being the true initial state.

[1] alternatively: $\mathcal{O}^o_s = \mathcal{P}[o_t = o | S_t = s]$

POMDP- graphical form

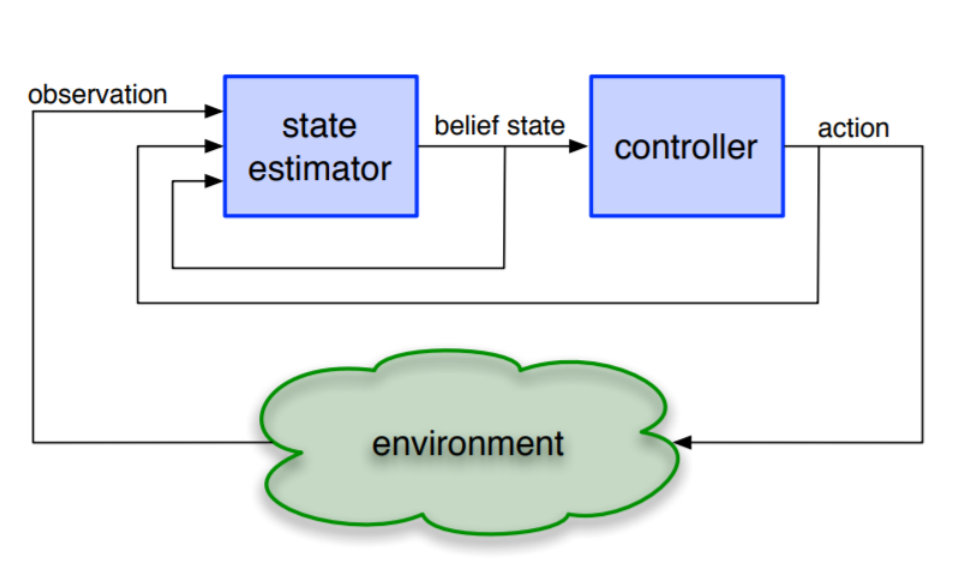


POMDP example



Planning in Belief Space

- A **belief** is a probability distribution over the possible world states such that $b(s)$ stands for the probability that s is the true world state.
- In partially observable domains, we may have a **sensor model / state estimator** represented as a mapping function from what is observed to the actual world state.

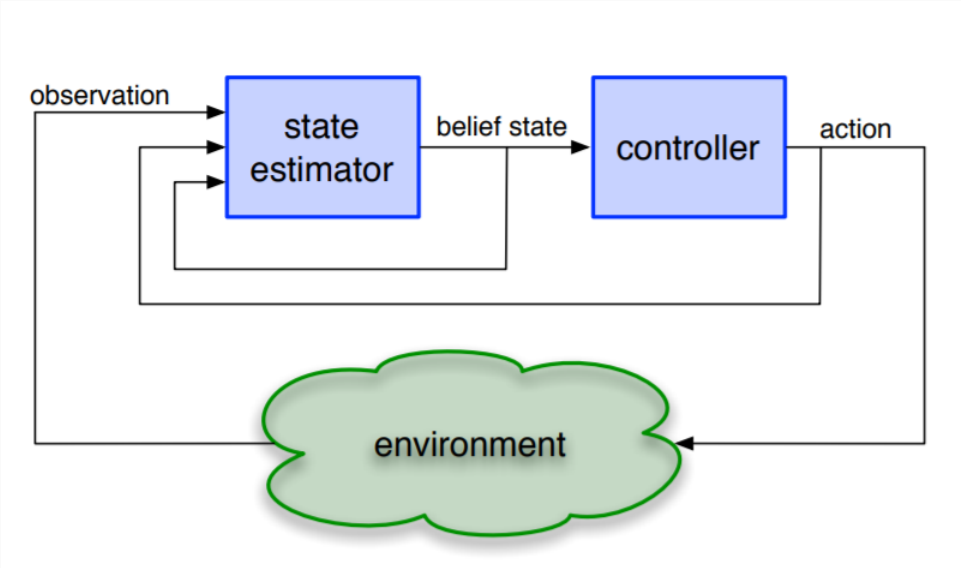


From Kaelbling, L. P., and T. Lozano-Perez. "Integrated Task and Motion Planning in Belief Space" 2013 *https://dspace.mit.edu/bitstream/handle/1721.1/87038/Kaelbling_Integrated%20task.pdf?sequence=1&isAllowed=y*

Planning in Belief Space

Two key challenges when planning in belief space:

- **Belief tracking** - what is the state of the world ?
- **Policy computation** - what is the best action to perform ?

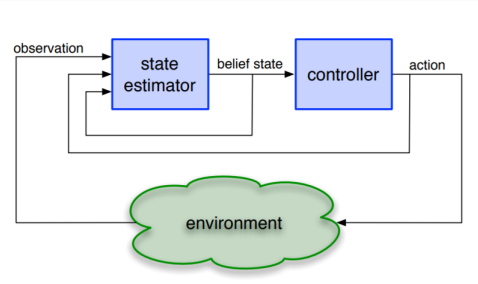


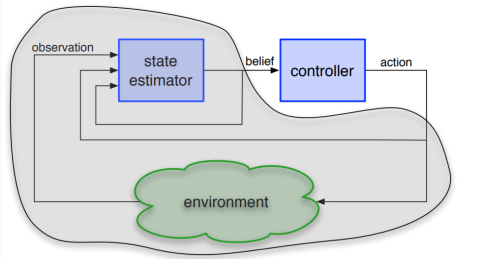
Pineau, Nicholas and Thrun. "A hierarchical approach to POMDP planning and execution." 2001. https://www.cs.mcgill.ca/ jpineau/files/jpineau-icml01.pdf

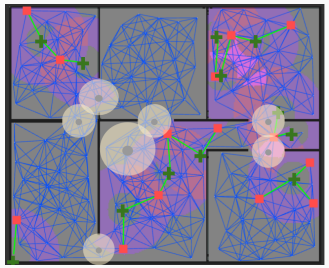
Planning in Belief Space: Solution Approaches

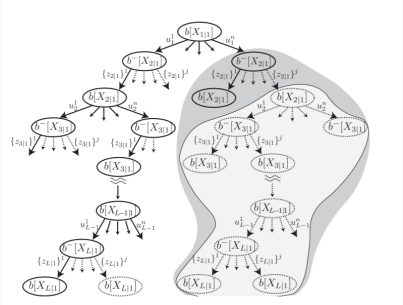
Combinations of different approaches:

- Planning in a **belief MDP**, an MDP with beliefs as states
- Sampling / discretization
- Approximations / relaxations









See work by Vadim Indelman from the Technion, e.g.,

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8793548

Belief Update

Belief Update

When receiving an observation o, the agent updates its current belief β using its **belief update function** $\tau : \mathcal{B} \times \Omega \times \mathcal{S} \mapsto \mathcal{B}$ which maps belief $\beta \in \mathcal{B}$ to the new belief.

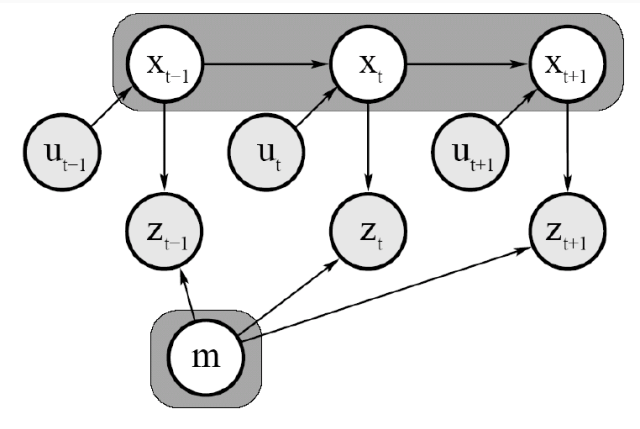
Commonly, a **Bayesian filter** is used:

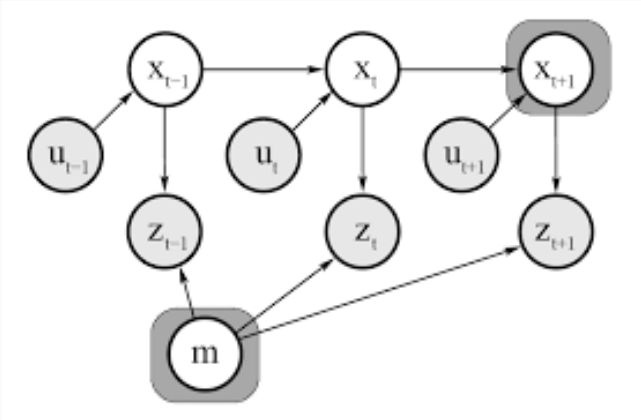
$$\beta^{o,a} = \frac{\mathcal{P}(o|s,a)\,\beta(s)}{\int_{s' \in \mathcal{S}} \mathcal{P}(o|s',a)\,\beta(s')\,ds'} \tag{1}$$

where $\mathcal{P}$ is the sensor function and $\beta(s)$ is the estimated probability that s is the actual world state
Discrete version:

$$\beta^{o,a} = \frac{\mathcal{P}(o|s,a)\,\beta(s)}{\Sigma_{p' \in \mathcal{P}} \mathcal{P}(o|s',a)\,\beta(s')} \tag{2}$$

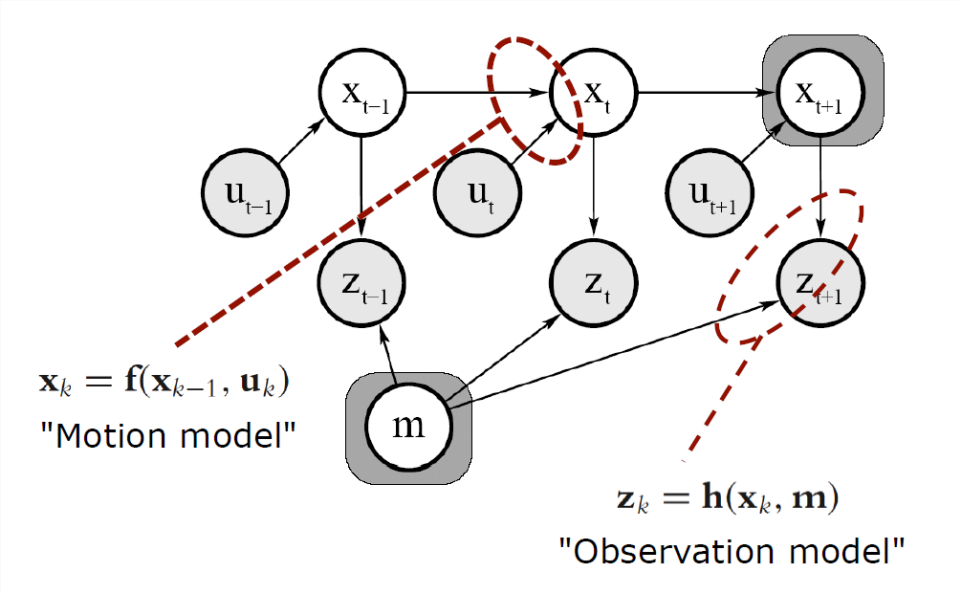
Probabilistic sensors and dynamics - graphical model





Circles represent variables. Arrows represent dependencies (influences).

Probabilistic sensors and dynamics - graphical model



Motion model: how does the agent move ?

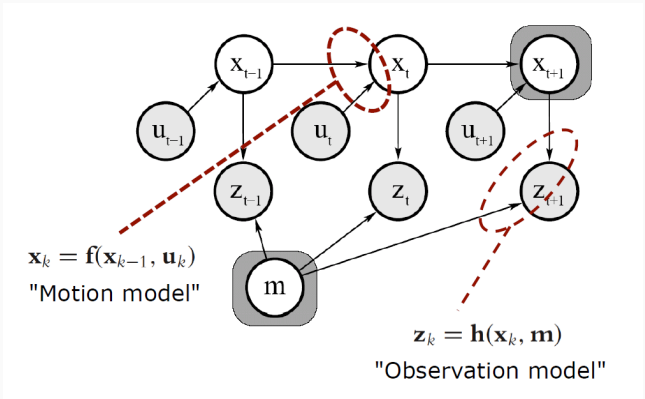
$$P(x_t|x_{t-1}, \mathbf{u}_t)$$

Observation model: how to interpret the observations ?

$$P(o_t|x_{t-1}, \mathcal{M})$$

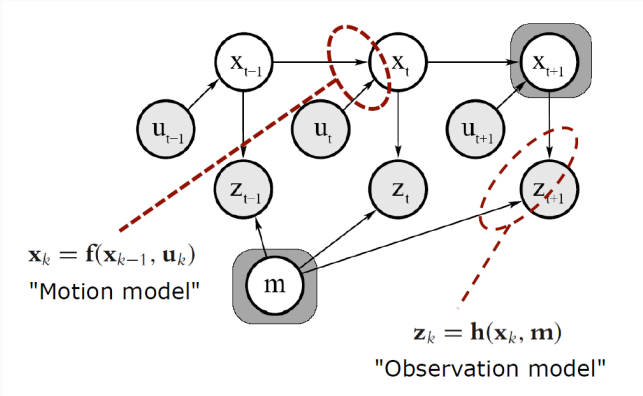
where $\mathcal{M}$ is the representation of the environment, e.g., map.

Probabilistic sensors and dynamics - graphical model



Which models this reminds us of ?

Probabilistic sensors and dynamics - graphical model

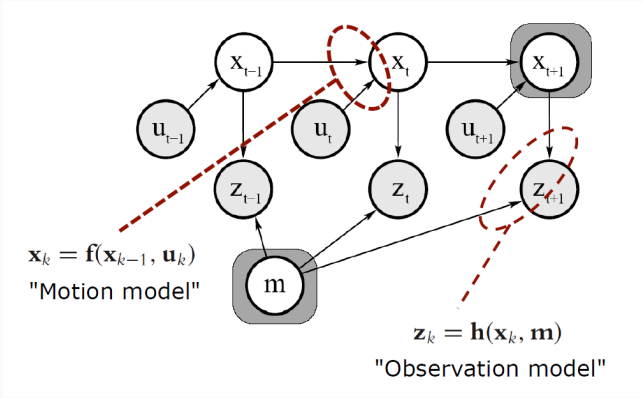


- Hidden Markov Models (HMM)
- Markov Decision Process (MDP)
- Partially observable Markov decision process (POMDP)

All based on the **Markov property** i.e., the future is independent of the past given the present.

Which model is most suitable?

Probabilistic sensors and dynamics - graphical model



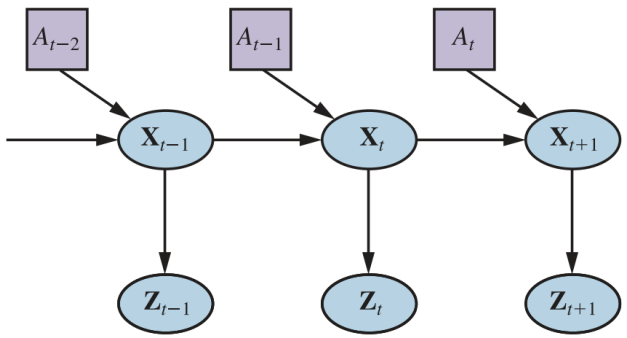
- Hidden Markov Models (HMM)
- Markov Decision Process (MDP)
- Partially observable Markov decision process (POMDP)

All based on the **Markov property** i.e., the future is independent of the past given the present.

Which model is most suitable? It depends!

Bayes' Filter*

1: **Algorithm Bayes_filter**$(bel(x_{t-1}), u_t, z_t)$**:**
2: for all x_t do
3: $\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1})\ bel(x_{t-1})\ dx$
4: $bel(x_t) = \eta\ p(z_t \mid x_t)\ \overline{bel}(x_t)$
5: endfor
6: return $bel(x_t)$



Where is the Markovian assumption used here?

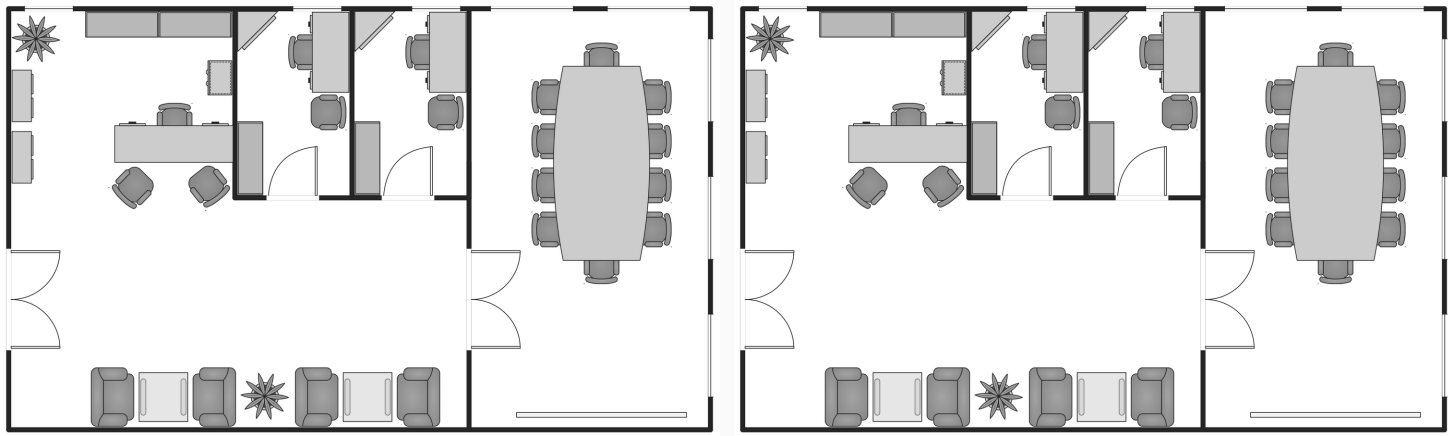
*From Probabilistic Robotics by S. Thrun(2002)

Particle Filters (Monte Carlo Localization)

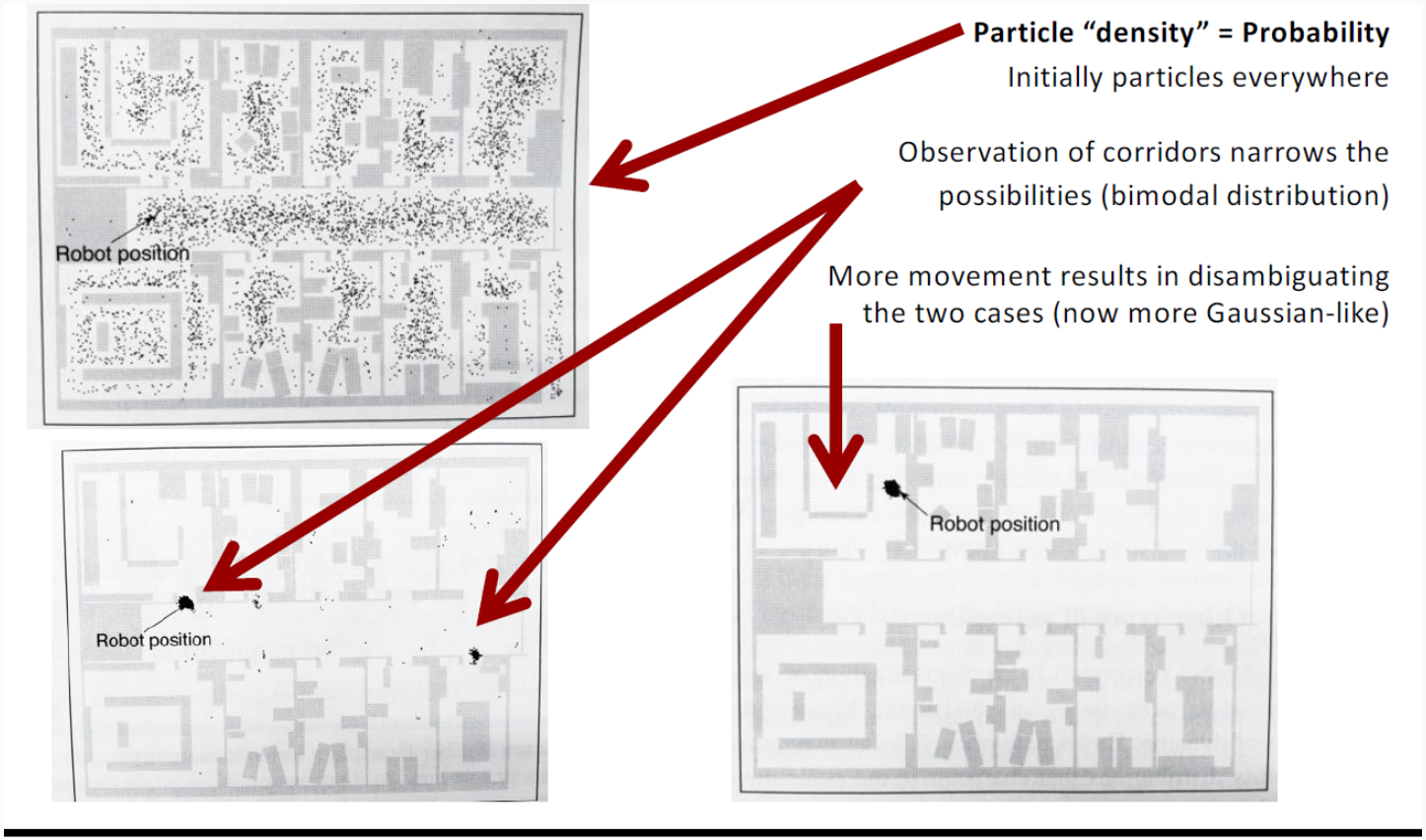
- What if you are in a building with a map, but you have no idea where you are? (ambiguity): you are definitely in a bathroom, but don't know 1st or 2nd floor
- Problem: Gaussians are not the right model of uncertainty

Particle Filters (Monte Carlo Localization)

- What if you are in a building with a map, but you have no idea where you are? (ambiguity): you are definitely in a bathroom, but don't know 1st or 2nd floor
- Problem: Gaussians are not the right model of uncertainty
- Instead, represent the estimated position and uncertainty (i.e., belief) using a constant set of "particles"
- Think of this as "sampling" from a probability distribution
- That is why it is called **Monte Carlo Localization**



Particle Filters (Monte Carlo Localization): Example

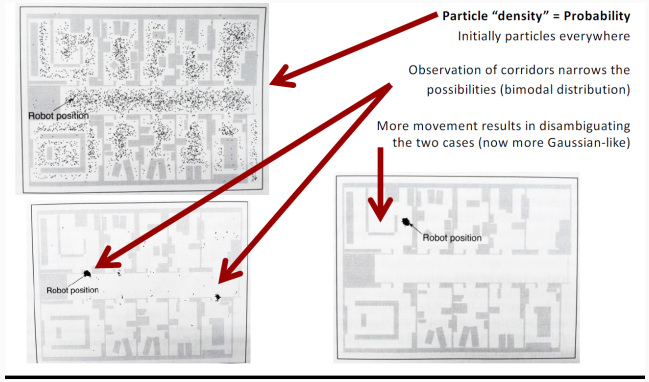


Particle Filters (Monte Carlo Localization): Running example

https://www.youtube.com/watch?v=aUkBa1zMKv4

Particle Filtering

- **Key idea:** use sequential importance sampling to recursively update a set of samples (representing possible configurations) and focus on high-probability samples at the next iteration.
- A recursive update operation (the set of samples).
- Samples collected according to their weight (with replacement, i.e., the same value can be selected multiple times).



https://youtu.be/0OWcPH1EFvc?feature=shared

Particle Filtering: general schema [2]

Uses a Dynamic Bayesian Network (DBN) to represent dependencies between variables.

function PARTICLE-FILTERING($\mathbf{e}, N, dbn$) **returns** a set of samples for the next time step
  inputs: $\mathbf{e}$, the new incoming evidence
    N, the number of samples to be maintained
    dbn, a DBN defined by $\mathbf{P}(\mathbf{X}_0)$, $\mathbf{P}(\mathbf{X}_1 \mid \mathbf{X}_0)$, and $\mathbf{P}(\mathbf{E}_1 \mid \mathbf{X}_1)$
  persistent: S, a vector of samples of size N, initially generated from $\mathbf{P}(\mathbf{X}_0)$
  local variables: W, a vector of weights of size N

  for $i = 1$ to N **do**
    $S[i] \leftarrow$ sample from $\mathbf{P}(\mathbf{X}_1 \mid \mathbf{X}_0 = S[i])$ // *step 1*
    $W[i] \leftarrow \mathbf{P}(\mathbf{e} \mid \mathbf{X}_1 = S[i])$ // *step 2*
  $S \leftarrow$ WEIGHTED-SAMPLE-WITH-REPLACEMENT(N, S, W) // *step 3*
  return S

[2] From Russel and Norvig

Particle Filtering: general schema

function PARTICLE-FILTERING($\mathbf{e}$, N, dbn) **returns** a set of samples for the next time step
  inputs: $\mathbf{e}$, the new incoming evidence
    N, the number of samples to be maintained
    dbn, a DBN defined by $\mathbf{P}(\mathbf{X}_0)$, $\mathbf{P}(\mathbf{X}_1 \mid \mathbf{X}_0)$, and $\mathbf{P}(\mathbf{E}_1 \mid \mathbf{X}_1)$
  persistent: S, a vector of samples of size N, initially generated from $\mathbf{P}(\mathbf{X}_0)$
  local variables: W, a vector of weights of size N

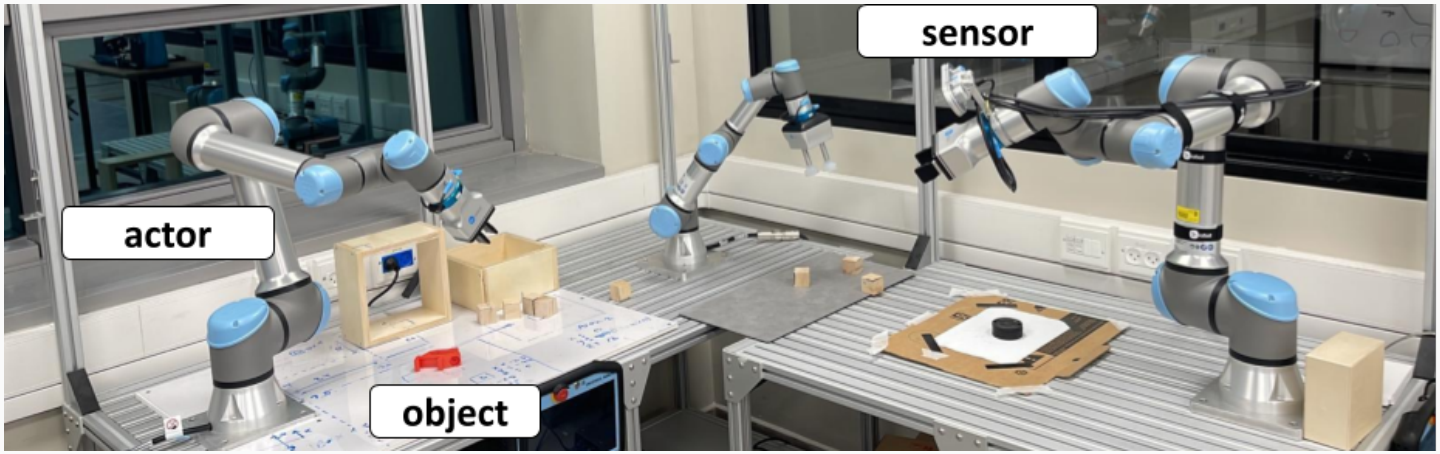
  for $i = 1$ to N **do**
    $S[i] \leftarrow$ sample from $\mathbf{P}(\mathbf{X}_1 \mid \mathbf{X}_0 = S[i])$ // *step 1*
    $W[i] \leftarrow \mathbf{P}(\mathbf{e} \mid \mathbf{X}_1 = S[i])$ // *step 2*
  $S \leftarrow$ WEIGHTED-SAMPLE-WITH-REPLACEMENT(N, S, W) // *step 3*
  return S

- First, a population S of N samples is generated from the prior distribution $P(X_0)$
- Then, N samples are generated by:
  1. **Forward propagation:** sample next state X_1 based on the transition model $P(X_1|X_0)$, and place in S[i].
  2. **Weight assignment:** Assign a weight to the generated sample according to the likelihood of the new evidence using $P(E_1|X_1)$
  3. **Resampling:** The population is *re-sampled* to generate a new population of N samples. The probability that a particular sample is selected is proportional to its weight. The new samples are unweighted.

What happens when this is applied iteratively?

CLAIR lab Examples

From Beliefs to Decisions

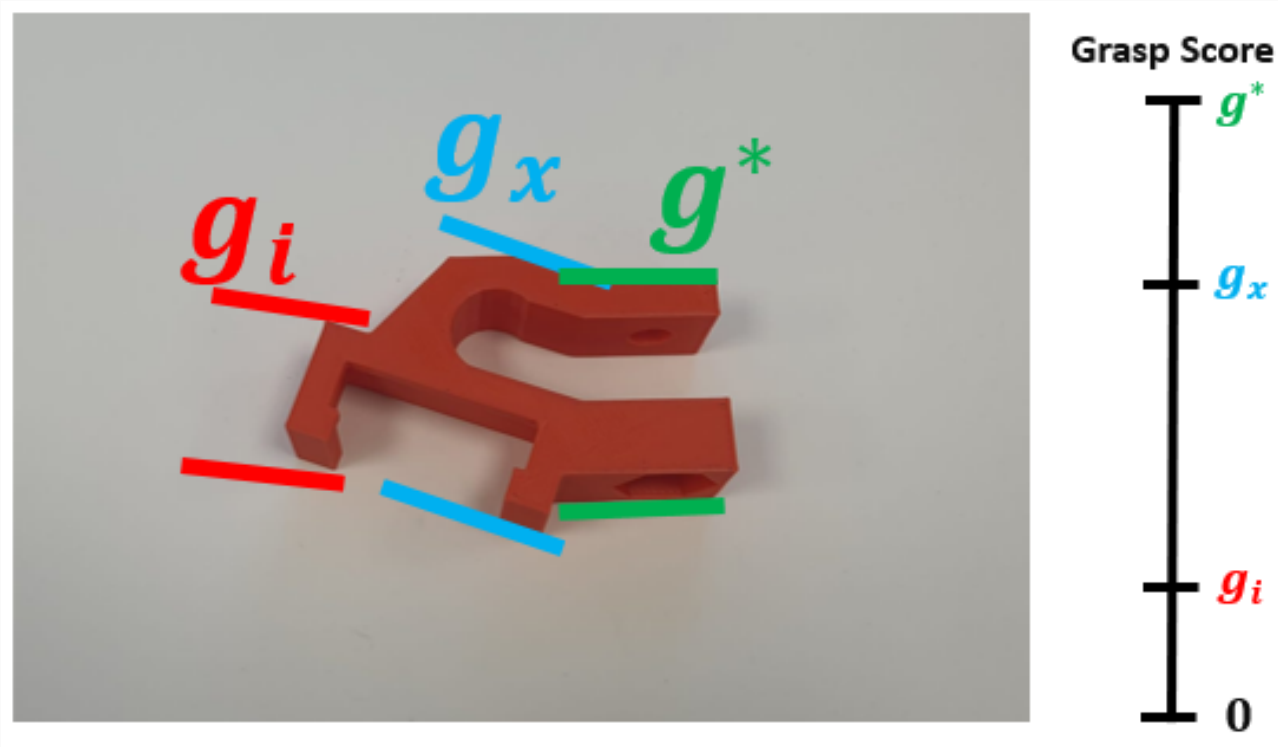


Value of Assistance for Grasping

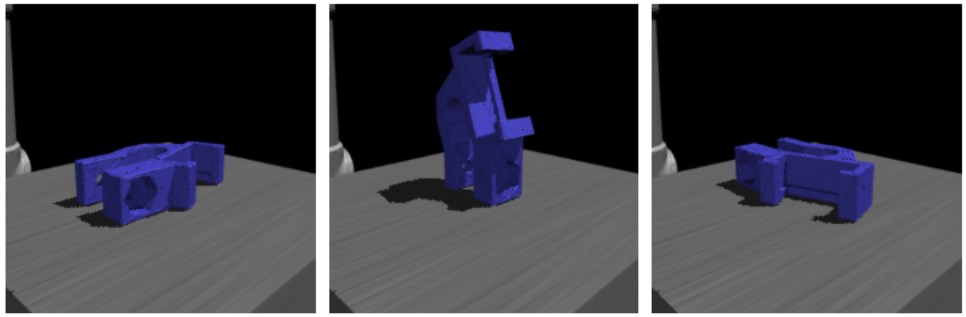
Mohammad Masarwy, Yuval Goshen, David Dovrat, Sarah Keren

https://arxiv.org/abs/2310.14402

From Beliefs to Decisions

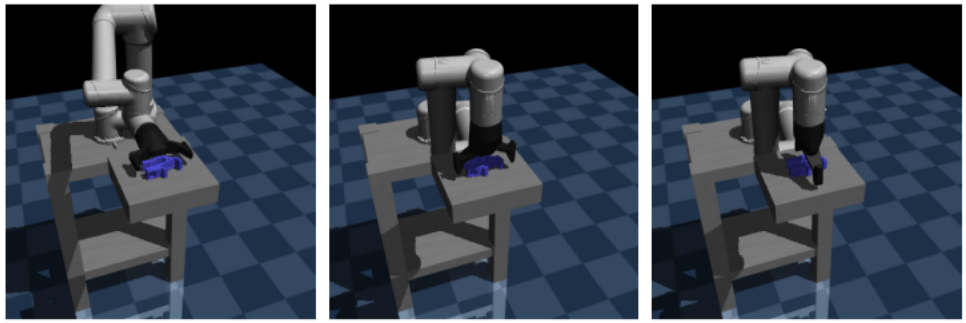


From Beliefs to Decisions



(a) (b) (c)

Fig. 2: Example stable poses.



(a) (b) (c)

Fig. 3: Example grasp configurations from which the actor can attempt to grasp the object - each configuration is associated with a score, i.e., probability of success.

From Beliefs to Decisions

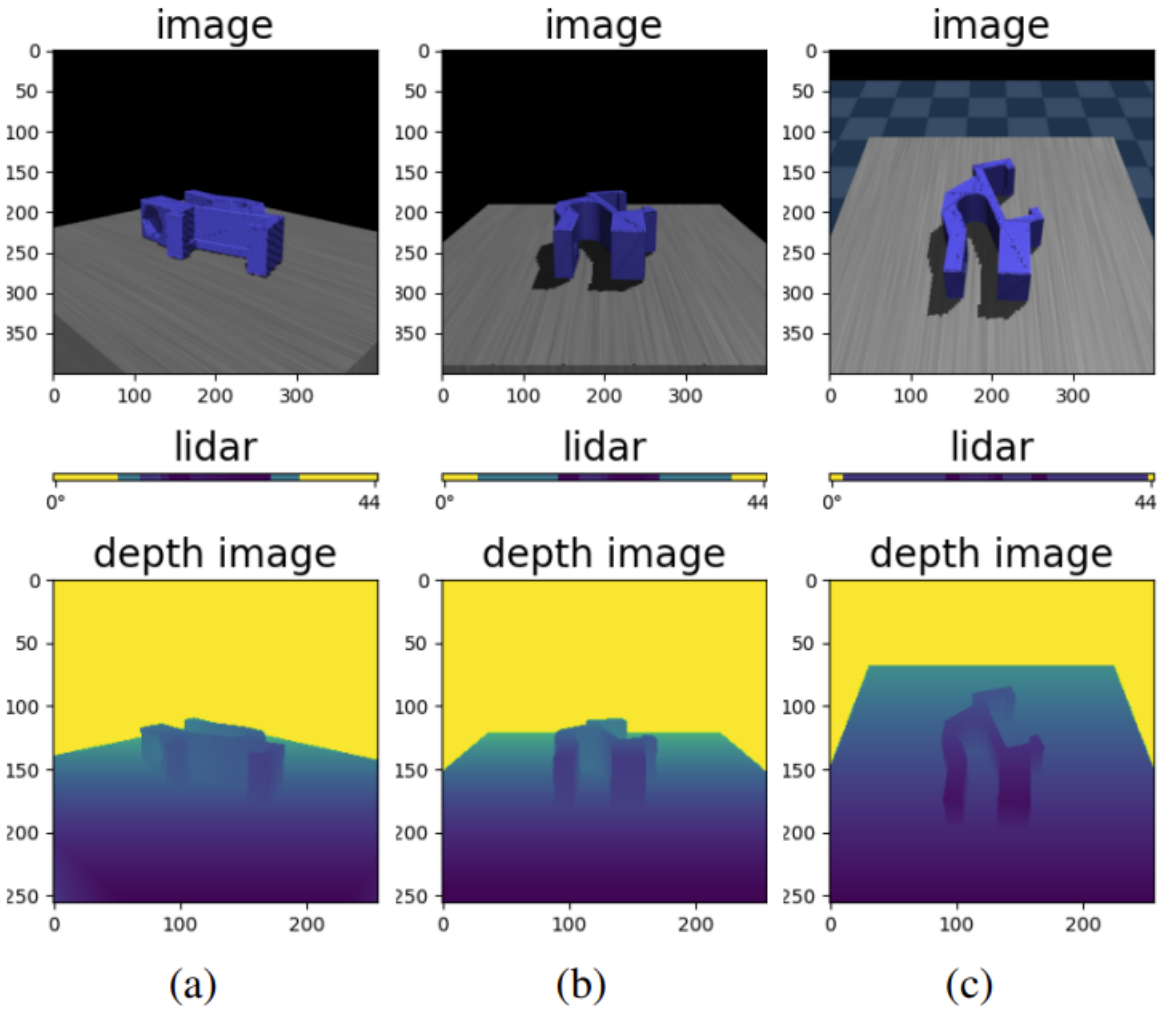
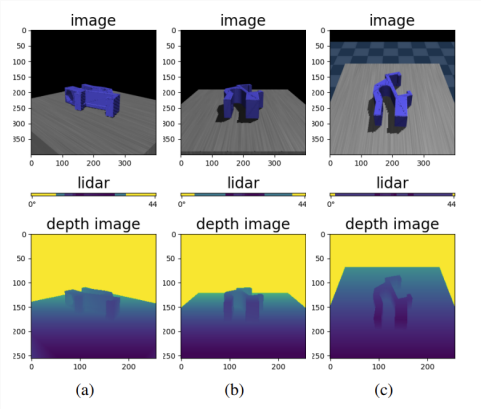


Fig. 4: Example sensor configurations and corresponding observations for a given stable pose. Each column represents the RGB image [top] lidar reading [middle] and depth image [bottom] for a sensor configuration-object pose pair.

From Beliefs to Decisions

Definition (Sensor Function)

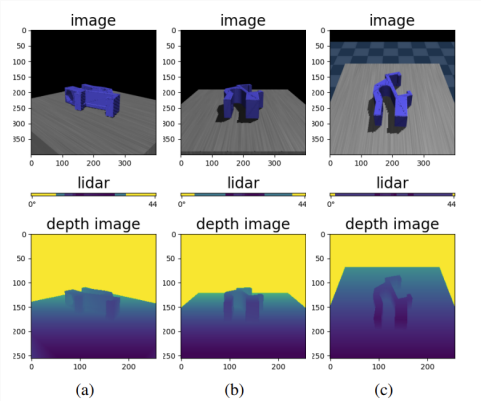
Given object pose $p \in \mathcal{P}$ and sensor configuration $q \in \mathcal{X}$, sensor function $\mathcal{O} : \mathcal{P} \times \mathcal{X} \mapsto \Omega$ is a random function, such that if $o = \mathcal{O}(p, q)$ then $P(o|p, q)$ provides the conditional probability of obtaining the observation o when the sensor configuration is q and the object pose is p.



From Beliefs to Decisions

Definition (Sensor Function)

Given object pose $p \in \mathcal{P}$ and sensor configuration $q \in \mathcal{X}$, sensor function $\mathcal{O} : \mathcal{P} \times \mathcal{X} \mapsto \Omega$ is a random function, such that if $o = \mathcal{O}(p, q)$ then $P(o|p, q)$ provides the conditional probability of obtaining the observation o when the sensor configuration is q and the object pose is p.



Typically, the actual distribution is not known, and we use a *predicted sensor function* $\hat{\mathcal{O}}$ and a *predicted observation probability* $\hat{P}$, which may be incorrect or inaccurate.

From Beliefs to Decisions

We use a similarity score, $\omega : \Omega \times \Omega \mapsto [0, 1]$ to compare the predicted and received observations:

$$\hat{P}(o|p, q) = \frac{\omega(\hat{\mathcal{O}}(p, q), o)}{\int_{p' \in \mathcal{P}} \omega(\hat{\mathcal{O}}(p', q), o) dp'} \tag{3}$$

The literature is rich of various definitions for ω, which may vary between applications and sensor types.

From Beliefs to Decisions

For **belief update** we use a Bayesian filter such that for any observation $o \in \Omega$ taken from sensor configuration $q \in \mathcal{X}$, the updated pose belief $\beta^{o,q}(p)$ for pose $p \in \mathcal{P}$ is given as

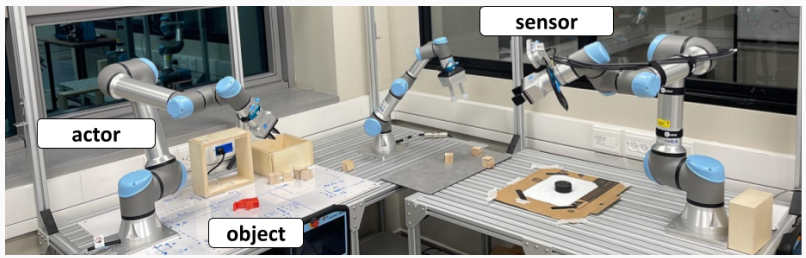
$$\beta^{o,q}(p) = \frac{\hat{P}(o|p,q)\,\beta(p)}{\int_{p' \in \mathcal{P}} \hat{P}(o|p',q)\,\beta(p')\,dp'} \tag{4}$$

where $\beta(p)$ is the estimated probability that p is the object pose prior to considering the new observation o.

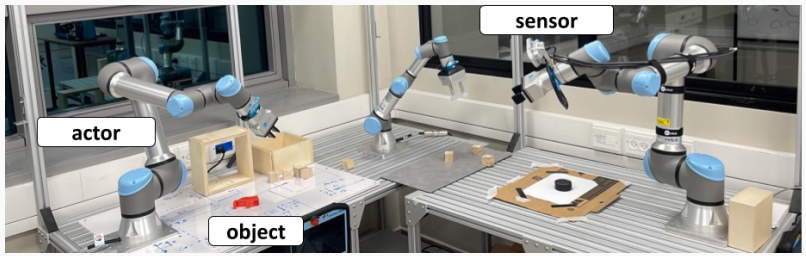
Value of Assistance (VOA) for Grasping

Given the actor's belief $\beta_a \in \mathcal{B}$, the belief of the helping agent $\beta_h \in \mathcal{B}$, the predicated observation probability $\hat{P}$, sensor configuration $q \in \mathcal{X}$, and the actor's belief update function τ_a,

$$U_\alpha^{VOA}(\beta_h, \beta_{ac}) \stackrel{\text{def}}{=\!=} \mathbb{E}_{p\sim\beta_h}\left[\mathbb{E}_{o\sim\hat{P}(o|p,q)}\left[\gamma(g^{max}(\beta_a^{o,q}), p)\right] - \gamma(g^{max}(\beta_{ac}), p))\right]. \tag{5}$$



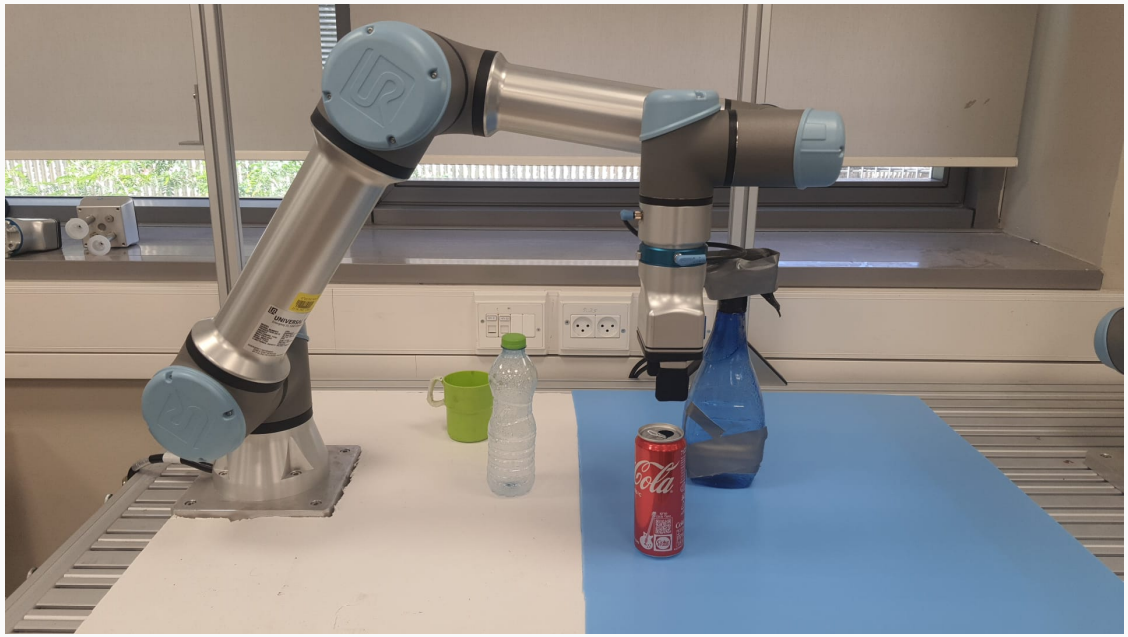
From Beliefs to Decisions



(a) P1 (b) P2 (c) P3 (d) P4 (e) P5 (f) P6

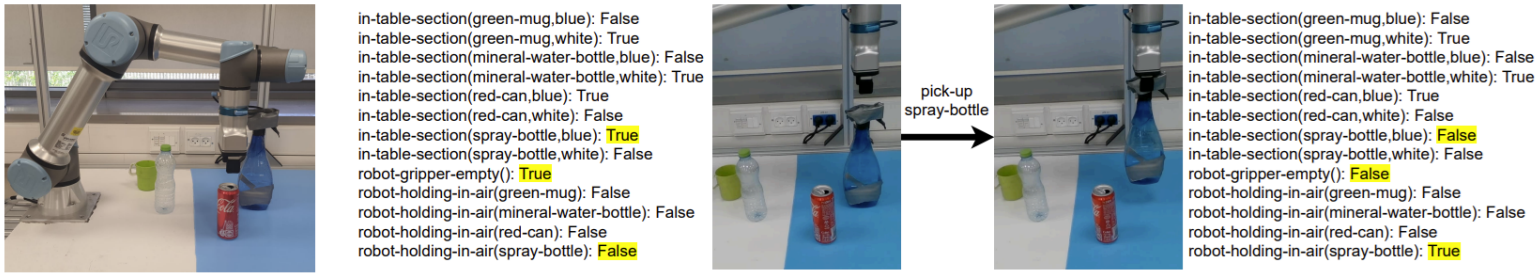
Is this a good observation?

Semantic State Estimation



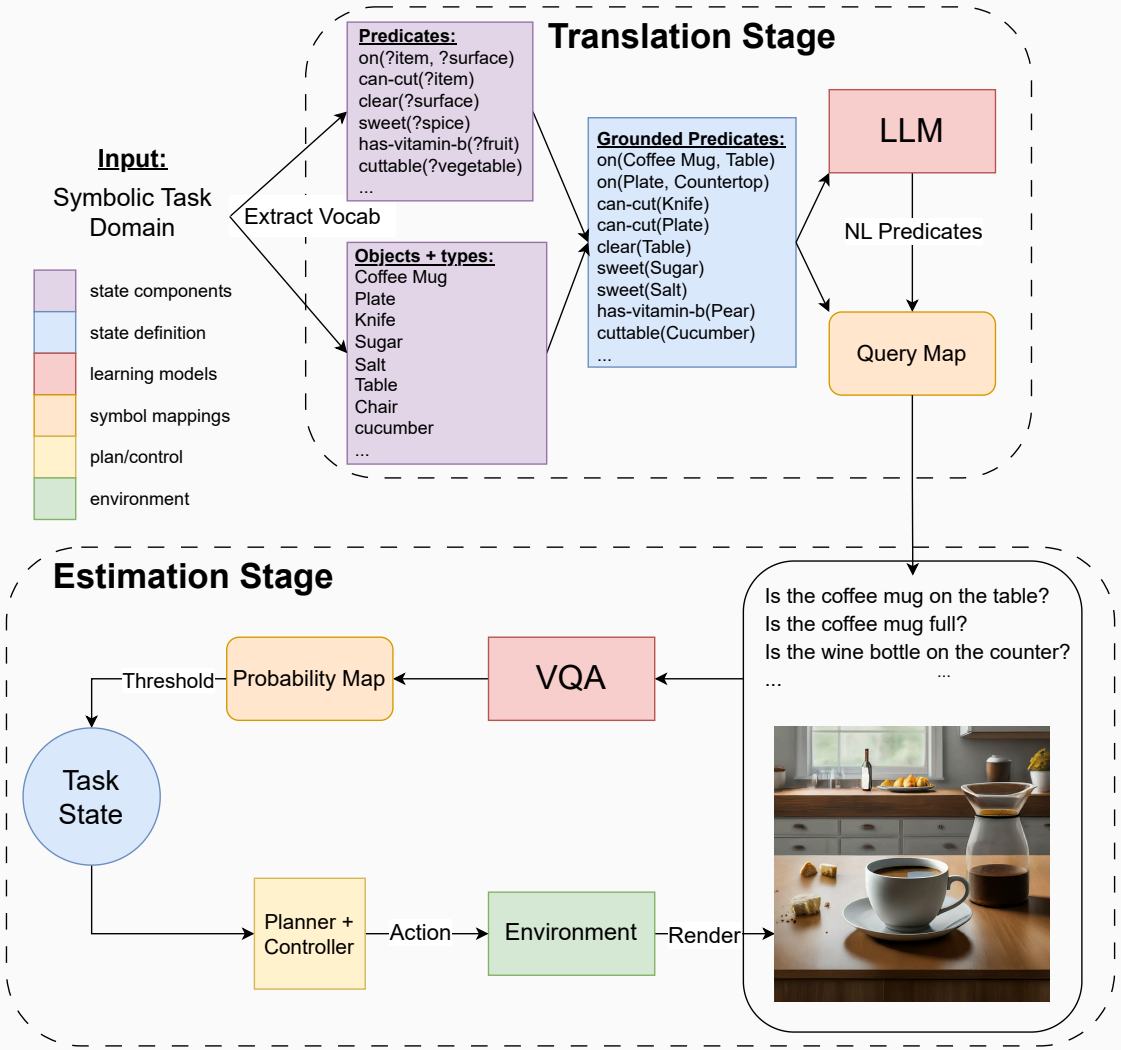
Semantic State Estimation

Semantic State Estimation



(a) Setup. (b) Example transition annotated with S3E. State changes highlighted.

Semantic State Estimation



What about sequential decision-making?

Planning with POMDPs

Planning with POMDPs

- Some relevant links:
  - *https://people.csail.mit.edu/lpk/papers/aij98-pomdp.pdf*
  - *https://people.eecs.berkeley.edu/~pabbeel/cs287-fa13/slides/pomdps.pdf*
  - *https://cs.brown.edu/research/ai/pomdp/tutorial/pomdp-solving.html*
  - *https://www.youtube.com/watch?v=cTu7mvRE354*

Planning with POMDPs

- A policy $\pi : \beta \mapsto \mathcal{A}$ of a POMDP maps the current belief into an action.
- The belief is assumed to be a sufficient statistic and an optimal policy is the solution of a continuous space "belief MDP"

Value iteration for POMDPs

Bellman optimality for MDPS:

$$\mathcal{V}^*(s) = \max_a \sum_{s'} \mathcal{P}(s'|s,a) \left[\mathcal{R}(s,a,s') + \gamma \mathcal{V}^*(s')\right]$$

Bellman optimality for POMDPs:

Value iteration for POMDPs

Bellman optimality for MDPS:

$$\mathcal{V}^*(s) = \max_a \sum_{s'} \mathcal{P}(s'|s,a) \left[\mathcal{R}(s,a,s') + \gamma \mathcal{V}^*(s')\right]$$

Bellman optimality for POMDPs:

$$\mathcal{V}^*(\beta) = \max_a \left[\mathcal{R}(\beta,a) + \gamma \sum_o \mathcal{P}(o|a,\beta)\mathcal{V}^*(\tau(\beta,a,o))\right]$$

Value iteration for POMDPs

Bellman optimality for MDPS:

$$\mathcal{V}^*(s) = \max_a \sum_{s'} \mathcal{P}(s'|s,a) \left[\mathcal{R}(s,a,s') + \gamma \mathcal{V}^*(s')\right]$$

Bellman optimality for POMDPs:

$$\mathcal{V}^*(\beta) = \max_a \left[\mathcal{R}(\beta,a) + \gamma \sum_o \mathcal{P}(o|a,\beta)\mathcal{V}^*(\tau(\beta,a,o))\right]$$

Update formula for MDPS:

$$\mathcal{V}_{k+1}(s) = \max_a \sum_{s'} \mathcal{P}(s'|s,a) \left[\mathcal{R}(s,a,s') + \gamma \mathcal{V}_k(s')\right]$$

Value iteration for POMDPs

Bellman optimality for MDPS:

$$\mathcal{V}^*(s) = \max_a \sum_{s'} \mathcal{P}(s'|s,a) \left[\mathcal{R}(s,a,s') + \gamma \mathcal{V}^*(s')\right]$$

Bellman optimality for POMDPs:

$$\mathcal{V}^*(\beta) = \max_a \left[\mathcal{R}(\beta,a) + \gamma \sum_o \mathcal{P}(o|a,\beta)\mathcal{V}^*(\tau(\beta,a,o))\right]$$

Update formula for MDPS:

$$\mathcal{V}_{k+1}(s) = \max_a \sum_{s'} \mathcal{P}(s'|s,a) \left[\mathcal{R}(s,a,s') + \gamma \mathcal{V}_k(s')\right]$$

Update formula for POMDPs:

$$\mathcal{V}_{k+1}(\beta) = \max_a \left[\mathcal{R}(\beta,a) + \gamma \sum_{o\in\Omega} \mathcal{O}(o|a,\beta)\mathcal{V}_k(\tau(\beta,a,o))\right]$$

Value iteration for POMDPs

$$\mathcal{V}_{k+1}(\beta) = \max_a \left[\mathcal{R}(\beta, a) + \gamma \sum_{o \in \Omega} \mathcal{O}(o|a, \beta) \mathcal{V}_k(\tau(\beta, a, o)) \right]$$

Problems?

Value iteration for POMDPs

$$\mathcal{V}_{k+1}(\beta) = \max_a \left[\mathcal{R}(\beta, a) + \gamma \sum_{o \in \Omega} \mathcal{O}(o|a, \beta) \mathcal{V}_k(\tau(\beta, a, o)) \right]$$

Problems?

- Reward is not a function of the belief, but of the state
- While states are discrete - beliefs are continuous (so the space is ∞)

Value iteration for POMDPs

Reward is a function of the state:

$$\mathcal{V}_{k+1}(\beta) = \max_a \left[\mathcal{R}(\beta, a) + \gamma \sum_{o \in \Omega} \mathcal{O}(o|a, \beta) \mathcal{V}_k(\tau(\beta, a, o)) \right]$$

Value iteration for POMDPs

Reward is a function of the state:

$$\mathcal{V}_{k+1}(\beta) = \max_a \left[\mathcal{R}(\beta, a) + \gamma \sum_{o \in \Omega} \mathcal{O}(o|a, \beta) \mathcal{V}_k(\tau(\beta, a, o)) \right]$$

Beliefs are continuous:

Value iteration for POMDPs

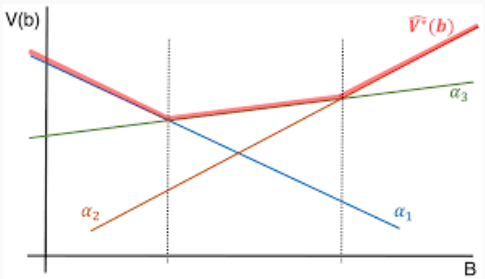
Reward is a function of the state:

$$\mathcal{V}_{k+1}(\beta) = \max_a \left[\mathcal{R}(\beta, a) + \gamma \sum_{o \in \Omega} \mathcal{O}(o|a, \beta) \mathcal{V}_k(\tau(\beta, a, o)) \right]$$

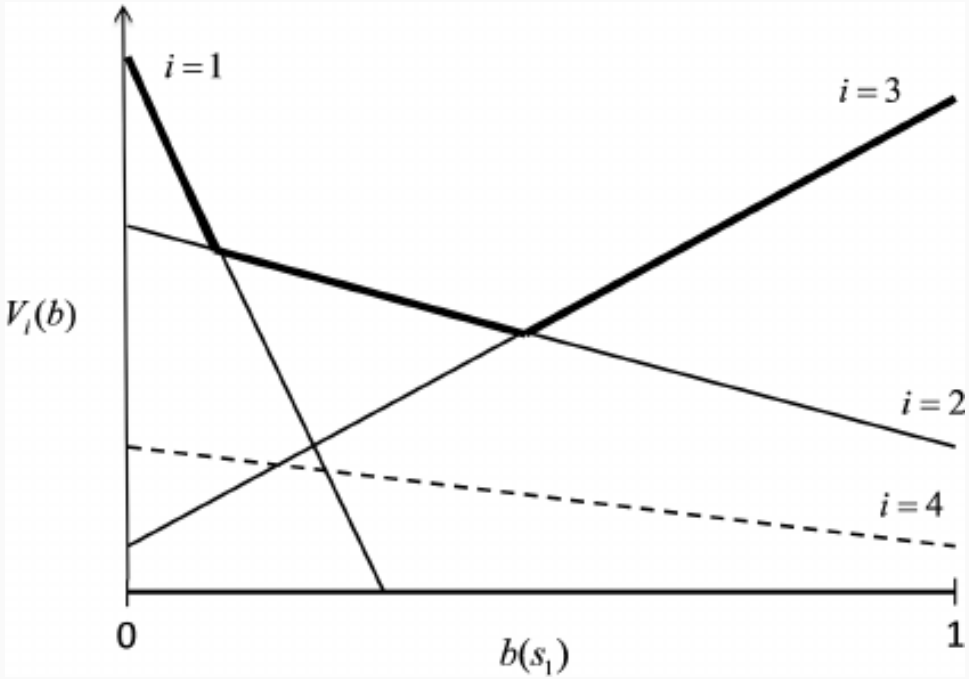
Beliefs are continuous:

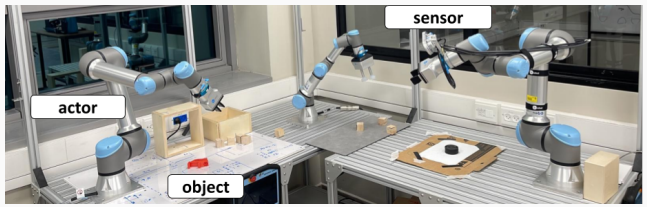
Instead of considering beliefs, we consider a finite set of α-vectors that represent beliefs.

$$\mathcal{V}_k(\beta) = max_{\alpha \in \Gamma_k} \alpha \cdot \beta = max_{\alpha \in \Gamma_k} \Sigma_s \alpha(s) \cdot \beta(s)$$



Value iteration for POMDPs





Value iteration for POMDPs

function POMDP-VALUE-ITERATION($pomdp, \epsilon$) **returns** a utility function
  inputs: $pomdp$, a POMDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    sensor model $P(e \mid s)$, rewards $R(s)$, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', sets of plans p with associated utility vectors α_p

  $U' \leftarrow$ a set containing just the empty plan $[\,]$, with $\alpha_{[]}(s) = R(s)$
  repeat
    $U \leftarrow U'$
    $U' \leftarrow$ the set of all plans consisting of an action and, for each possible next percept,
      a plan in U with utility vectors
    $U' \leftarrow$ REMOVE-DOMINATED-PLANS(U')
  until MAX-DIFFERENCE(U, U') $\leq \epsilon(1-\gamma)/\gamma$
  return U

- A high-level sketch of the value iteration algorithm for POMDPs.
- The REMOVE-DOMINATED-PLANS step and MAX-DIFFERENCE test are typically implemented as linear programs.

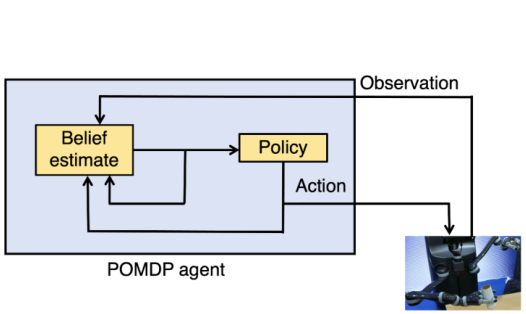
Planning with POMDPs

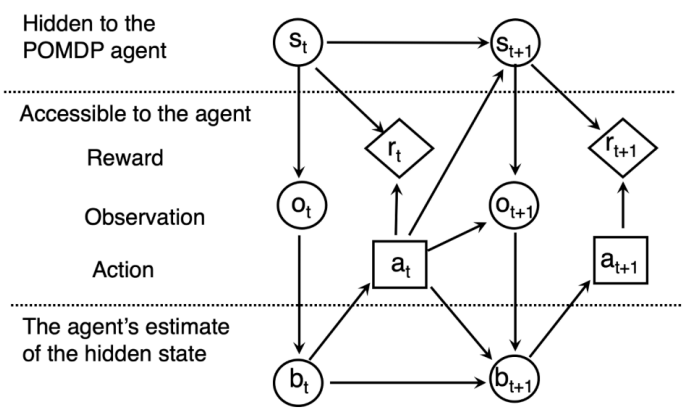
- SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. Kurniawati et al. 2008 *https://bigbird.comp.nus.edu.sg/m2ap/wordpress/wp-content/uploads/2016/01/rss08.pdf*
- Efficient point-based POMDP planning by approximating optimally reachable belief spaces: Kurniawati (2021): *https://arxiv.org/pdf/2107.07599.pdf*

Planning with POMDPs

When a POMDP $\langle \mathcal{S}, \mathcal{A}, \mathbb{P}, \mathcal{R}, \gamma, \Omega, \mathcal{O}, \beta_0 \rangle$ is used to represent a robot's task

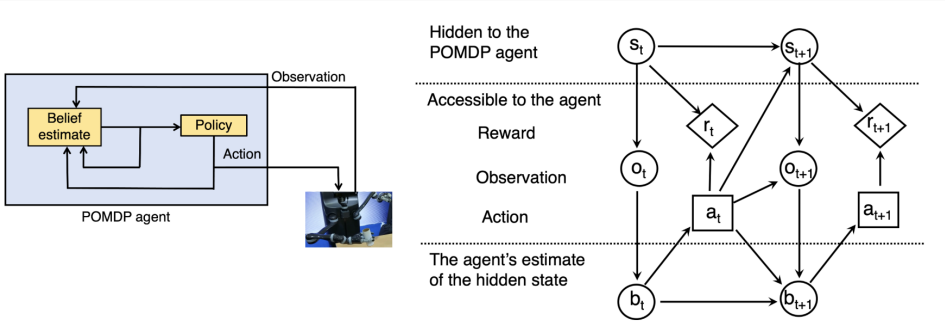
- the transition function is typically represented as a noisy dynamics function $s' = f(s, a, \eta)$, where $s, s' \in \mathcal{S}$ and $\eta \sim N$ is a noise vector sampled from noise distribution N, while $f(\cdot)$ denotes the system's dynamics.
- Similarly, $\mathcal{O}$ denotes the sensor/ observation function, representing errors and noise in measurement and perception.





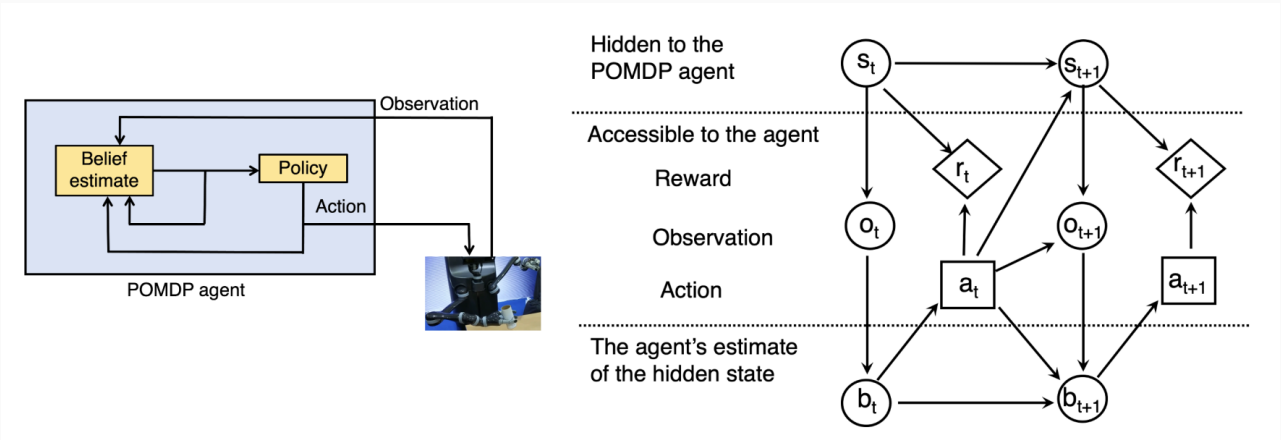
Planning with POMDPs

- POMDP is powerful in its quantification of the non-deterministic effects of actions and partial observability due to errors in sensor measurements and in perception
- The computed policy will balance information gathering and goal attainment.
- But precisely because of this, POMDP is notorious for its high computational complexity and deemed impractical for robotics.
- Until recently, most benchmark problems for POMDPs had less than 30 states and the best algorithms that could solve them took hours.



Planning with POMDPs

- In the past 2 decades, POMDPs solving capabilities have advanced tremendously, thanks to **sampling-based approximate solvers**.
- Although optimality is compromised, robustness and computational efficiency is improved: practical for many realistic robotics problems.



Planning with POMDPs

Algorithm 1 A typical program skeleton for sampling-based POMDP solvers

1: Initialize policy π and a set of sampled beliefs B
{Generally, B is initialised to contain only a single belief (e.g., the initial belief b_0)}
2: **repeat**
3: Sample a (set of) beliefs {Some methods sample histories (a history is a sequence of action–observation tuples) rather than beliefs. In POMDPs, beliefs provide sufficient statistics of the entire history [25], and therefore the two provide equivalent information}
4: Estimate the values of the sampled beliefs
{Generally, via a combination of heuristics and update / backup operation}
5: Update π {In most methods, this step is a byproduct of the previous step}
6: **until** Stopping criteria is satisfied

- Key idea: sample a set of representative beliefs and computes optimal policy only for them, thus substantially reducing complexity.
- Which set would be sufficiently representative ?
  - A variety of sampling strategies have been proposed to select the sample set and to estimate the values of the sampled beliefs.
  - Most sampling-based approximate POMDP solvers are **anytime**
  - Some methods compute upper and lower bound estimates of the value functions
  - Can be broadly divided into offline and online.

SARSOP

SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. Kurniawati et al. 2008
https://bigbird.comp.nus.edu.sg/m2ap/wordpress/wp-content/uploads/2016/01/rss08.pdf

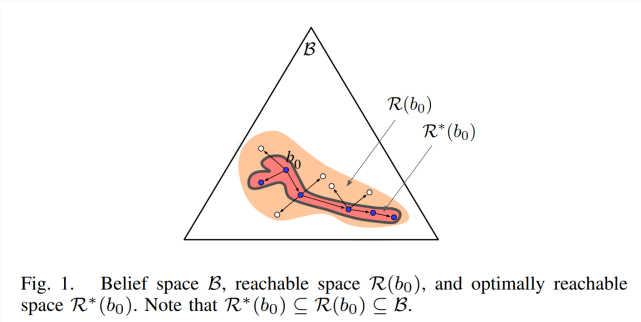


Fig. 1. Belief space $\mathcal{B}$, reachable space $\mathcal{R}(b_0)$, and optimally reachable space $\mathcal{R}^*(b_0)$. Note that $\mathcal{R}^*(b_0) \subseteq \mathcal{R}(b_0) \subseteq \mathcal{B}$.

- Some early POMDP algorithms sample the entire belief space B, using a uniform sampling distribution, such as a grid.
- More recent point-based algorithms sample only $\mathcal{R}(\beta_0)$, the subset of belief points reachable from a given initial point $\beta_0 \in \mathcal{B}$ under arbitrary sequences of actions.

SARSOP

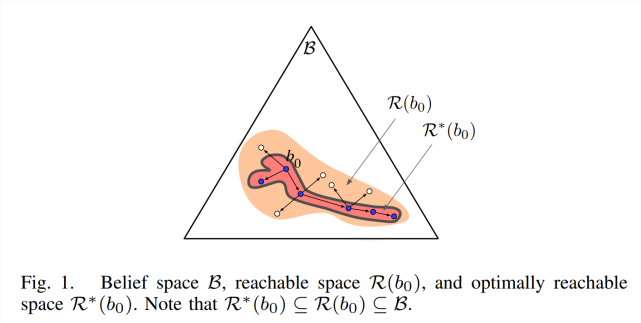


Fig. 1. Belief space $\mathcal{B}$, reachable space $\mathcal{R}(b_0)$, and optimally reachable space $\mathcal{R}^*(b_0)$. Note that $\mathcal{R}^*(b_0) \subseteq \mathcal{R}(b_0) \subseteq \mathcal{B}$.

- SASOP pushes this direction further, by sampling near $\mathcal{R}^*(\beta_0)$, a subset of belief points reachable from β_0 under we**optimal sequences of actions** ($\mathcal{R}^*(\beta_0)$ is usually much smaller than $\mathcal{R}(\beta_0)$).
- Optimality not achievalbe, so approximations of $\mathcal{R}^*(\beta_0)$ are used.
  - Use successive approximations of $\mathcal{R}^*(\beta_0)$ and converge to it iteratively.
  - The algorithm relies on heuristic exploration to sample $\mathcal{R}(\beta_0)$ and improves sampling over time through a simple on-line learning technique.
  - Bounding technique are used to avoid sampling in regions that are unlikely to be optimal
  - This leads to substantial gain in computational efficiency

SARSOP- Successive Approximations of the Reachable Space under Optimal Policies

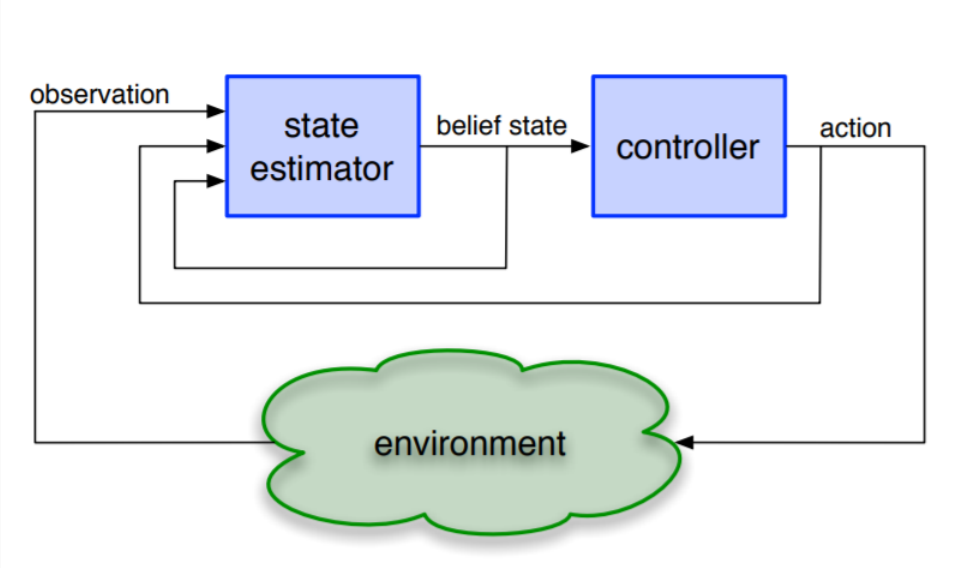
Require: A POMDP $(S, A, T, R, \Omega, O, \gamma)$
  Initialize belief space B
  Initialize lower bound α and upper bound β
  Set the initial belief b_0
  while convergence not achieved **do**
    Sample belief points reachable under current policy
    Refine belief points using forward simulation
    Update the lower bound α via backup operations
    Update the upper bound β to approximate the value function
    Prune irrelevant belief points from B
  end while
  Return policy π derived from α

What if we don't have full access to the model of the environment ?

MCTS in POMDPs

Beliefs and Belief Tracking

- The agent maintains its belief via a **state estimator** - which we will refer to as the process of **Belief Tracking**.



From Kaelbling, L. P., and T. Lozano-Perez. "Integrated Task and Motion Planning in Belief Space" 2013 *https://dspace.mit.edu/bitstream/handle/1721.1/87038/Kaelbling_Integrated%20task.pdf?sequence=1&isAllowed=y*

Partially Observable Markov Decision Process (POMDP)

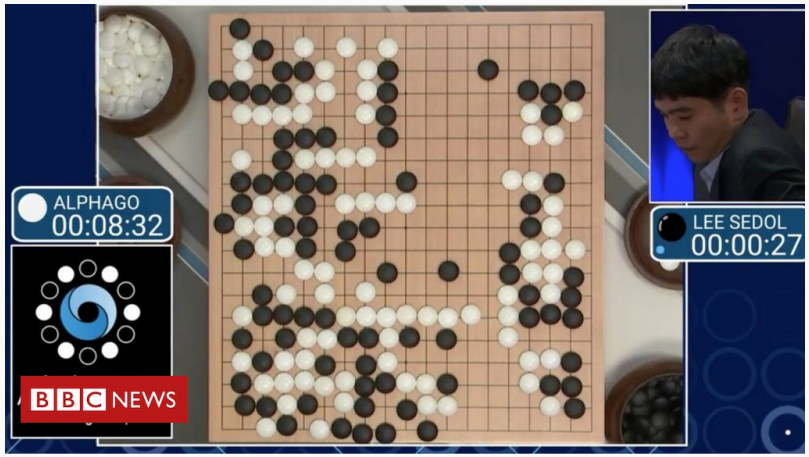
A **Partially Observable Markov Decision Process**(POMDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \Omega, \mathcal{O}, \beta_0 \rangle$ where

- $\mathcal{S}$, $\mathcal{A}$, $\mathcal{P}$, $\mathcal{R}$ and γ are as for an MDP.
- Ω is a set of observations (observation tokens),
- $\mathcal{O}$ is a sensor function specifying the conditional observation probabilities $\mathcal{O}^o_{s,a} = \mathcal{P}[O_{t+1} = o | S_t = s, A_t = a]$ of receiving observation token $o \in \Omega$ in state s after applying action a [3].
- β_0 the initial belief: a probability distribution over the states such that $\beta_0(s)$ stands for the probability of s being the true initial state.

[3] alternatively: $\mathcal{O}^o_s = \mathcal{P}[o_t = o | S_t = s]$

Monte Carlo Tree Search- Reminder

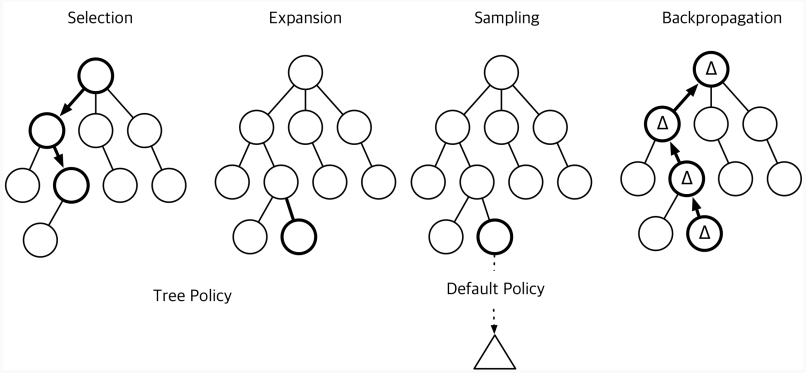
- Combines exploration and exploitation to choose an action in a tree search setting
- Is relevant to classical planning, stochastic planning, and game playing
  - Revolutionized the world of computer Go
- Has many different variants
- Explosion in interest, applications far beyond games: Planning, motion planning, optimization, finance, energy management



Monte Carlo Tree Search- Reminder

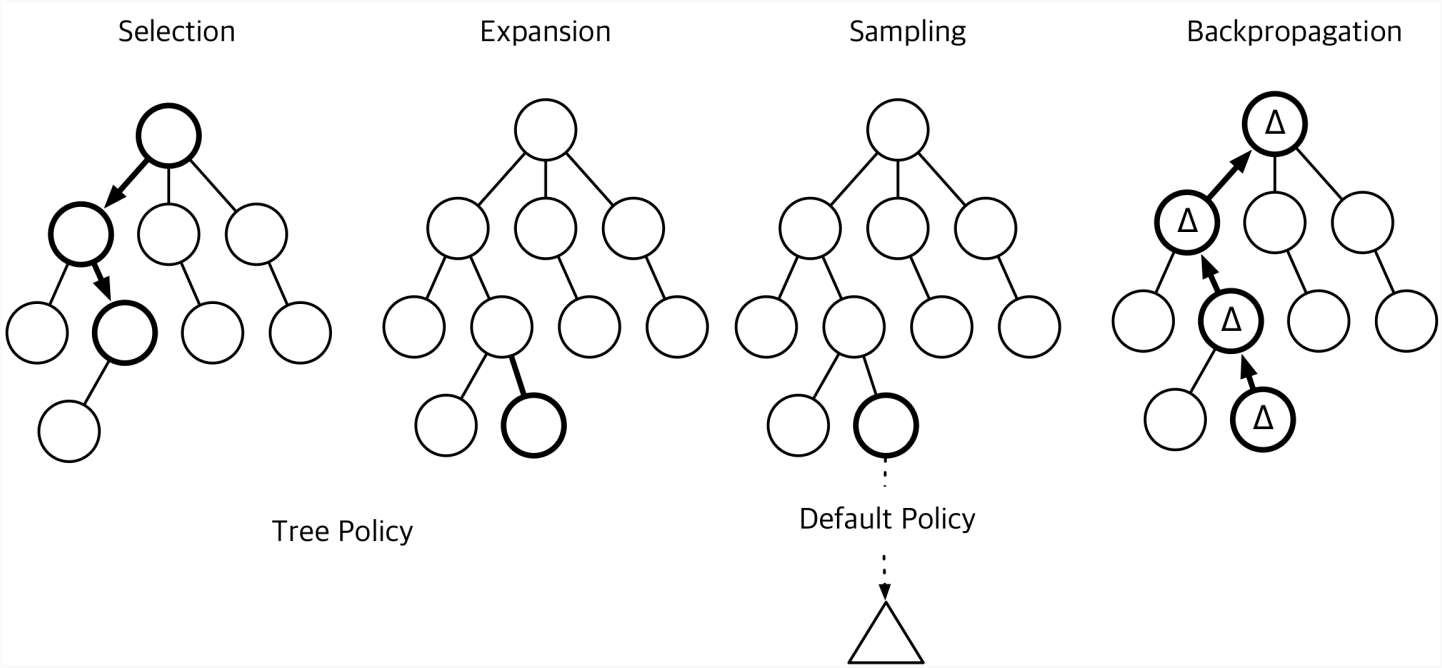
Key Idea: use **Monte Carlo simulation** to accumulate value estimates to guide towards highly rewarding trajectories in a **search tree**.

- Each simulation consists of two phases
  - **Tree policy:** pick actions to maximise values
  - **Default / roll-out policy:** pick actions randomly to simulate a trajectory.
- Repeat (each simulation)
  - **Evaluate** states $Q(S, A)$ by Monte-Carlo evaluation
  - **Improve** tree policy e.g. by ϵ-greedy.
- Converges on the optimal search tree $Q(S, A) \rightarrow q(S, A)$



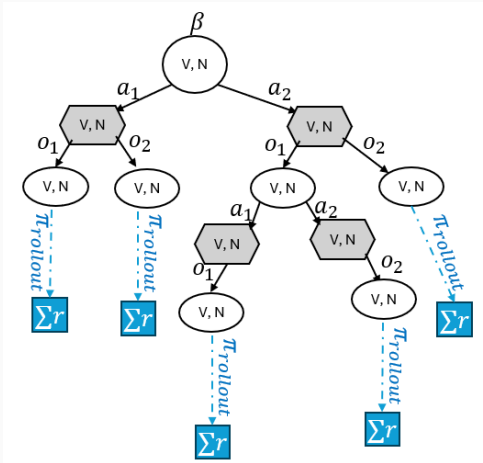
Monte Carlo Tree Search- Reminder

function MONTE-CARLO-TREE-SEARCH(*state*) **returns** *an action*
  tree ← NODE(*state*)
  while IS-TIME-REMAINING() **do**
    leaf ← SELECT(*tree*)
    child ← EXPAND(*leaf*)
    result ← SIMULATE(*child*)
    BACK-PROPAGATE(*result*, *child*)
  return the move in ACTIONS(*state*) whose node has highest number of playouts

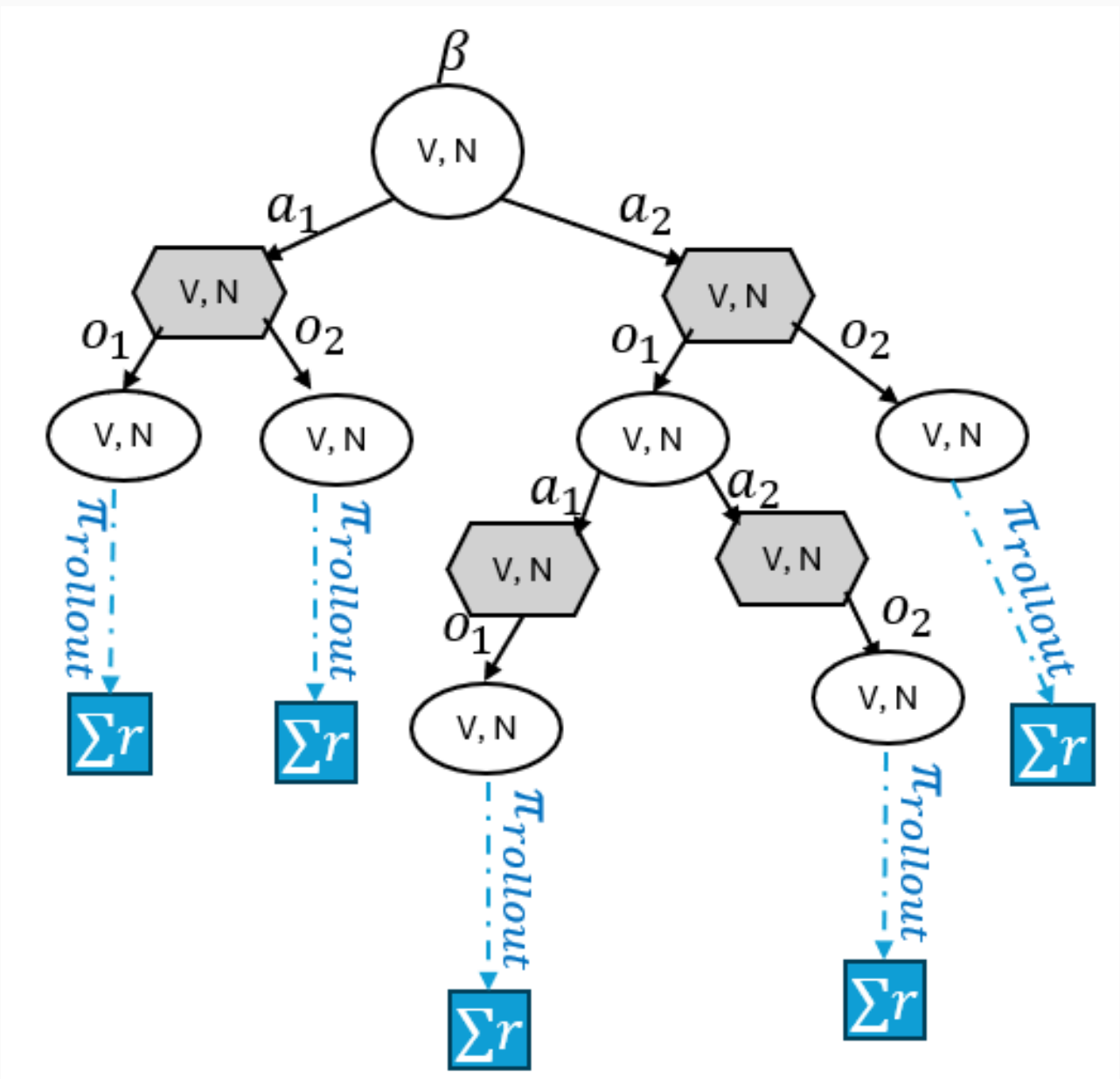


POMCP

- Extending MCTS to POMDPs.
- In a problem with n states, value iteration reasons about an n-dimensional belief state.
- Furthermore, the number of histories that it must evaluate is exponential in the horizon.
- Basic idea: use Monte-Carlo in two ways:
  - sampling start states from the belief state
  - sampling histories using a black box simulator.
- The algorithm constructs online a search tree of histories.



POMCP


β
V, N
a_1
a_2
V, N
V, N
o_1
o_2
o_1
o_2
V, N
V, N
V, N
V, N
a_1
a_2
V, N
V, N
o_1
o_2
V, N
V, N
$\pi_{rollout}$
$\pi_{rollout}$
$\pi_{rollout}$
$\pi_{rollout}$
$\pi_{rollout}$
$\pi_{rollout}$
$\sum r$
$\sum r$
$\sum r$
$\sum r$
$\sum r$

POMCP

Algorithm 1 Partially Observable Monte-Carlo Planning

procedure SEARCH(h)
  repeat
    if $h = empty$ **then**
      $s \sim \mathcal{I}$
    else
      $s \sim B(h)$
    end if
    SIMULATE($s, h, 0$)
  until TIMEOUT()
  return $\underset{b}{\text{argmax}}\, V(hb)$
end procedure

procedure ROLLOUT($s, h, depth$)
  if $\gamma^{depth} < \epsilon$ **then**
    return 0
  end if
  $a \sim \pi_{rollout}(h, \cdot)$
  $(s', o, r) \sim \mathcal{G}(s, a)$
  return r $+ \gamma$.ROLLOUT($s', hao, depth{+}1$)
end procedure

procedure SIMULATE($s, h, depth$)
  if $\gamma^{depth} < \epsilon$ **then**
    return 0
  end if
  if $h \notin T$ **then**
    for all $a \in \mathcal{A}$ **do**
      $T(ha) \leftarrow (N_{init}(ha), V_{init}(ha), \emptyset)$
    end for
    return ROLLOUT($s, h, depth$)
  end if
  $a \leftarrow \underset{b}{\text{argmax}}\, V(hb) + c\sqrt{\frac{\log N(h)}{N(hb)}}$
  $(s', o, r) \sim \mathcal{G}(s, a)$
  $R \leftarrow r + \gamma$.SIMULATE($s', hao, depth + 1$)
  $B(h) \leftarrow B(h) \cup \{s\}$
  $N(h) \leftarrow N(h) + 1$
  $N(ha) \leftarrow N(ha) + 1$
  $V(ha) \leftarrow V(ha) + \frac{R - V(ha)}{N(ha)}$
  return R
end procedure

Other Approaches

RL in Partially Observable Settings: Approaches

- Point-Based Value Iteration (PBVI)
- Model-free RL (e.g., DQN or policy gradient) using belief states or raw observations as input.
- Value Iteration Networks (VIN): Use neural networks to approximate value iteration in belief space.
- Factored POMDPs: Decompose the state and action spaces into smaller factors for efficient computation.
- Bayesian Reinforcement Learning