Mobile Robotics, Planning under uncertainty

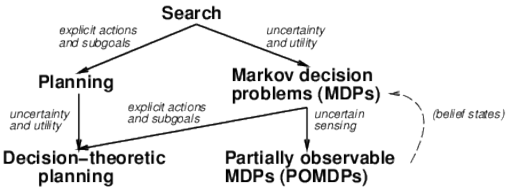
Material based on the book Material based on the book Autonomous Mobile Robot 2nd Ed. (Siegwart, Nourbakhsh, Scaramuzza) [AMR], Chapter 14, and Planning Algorithms [Steve LaValle], partly

Summary

- Introduction to planning under uncertainty
- Markov Decision Processes
- Partially Observable MDPs
- Partial Observable Monte Carlo Planning

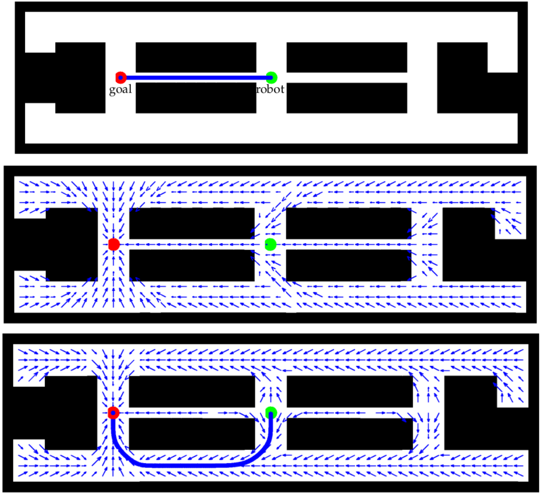
Planning Under Uncertainty: Introduction

- Path planning
  - explicit actions and goals
  - known state, deterministic actions
  - solution is a sequence of movements
- Planning under uncertainty
  - utilities
  - non-deterministic actions (MDPs)
  - unknown state (POMDPs)
  - solution is a policy



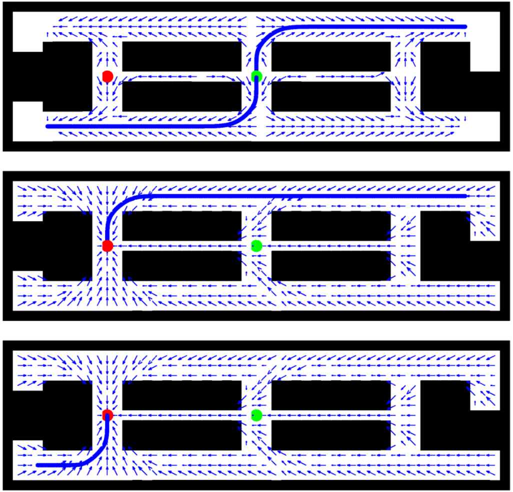
Courtesy Russel and Norvig, Artificial Intelligence a Modern Approach

Planning under uncertainty, non-deterministic actions



Different solutions for path planning when considering non-deterministic actions, Source [PR]

Planning under uncertainty, partially observable state

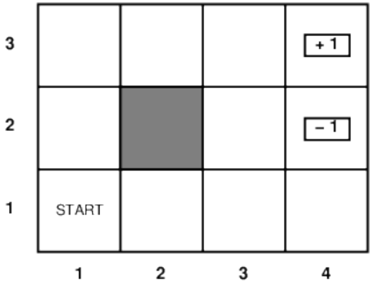


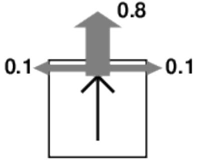
Knowledge gathering actions in POMDPs, Source [PR]

Markov Decision Processes

- MDPs: a general class of non-deterministic search problem
- Four components: $\langle S, A, R, Pr \rangle$
- S a (finite) set of states ($|S| = n$)
- A a (finite) set of actions ($|A| = m$)
- Transition function $p(s'|s,a) = Pr\{S_{t+1} = s' | S_t = s, A_t = a\}$
- Real valued reward function $r(s,a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$

Example problem: exploring a maze





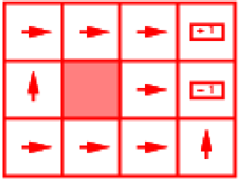
States $s \in S$, actions $a \in A$

Model $T(s, a, s') \equiv P(s'|s, a)$ = probability that a in s leads to s'

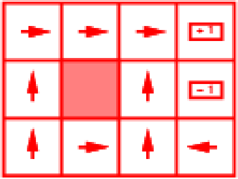
Reward function $R(s)$ (or $R(s, a)$, $R(s, a, s')$)

$$= \begin{cases} -0.04 & \text{(small penalty) for nonterminal states} \\ \pm 1 & \text{for terminal states} \end{cases}$$

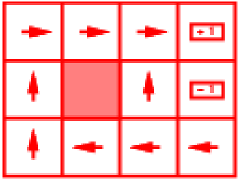
Risk and reward



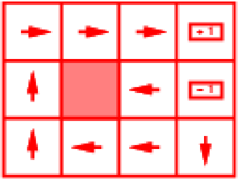
$r = [-\infty : -1.6284]$



$r = [-0.4278 : -0.0850]$



$r = [-0.0480 : -0.0274]$



$r = [-0.0218 : 0.0000]$

Markov Property and other assumptions

- Markov Dynamics (history independence)

$Pr\{R_{t+1}, S_{t+1} | S_0, A_0, R_1, \cdots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\}$

Markov property:

$Pr\{R_{t+1}, S_{t+1} | S_t, A_t\}$

- Stationary (not dependent on time)

$Pr\{R_{t+1}, S_{t+1} | S_t, A_t\} = Pr\{R_{t'+1}, S_{t'+1} | S_{t'}, A_{t'}\} \forall\, t, t'$

- Full observability: we can not predict exactly which state we will reach but we know where we are

Policies

- Types of policy
  - Non-stationary policy
    - $\pi : S \times T \rightarrow A$
    - $\pi(s, t)$ action at state s with t states to go.
  - Stationary policy
    - $\pi : S \rightarrow A$
    - $\pi(s)$ action for state s (regardless of time)
  - Stochastic policy
    - $\pi(a|s)$ probability of choosing action a in state s
- **Goal**: find policy that leads to the highest expected accumulated reward

Planning horizon and Payoffs

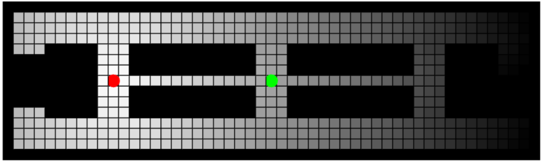
- cumulative reward $R_T = \mathbb{E}\left[\sum_{\tau=1}^{T} \gamma^{\tau} r_{t+\tau}\right]$
  - $T = 1$ greedy,
    - sub-optimal but simple and sometimes effective
  - $1 < T < \infty$, finite horizon
    - usually $\gamma = 1$, leads to non-stationary policies.
  - $T = \infty$ infinite-horizon
    - with $\gamma < 1$ leads to finite accumulated expected reward
    - $R_{\infty} \leq r_{max} + \gamma r_{max} + \gamma^2 r_{max} + \cdots = \frac{r_{max}}{1-\gamma}$
    - **stationary** policy

MDPs for mobile robotic systems, usual terminology

- State is typically robot pose $x = \{x, y, \theta\}$
- Actions is control commands u (e.g., $u = (v, \omega)$ for a differential drive robot)
- Transition function is the motion model $p(x'|x, u)$
- Value function depends on specific application usually represented with $r(x, u)$
  - for a standard motion planning task reward could be:
    - high reward at the goal
    - small negative penalty for every movement
    - high penalty for collisions

Values and Q-Values

- Value of a state x when following policy π: expected accumulated (discounted) reward when starting at x and following π everafter
  - $V_\pi(x) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | x_t = x\right]$
- Q-value (action value or quality function): value of taking action a in state s following policy π
  - $Q_\pi(x, u) = \gamma\left[r(x, u) + \int p(x'|u, x) v_\pi(x') dx'\right]$
  - Note: $V_\pi(x) = Q_\pi(x, \pi(x))$



Example of value function for $T = \infty$ assuming goal state is absorbing, source [PR]

Bellman equations for policy value

- value of the start state must equal the immediate reward of executing action u in state x, plus the (discounted) reward expected along the way.

$$V_\pi(x) = \gamma \left[r(x, \pi(x)) + \int p(x' | \pi(x), x) V_\pi(x') dx' \right]$$

- can be considered as a self-consistency condition

Optimal policy

- $\pi_*(x)$ is an optimal policy iff $V_{\pi_*}(x) \geq V_\pi(x) \forall\, x, \pi$
- $V_*(x) = \max_\pi V_\pi(x)$ expected utility starting in x and acting optimally everafter
- optimal action-value function $Q_*(x,u) = \max_\pi Q_\pi(x,u)$
- $V_*(x)$ must comply with the self-consistency condition dictated by the Bellman equation
- $V_*(x)$ is the optimal value hence the consistency condition can be written in a special form
- The value of a state under an optimal policy must equal the expected return for the best action from that state

$$V_*(x) = \max_u Q_*(x,u) = \gamma \max_u \left[r(x,u) + \int p(x'|u,x) V_*(x') dx' \right]$$

Solution approaches to MDPs

- most prominent solution approaches are
  - Value iteration, build a value function and the find optimal policy
  - Policy iteration, build directly the optimal policy
- Discussions made so far are valid for MDPs defined in **continuous** spaces
- Most solution methods discretize the state space and action space
- Can develop similar approaches based on sampling

Value iteration

- ◇ Turn the Bellman optimality equation into an "update rule", combining policy evaluation (computing the value v_π of a given policy) and policy improvement (making π greedy with respect to v_π).
- ◇ Bellman backup:

$$\hat{V}(x) = \gamma \max_u \left[r(x, u) + \int p(x'|u, x)\hat{V}(x')dx' \right]$$

  - Back up the value of every state to produce new value function estimates
- ◇ Discrete Bellman backup:

$$\hat{V}(x_i) = \gamma \max_u \left[r(x_i, u) + \sum_{j=1}^{N} p(x_j|u, x_i)\hat{V}(x_j) \right]$$

Discrete Value iteration Algorithm

Algorithm 1: Discrete Value Iteration

Data: $\{X, U, p(x'|u,x), r(x,u)\}$
Result: $\hat{V}$

for $i = 1$ **to** N **do**
    $\hat{V}(x_i) = r_{min}$;
end
while **not converged** **do**
    for $i = 1$ **to** N **do**
        $\hat{V}(x_i) = \gamma \max_u \left[r(x_i, u) + \sum_{j=1}^{N} p(x_j | u, x_i) \hat{V}(x_j) \right]$;
    end
end
return $\hat{V}$;

Discrete Value iteration, computing the policy

Algorithm 2: Policy computation for state x

Data: $\{x, \hat{V}\}$
Result: $\pi(x)$
return $\arg\max_u \left[r(x_i, u) + \sum_{j=1}^{N} p(x_j | u, x_i) \hat{V}(x_j) \right]$;

Pros guaranteed to converge to optimal policy, convergence rate is linear
Cons quadratic in number of states and linear in number of actions

Alternative approach, Policy iteration

Algorithm:

$\pi \leftarrow$ an arbitrary initial policy

repeat until no change in π

compute utilities given π (policy evaluation)

update π as if utilities were correct (policy improvement)

◇ policy evaluation compute utilities given a fixed π:

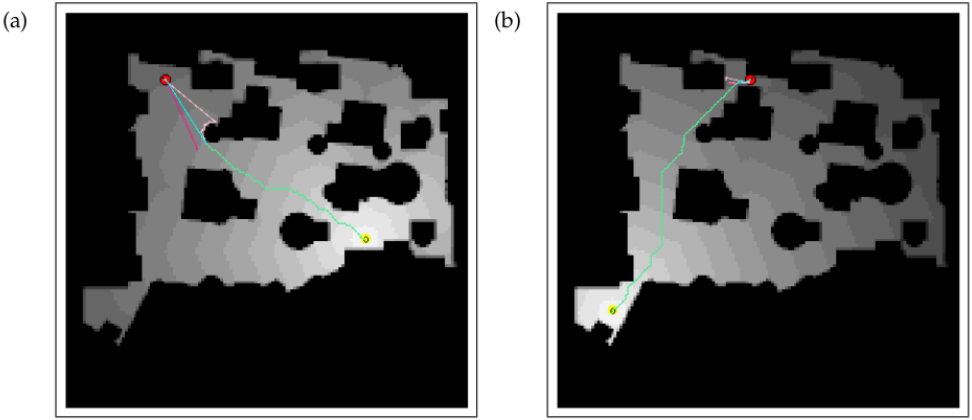
$$\hat{V}(x_i) = \gamma \left[r(x_i, u) + \sum_{j=1}^{N} p(x_j | \pi(x_i), x_i) \hat{V}(x_j) \right]$$

◇ policy improvement given the value of all state ($\hat{V}(x_i)$):

- greedily change the first action taken when in a state based on current value of states
- if the value of the state can be improved, the new action is adopted by the policy; thus, the performance of the policy is strictly **improved**.

Planning for mobile robots using MDPs, value iteration

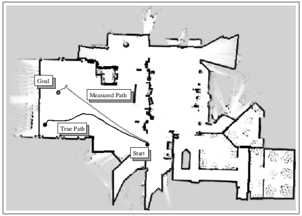
◇ Transition model is $p(x_{t+1}|x_t, u_t)$, state is $\hat{x}_t = \mathbb{E}\left[p(x_t|z_{1:t}, u_{1:t})\right]$

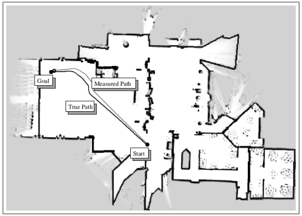


Value function and path for robot motion planning, brighter areas have higher values, source [PR]

Partially Observability

- ♢ In most realistic situation the state is **not** completely observable
  - robot does not know where it is and need to find this out
- ♢ If we do not know the state it makes no sense to talk about policy $\pi(x)$
- ♢ Typical solution: $\hat{x}_t = \mathbb{E}\left[p(x_t | z_{1:t}, u_{1:t})\right]$ work but ignores uncertainty on state estimation





Example of coastal navigation, courtesy of Nicholas Roy, source [PR]

MDPs vs. POMDPs

- MDPs

$$V_T(x) = \gamma \max_u \left[r(x,u) + \int p(x'|u,x) V_{T-1}(x') dx' \right]$$

where, $V_1(x) = \gamma max_u r(x,u)$

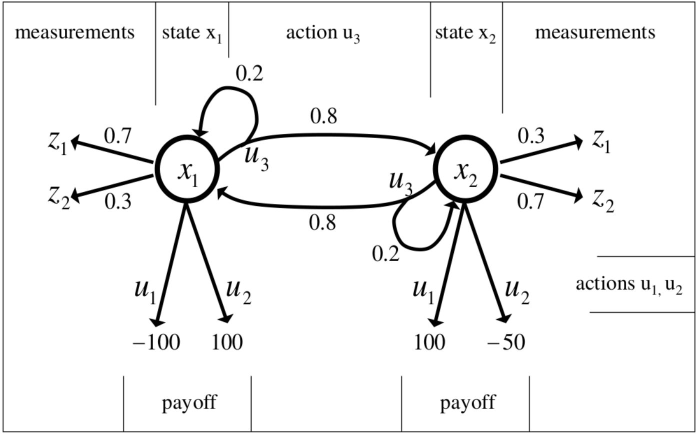
- In POMDPs the robot must maintain a **belief** on the state: $Bel(x_t)$
  - $Bel(x_t) = p(x_t|z_{1:t}, u_{1:t})$, we will indicate $Bel(x_t)$ with b
  - need an **observation model** $p(z_t|x_t)$
- POMDPs

$$V_T(b) = \gamma \max_u \left[r(b,u) + \int p(b'|u,b) V_{T-1}(b') db' \right]$$

$$\pi_T(b) = \arg\max_u \left[r(b,u) + \int p(b'|u,b) V_{T-1}(b') db' \right]$$

where $V_1(b) = \gamma max_u \mathbb{E}[r(x,u)]$

POMDP example



Example of a two state POMDP, source [PR]

Solving POMDPs, challenges

◇ Canonical solution method: Continuous state "belief MDP"

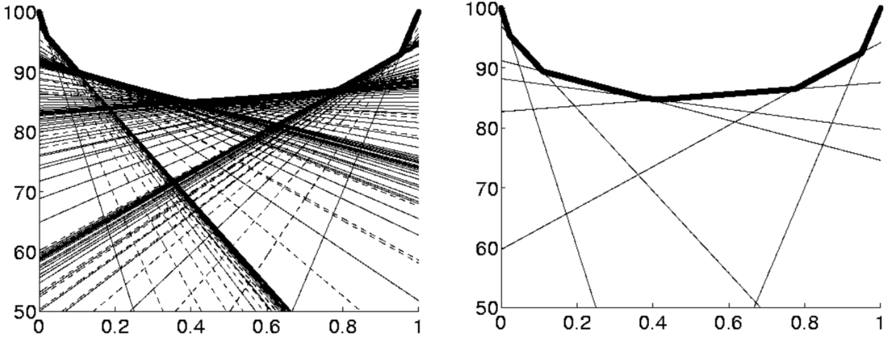
- Run value iteration on a state space of probability distributions
- find value and optimal action for every possible probability distribution
- will optimally address the trade-off between information gathering actions versus actions that affect the state of the world

◇ However, value iteration can not be carried out directly because we have an uncountable number of belief states

◇ For finite worlds with finite state, action, and measurement spaces and finite horizons, we can effectively represent the value functions by piecewise linear functions.

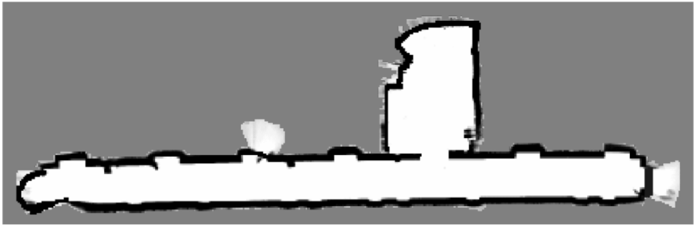
- Use linear functions (α-vectors) to represent the value function
- *alpha*-vectors are built by incorporating measurement and actions into the beliefs
- challenge: number of vectors grows significantly and is difficult to control

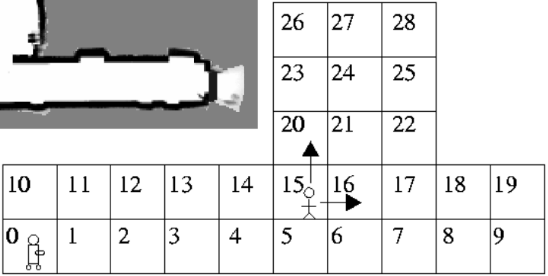
POMDP example



Linear functions for value iteration and PBVI for T=30 , source [PR]

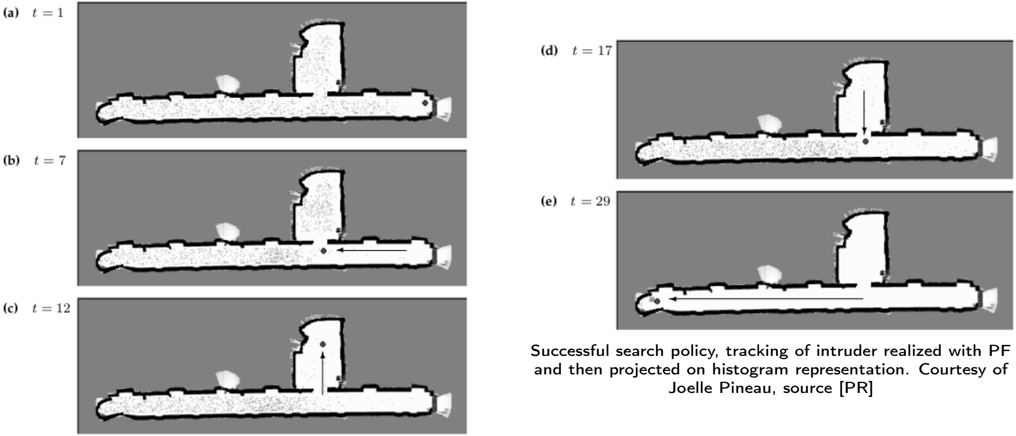
POMDP for intrusion detection, problem





Intrusion detection policy based on POMDPs. Robot is well localized, intruder position is unknown, courtesy of Joelle Pineau, source [PR]

POMDP for intrusion detection, example of solution



Successful search policy, tracking of intruder realized with PF and then projected on histogram representation. Courtesy of Joelle Pineau, source [PR]

POMCP an approximate methods for POMDPs

- Powerful approximate approach to solve large scale POMDPs
- Key insights:
  - Particle Filter to represent the belief
  - Monte Carlo Tree Search for action selection
  - Black box simulator $(s_{t+1}, o_{t+1}, r_{t+1}) \sim \mathcal{G}(s_t, a_t)$ to represent the POMDP dynamics
    - POMDP model does not need to be fully specified in closed form
  - Policy is never explicitly represented

MCTS, introduction

- ◇ incremental, asymmetric tree-search algorithm
- ◇ aim to expand nodes in promising areas of the search space
  - not complete, not optimal
  - scale extremely well
  - Usually works very well in practice
  - at the heart of several successful applications (particularly for games)

MCTS, overview

- Selection (or traversal)
  - Select next node to expand trading-off exploration and exploitation
  - Standard criteria: Upper Confidence Bound $UCB1(S_i) = \overline{V}_i + C \cdot \sqrt{\frac{\ln N}{n_i}}$, where N is number of visit of S_i **parent state** and n_i is number of visit of state S_i
- Expansion
  - Add further nodes to the tree
- Simulation (or rollout)
  - move down the tree using random action selection until we reach a terminal node
- Backpropagation
  - propagate value of terminal node up the tree branch

MCTS, algorithm for Selection and Expansion

Algorithm 3: Selection and Expansion phase

$Current = S_0$;
while $Current$ **is not leaf do**
  $Current$ = child that maximizes UCB1
end
if n_i **of current is 0 then**
  $Rollout(Current)$;
else
  for each available action from $Current$ add a new node to the tree;
  $Current$ = first new children;
  $Rollout(Current)$;
end

MCTS, rollout process

Algorithm 4: *Rollout*(S_i)

Data: S_i
Result: the value of a terminal node
while true **do**
  if S_i is terminal **then**
    return $Value(S_i)$;
  end
  $A_i = random(available\_actions(S_i))$;
  $S_i = simulate(A_i, S_i)$;
end

Worked example

MCTS discussion

- ◇ can be seen as a version of best-first search that focuses on promising part of the search space
- ◇ sample state transition, does not suffer from curse of dimensionality
- ◇ requires a black-box simulator rather than a full specification of the dynamics
- ◇ if exploration is controlled appropriately (e.g., UCB) it is guaranteed to converge to optimal policy
- ◇ anytime, computationally efficient and highly parallelizable

Partially Observable Monte Carlo Planning

- POMCP, use MCTS to break curse of dimensionality and curse of history
- POMCP builds, **online** a search tree of **histories**
- Nodes estimate values of a history using Monte-Carlo simulation
- For each simulation:
  - start state sampled from current belief state
  - state transitions and observations are sampled from a black-box simulator
- If belief state is correct **POMCP** converges to optimal policy for any finite horizon POMDP
- belief state approximated by set of sample states corresponding to the actual history
- The same set of Monte-Carlo simulations are used both to perform tree-search and to update the belief state

POMCP, illustration

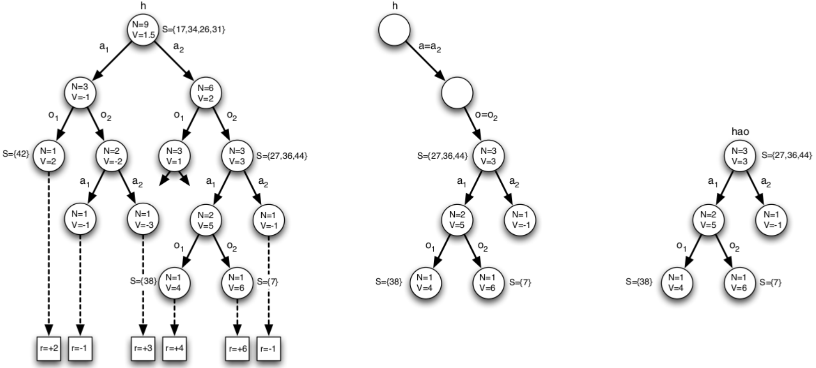


Illustration of POMCP in an environment with 2 actions, 2 observations and 50 states. Courtesy of Silver and Vennes

POMCP, tree search

- History $h = \{a_1, o_1, a_2, o_2, \dots\}$
- Node tree $T(h) = \langle N(h), V(h), B(h) \rangle$
  - $N(h)$ number of time history h was visited
  - $V(h)$ average **return** of all simulations starting from h
  - $B(h)$ particles that represent $\mathcal{B}(s,h) = p(s|h)$, belief state of s given history h
- selection proceeds by using

$$V^{\oplus}(ha) = V(ha) + c\sqrt{\frac{lnN(h)}{N(ha)}}$$

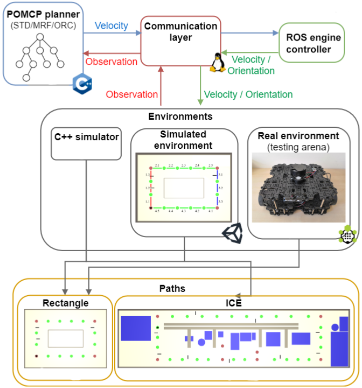
- rollout is performed using random action selection

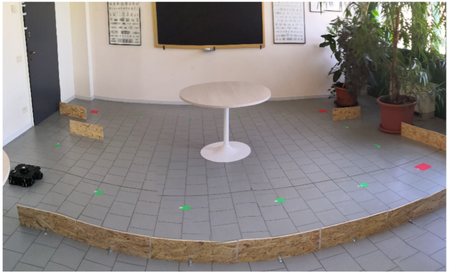
POMCP, belief update

- belief state is represented using and unweighted particle filter
  - unweighted PF is easier to implement and update with a black box simulator
- belief for history h_t uses K particles $B_t = B_t^i$, $\hat{\mathcal{B}}(s, h_t) = \frac{1}{K} \sum_{i=1}^{K} \delta_{sB_t^i}$
- particles are updated when a **real** action a_t is executed and a **real** observation o_t is obtained
  - a state s is sampled from $\hat{\mathcal{B}}(s, h_t)$ selecting a random particle from B_t
  - the particles is passed to the black-box simulator to get a nes state s' and an observation o' $(s', o', r) \sim \mathcal{G}(s, a_t)$
  - if sampled observation o' matches the real observation $o = o_t$ then a new particle s' is added to B_{t+1}
  - the process is repeated until K particles are added
  - $\lim_{k\to\infty} \hat{\mathcal{B}}(s, h_t) = \mathcal{B}(s, h_t)$ but particle deprivation can happen for large t

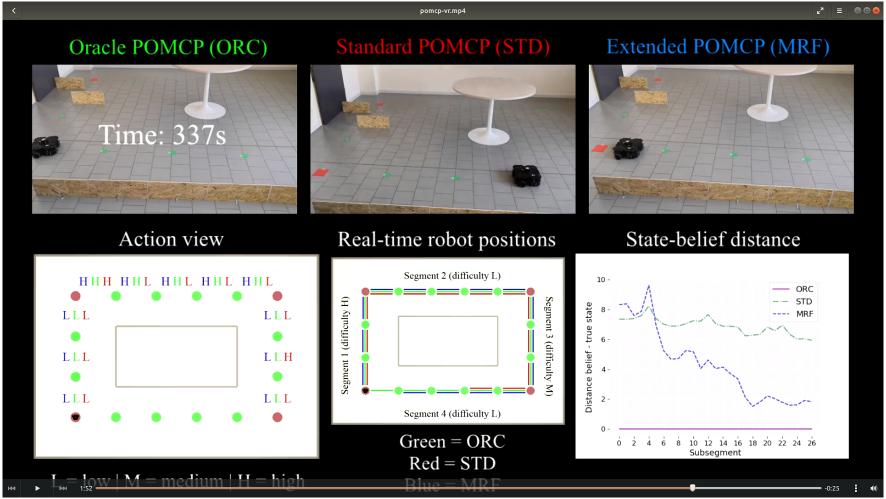
POMCP example

- Velocity regulation for mobile robots, difficulty of segment is partially observable
- Improve POMCP performance by providing prior-knowledge
  - similarity between segments





POMCP in action



Further Readings

◇ Browne et al., A Survey of Monte Carlo Tree Search Methods, IEEE Trans. on Computational Intelligence and AI in Games, 2021

◇ Silver and Vennes, Monte-Carlo Planning in Large POMDPs, NeurIPS, 2010

◇ Pineau, et al., Anytime point-based approximations for large POMDPs. Journal of Artificial Intelligence Research, 2006

◇ Auer et al., Finite-time analysis of the multi-armed bandit problem. Machine Learning, 2002