UNIVERSITY OF TRIESTE

Department of Mathematics, Informatics and Geoscience

Master's Degree in

Data Science & Scientific Computing

Leveraging POMDPs for Adaptive Decision-Making in Robotic-Assisted Surgery

March 11, 2025

Candidate	Supervisor
Matteo Nunziante	**Prof. Antonio Celani**

A.Y. 2023/2024

Abstract

Thesis Outline and Summary

Contents

List of Figures

List of Tables

Acronyms

DL Deep Learning.

DP Dynamic Programming.

DRL Deep Reinforcement Learning.

GPI Generalized Policy Iteration.

MC Monte Carlo.

MDP Markov Decision Process.

PBVI Point Based Value Iteration.

POMDP Partially Observable Markov Decision Process.

PPO Proximal Policy Optimization.

RL Reinforcement Learning.

SGD Stochastic Gradient Descent.

TD Temporal Difference.

TRPO Trust Region Policy Optimization.

Chapter 1

Introduction

The advancement of robotic systems has significantly improved minimally invasive surgery, allowing for smaller incisions, reduced blood loss, and faster recovery times, while enabling complex procedures that require exceptional dexterity and visualization capabilities. The marriage of robotic surgery and AI promises further innovations which could redefine healthcare access and delivery on a global scale. Among these, remote surgical capabilities, enhanced imaging technologies, real-time decision support, personalized treatments and autonomous surgical systems are some of the most promising applications of AI in healthcare.
As robotic-assisted surgical techniques gain traction - increasingly relevant as the demand for minimally invasive approaches grows - the integration of Reinforcement Learning (RL) algorithms into these workflows has become notable for its potential to transform patient outcomes enhanceing once more surgical precision.
The development of RL techniques in this field leverages the ability of algorithms to learn from complex interactions within dynamic environments by training robotic systems to optimize intricate tasks through feedback mechanisms, RL not only enhances the operational efficiency of surgical robots but also contributes to more informed preoperative planning through predictive analytics. These advancements allow for tailored risk assessments that improve decision-making in patient care.
Despite its promise, integration of such techniques into minimally surgery faces challenges, including the need for robust regulatory frameworks and ongoing ethical scrutiny. Moreover manipulating and navigating in deformable environments, such as soft tissues, remains a major challenge due to their dynamic and unpredictable nature. Traditional control algorithms struggle to adapt to these uncertainties, leading to suboptimal performance and potential risks during surgical procedures.
Overall, the ongoing research and development in RL for minimally surgery exemplify a transformative potential in the healthcare landscape, with opportunities for improving surgical techniques, outcomes, and training methods while navigating

the complexities of integrating AI into human-centered care.

1.1 Problem Statement

RL has emerged as a promising approach, however, applying RL to robotic surgery introduces additional challenges, since surgical environments are inherently partially observable, as direct and complete state information is often unavailable due to occlusions, sensor noise, and real-time constraints. Existing RL techniques often assume full observability, limiting their effectiveness in real-world surgical applications.

This thesis aims to address these challenges by developing RL-based strategies for minimally invasive robotic surgery, specifically focused on deformable environment navigation and manipulation under partial observability. The research will explore Partially Observable Markov Decision Process (POMDP)-based RL frameworks, integrating advanced perception models, uncertainty-aware decision-making, and efficient policy learning methods. The goal is to enhance robotic surgical capabilities by improving adaptability, robustness, and precision in deformable tissue interactions, ultimately contributing to safer and more reliable autonomous surgical systems.

1.2 Related Work

While robotic systems have drastically improved surgeon skills, they still only replicate motions performed by surgeons in a console, lacking the ability to adapt to unexpected intraoperative changes autonomously.

To address these limitations, integration of Computer vision, Deep Learning (DL) and RL has been explored to possibly enhance the autonomy and adaptability of robotic systems in surgical environments.

Computer vision techniques such as classification, detection and segmentation, have been used to recognize anatomical structures and surgical instruments to provide support and valuable intraoperative information to the surgeon and eventually to autonomous robotic systems (Amparore et al., 2022; Piana et al., 2024).

RL algorithms have been tested in simulation based training environments to directly control robotic systems for surgical tasks. The need for simulation is dual, it provides a safe environment and it allows to circumvent the high costs of real-world needs amplifyed by sample inefficience of RL. The policy learned in simulation can then be transferred to the real-world setting, where it can be further fine-tuned to account for the discrepancies between the two environments (Ou and Tavakoli, 2023).

A large number of recent research focuses on the automation of surgical sub-tasks, such as needle manipulation (Bendikas et al., 2023), suturing (Varier et al., 2020), cutting (Shahkoo and Abin, 2023), vessel manipulation (Dharmarajan et al., 2023), blood suction (Ou et al., 2024), tissue retraction and deformation (Scheikl et al., 2023b). Imitation Learning and expert demonstrations have shown to be effective in training robotic systems for surgical tasks (Kim et al., 2024).

Part I

Theoretical Background

Chapter 2

Markov Decision Processes

RL is a subfield of machine learning focused on the problem of learning optimal behaviors: agents repeatedly interact with the environment, balancing exploration (trying new strategies) and exploitation (using known strategies) to maximize success rate through constant feedback.
For instance, consider a baby learning to walk: it repeatedly attempts to stand and take steps, getting positive feedback for each successful movement and negative feedback from falls. Over time, it refines its attempts to walk more effectively.
Differently from supervised learning, where the agent is trained on labeled data, and from unsupervised learning, where the agent must find patterns in unlabeled data, RL makes use of rewards and punishments, making it suitable for problems where the agent (be it a baby or a Large Language Model) must learn through trial and error, creating its own training data through interaction.
In the next sections, we will introduce the foundational concepts of RL, mathematical frameworks, and algorithms that enable agents to learn optimal behaviors for a wide range of tasks.
The following analysis of Markov Decision Processs (MDPs) is mostly based on Wiering and Van Otterlo (2012) and Sutton and Barto (2018).

RL leverages MDPs to model interactions between an agent and its environment in terms of states, actions, and rewards. This framework captures in a simple way features of the problem such as cause-and-effect, uncertainty, and explicit objectives. Although general MDPs may have infinite, possibly uncountable, state and action spaces, we limit the discussion to finite-state and finite-action problems.
A MDP is a model for sequential decision making in a probabilistic environment; it requires the following components:

States The set of environmental states S is defined as the finite set $\{s_1, ..., s_N\}$ where the size of the state space is $|S| = N$. A state is a unique characterization of all that is important (at a given time) of the problem that is modeled. For example, a chess game state could be the position of all the pieces on the board.

Actions The set of actions A is defined as the finite set $\{a_1, ..., a_K\}$ where the size of the action space is $|A| = K$. Actions can be used to control the system state. The set of actions that can be applied in some particular state $s \in S$, is denoted $A(s)$, where $A(s) \subset A$. In some systems, not all actions can be applied in every state, but in general, we will assume that $A(s) = A$ for all $s \in S$.

Transition Function By applying action $a \in A$ in a state $s \in S$, the system makes a transition from s to a new state $s' \in S$, based on a probability distribution over the set of possible transitions. The transition function $\mathcal{T}$ is defined as

$$\mathcal{T} : S \times A \times S \to [0, 1]$$

which represents the probability of ending up in state s' after doing action a in state s is denoted $\mathcal{T}(s, a, s')$. It is required that for all actions a, and all states s and s', $\mathcal{T}(s, a, s') \geq 0$ and for all states s and actions a,

$$\sum_{s' \in S} \mathcal{T}(s, a, s') = 1$$

such that $\mathcal{T}$ defines a proper probability distribution over possible next states.
The sequential nature of the framework is captured by that of *Markov chain*: a Markovian Stochastic Process. Stochastic Processes arise in many problems from the natural sciences in which one has to keep track of a value observed at time t. The process is Markovian if the result of an action does not depend on the previous actions and history of visited states, but only depends on the current state.
Going back to our notation, given a trajectory of the form

$$s_0, a_0, ..., s_t, a_t, s_{t+1}$$

the Markov property is defined as follows:

$$P(s_{t+1}|s_t, a_t, s_{t_1}, a_{t-1}, .., s_0, a_0) = P(s_{t+1}|s_t, a_t) = \mathcal{T}(s_t, a_t, s_{t+1})$$

The idea of Markovian dynamics is that the current state s gives enough information to make an optimal decision; it is not important which states and actions preceded s.

Reward Function Rewards guide learning by signaling success. A higher reward indicates a more desirable outcome. In the chess example, capturing a piece might give a large reward, encouraging the agent to repeat the actions that led to that success. The reward function $\mathcal{R}$ is usually defined as:

$$\mathcal{R} : S \times A \times S \to [0, 1]$$

which represents the reward received after transitioning from state s to s' by taking action a. Alternatively, the reward function can be defined as $\mathcal{R} : S \times A \to \mathbb{R}$, where the reward depends only on the current state and action.

With the above components, we can define a MDP as in Figure 2.1.

Definition 2.0.1. A *Markov Decision Process* is a tuple $M = (S, A, \mathcal{T}, \mathcal{R})$ where:

- S is a finite set of states
- A is a finite set of actions
- $\mathcal{T}$ is the transition function $\mathcal{T} : S \times A \times S \to [0, 1]$
- $\mathcal{R}$ is the reward function $\mathcal{R} : S \times A \times S \to \mathbb{R}$

The transition and reward functions together define the *Model* of the MDP.

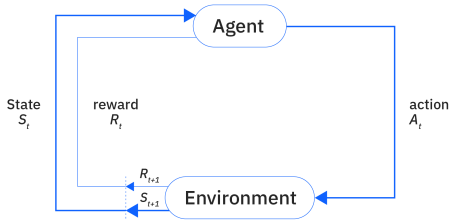


Figure 2.1: Markov Decision Process

A unified notion of transitions and rewards is given by what's called the dynamic of the MDP:

$$p : S \times R \times S \times A \to [0, 1]$$

where

$$p(s', r|s, a) = Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$$

p is the probability associated with transitioning from state s choosing action a and ending up in state s' and getting reward r. Note that the following identities hold:

$$\sum_{s' \in S} \sum_{r \in R} p\left(s', r \mid s, a\right) = 1, \text{ for all } s \in S, a \in A(s).$$

$$\mathcal{T}(s,a,s') = p\left(s' \mid s,a\right) \doteq \Pr\left\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\right\} = \sum_{r \in R} p\left(s', r \mid s, a\right)$$

and for $r : S \times A \times S \to \mathbb{R}$:

$$\mathcal{R}(s,a,s') = r\left(s,a,s'\right) \doteq \mathbb{E}\left[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'\right] = \sum_{r \in \mathcal{R}} r \frac{p\left(s', r \mid s, a\right)}{p\left(s' \mid s, a\right)}$$

These formulations are useful for the analysis of MDPs and the development of algorithms. In such context, the goal of the agent is to learn an optimal behavior; let's define behaviors and optimality.

2.1 Policy and Value Function

Policy The formalization of the naive concept of behavior is that of policy. A policy is a function that maps states into actions $\pi : S \to A$. It can be deterministic or stochastic; in the latter case, it is represented as a probability distribution over actions given states.

$$\pi(a|s) = Pr\{A_t = a | S_t = s\}$$

it holds that for all states $s \in S$, $\pi(a|s) \geq 0$ and $\sum_{a \in A} \pi(a|s) = 1$. Another distinction is made between stationary and nonstationary policies. A stationary policy does not change over time; it is a function of the current state only, while a nonstationary policy can be imagined as a sequence of policies indexed by time. In the *finite-horizon* model, a MDP in which the agent has a limited amount of time to achieve its goal, the optimal policy is typically non-stationary: the way an agent chooses its actions on the last step of its life is generally going to be very different from the way it chooses them when it has a long life ahead of it. In the *infinite-horizon* discounted model, the agent always has a constant expected amount of time remaining, so there is no reason to change action strategies: there is a stationary optimal policy. Such policies represent the strategy of the agent; it is the way the agent interacts with the environment by iteratively selecting actions based on the current state of the system and thereby influencing the next state and reward received.
The main goal of RL is to find "good" policies. It is intuitive to think that a policy is good if it produces high rewards.

Value Function The concept of value function is useful to evaluate the quality of a policy. The value function of a state s under a policy π, denoted $V^\pi(s)$, is the expected return when starting in s and following π thereafter. We can define the state-value function V^π for policy π in the *discounted infinite model* by

$$V^{\pi}(s) \doteq \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right], \text{ for all } s \in S,$$

where $\mathbb{E}_{\pi}[\cdot]$ denotes the expected value of a random variable given that the agent follows policy π, and t is any time step, $\gamma \in [0, 1]$ is the discount factor, it is used to weigh the importance of immediate rewards and to make sure that the sum converges even in such infinite horizon problems. The value function satisfies a recursive relationship known as the *Bellman equation* (2.1) for V^{π}, a crucial concept in the development of learning algorithms.

$$\begin{aligned}
V^{\pi}(s) &= E\left[\sum_{k} \gamma^k R_{t+k+1} + |S_t = s\right] \\
&= E[G_t | S_t = s] \\
&= E[R_{t+1} + \gamma G_{t+1} | S_t = s] \\
&= \sum_{s'} \sum_{r} \sum_{g_{t+1}} \sum_{a} p(s', r, g_{t+1}, a|s)(r + \gamma g_{t+1}) \\
&= \sum_{a} \pi(a|s) \sum_{s'} \sum_{r} \sum_{g_{t+1}} p(s', r, g_{t+1}|a, s)(r + \gamma g_{t+1})
\end{aligned}$$

since $p(g_{t+1}|s', r, a, s) = p(g_{t+1}|s')$ by MDP assumption

$$\begin{aligned}
&= \sum_{a} \pi(a|s) \sum_{s'} \sum_{r} \sum_{g_{t+1}} p(s', r|a, s) p(g_{t+1}|s', r, a, s)(r + \gamma g_{t+1}) \\
&= \sum_{a} \pi(a|s) \sum_{s'} \sum_{r} p(s', r|a, s) \sum_{g_{t+1}} p(g_{t+1}|s')(r + \gamma g_{t+1}) \\
&= \sum_{a} \pi(a|s) \sum_{s'} \sum_{r} p(s', r|a, s)(r + \gamma \sum_{g_{t+1}} p(g_{t+1}|s') g_{t+1}) \\
&= \sum_{a} \pi(a|s) \sum_{s'} \sum_{r} p(s', r|a, s) \left(r + \gamma V^{\pi}(s')\right) \qquad (2.1)
\end{aligned}$$

Similarly one can define the state-action value function, giving back the value of taking action a in state s under a policy π, denoted $Q^{\pi}(s, a) : S \times A \to \mathbb{R}$, as the expected return starting from s, taking the action a, and following policy π thereafter:

$$\begin{aligned}
Q^{\pi}(s, a) &\doteq \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right] \\
&= \mathbb{E}_{\pi}\left[R_t + \gamma V^{\pi}(S_{t+1}) | S_t = s, A_t = a\right] \qquad (2.2)
\end{aligned}$$

it follows naturally that

$$V^{\pi}(s) = \max_{a \in A(s)} Q^{\pi}(s, a)$$

2.2 Bellman Optimality Equation

Introducing a partial ordering on the space of policies such that
$\pi \geq \pi'$ if and only if $V^{\pi}(s) \geq V^{\pi'}(s)$ for all $s \in S$ allows to compare different policies, and to define optimal ones as the class π^*

$$\pi^* \text{ such that } \pi^* \geq \pi \text{ for all } \pi$$

MDPs (but not POMDPs, we'll introduce them in the next chapter), always have[1] is a deterministic, stationary policy that maximizes the value of every state. Bellman (1952)
If we know this optimal policy, then we get the optimal value function $V^*(s_t)$:

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

Similarly, the optimal state-action value function is obtained under the optimal policy:

$$Q^*\left[s_t, a_t\right] = \max_{\pi} Q^{\pi}\left(s_t, a_t\right)$$

Vice versa, knowing the optimal state-action value function allows to derive the optimal policy by choosing the action a_t with the highest value

$$\pi^*\left(s_t\right) = \operatorname*{argmax}_{a}\left[Q^*\left(s_t, a\right)\right]$$

The optimal value function satisfies the *Bellman Optimality Equation*, given by:

$$\begin{aligned} V^*(s) &= \max_{a \in A(s)} Q^{\pi_*}(s,a) \\ &= \max_{a} \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\ &= \max_{a} \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\ &= \max_{a} \mathbb{E}[R_{t+1} + \gamma V^*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_{a} \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V^*(s')\right]. \end{aligned} \tag{2.3}$$

An analogous result holds for the optimal state-action value function:

$$Q^*(s,a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

[1] This is true in discrete state space MDPs when $\gamma < 1$. Existence of optimal policies is not guaranteed in discrete problems for $\gamma = 1$. Generally, in a continuous setting, it is required $\gamma \in (0,1)$, and costs (negative rewards) to be bounded $|c(s,a)| < M$ for all s and a

$$= \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma \max_{a'} Q^*(s', a') \right]. \tag{2.4}$$

$$= \sum_{s',r} p(s', r|s, a) \left[r + \gamma V^*(s') \right] \tag{2.5}$$

These equations must be satisfied and can, in principle, be solved for the optimal value functions, from which an optimal policy can be determined with relative ease. In practice this is hardly the case due to computational limitations and the optimal value functions are usually approximated.

2.3 Solving MDPs

Solving a given MDP means computing an optimal policy. The most crucial distinction in available techniques is that between *model-based* and *model-free* algorithms. Model-based methods use the MDP structure explicitly and find the best policy from the transition and reward functions. If these are known, this is a straightforward optimization problem that can be tackled using dynamic programming. If they are unknown, they must first be estimated from observed trajectories. The main advantage to having a model is that it allows the agent to plan, seeing beforehand what would happen in every possible trajectory, and then choosing the best strategy. A particularly famous example of this approach is AlphaZero by Silver et al. (2017). The presence of a model can result in a substantial improvement in sample efficiency over methods that don't exploit it.
The main downside is that a ground-truth model of the environment is usually not available to the agent. If an agent wants to use a model in this case, it has to learn the model purely from experience, which creates several challenges.
Before diving into modern solution methods - based on Deep Reinforcement Learning (DRL) (fig. 2.2) and relying on Neural Networks to approximate the value function, the policy, or both - we will introduce some of the most common and foundational algorithms used to solve MDPs that are the building blocks of more advanced techniques.

2.4 Dynamic Programming

Dynamic Programming (DP) is a class of algorithms of little practical use, nonetheless of great theoretical importance in the field of model-based RL. The assumption of the availability of the model is crucial for the application of these methods as the finitenees of the state and action spaces, though continous spaces can be discretized, exact solutions can be rarely found. Most RL methods work in two main steps: policy evaluation and policy improvement.

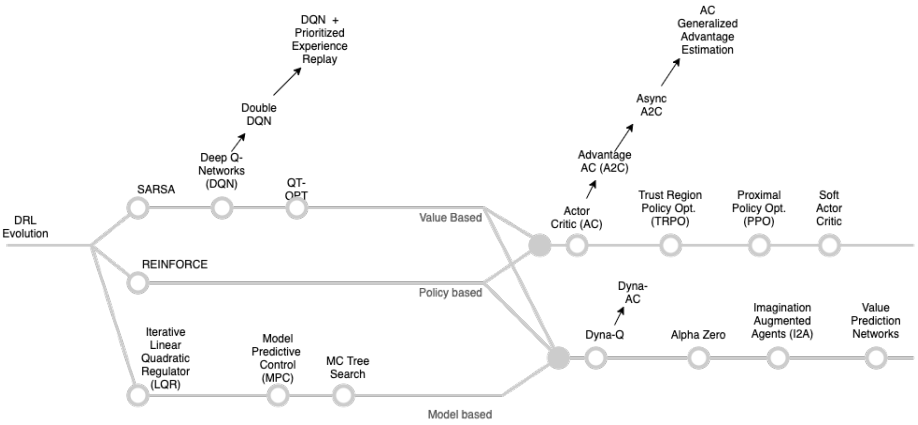


Figure 2.2: Evolution of Deep Reinforcement Learning Algorithms

Policy Evaluation policy evaluation or prediction is the task of determining the value function for a given policy. If the environment's dynamics are known, eq. (2.1) represents a linear system in the $|S|$ unknowns $V^\pi(s)$. It can be solved exactly but the computational burden of exact solutions, makes iterative methods preferred. The most famous of these, known as *Iterative Policy Evaluation*, is based on the Bellman Equation:

$$V^{k+1}(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|a, s)(r + \gamma V^k(s'))$$

where $V^k(s)$ is the value function at iteration k. The algorithm is guaranteed to converge to the true value function V^π as $k \to \infty$ under the assumption that $\gamma < 1$ as a consequence of the contraction property of the Bellman operator. See Theorem 2.4.3 for a similar argument proof.

Policy Improvement The purpose of calculating the value function for a policy is to identify better policies. To determine if a policy can be improved, we compare the value of taking a different action a in state s with the current policy. This is done using the state-action value function $Q^\pi(s, a)$: If $Q^\pi(s, a) > V^\pi(s)$, choosing action a in s is more advantageous than following π, leading to an improved policy π'.

$$\begin{aligned} \pi'(s) &\doteq arg\max_a Q^\pi(s, a) \\ &= \mathbb{E}[R_{t+1} + \gamma V^\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= arg\max_a \sum_{s',r} p(s', r\,|s, a)\Big[r + \gamma V^\pi(s')\Big] \end{aligned} \tag{2.6}$$

This is motivated by the following theorem:

Theorem 2.4.1. *(Policy Improvement Theorem) Let π and π' be any pair of deterministic policies such that, for all $s \in S$*

$$Q^{\pi}(s, \pi'(s)) \geq V^{\pi}(s)$$

Then the policy $\pi' \geq \pi$. That is, it must obtain greater or equal expected return from all states $s \in S$:

$$V^{\pi'}(s) \geq V^{\pi}(s)$$

If there is strict inequality at any state, π' is superior to π.

Proof.

$$\begin{aligned} V^{\pi}(s) &\leq Q^{\pi}(s, \pi'(s)) \\ &= \mathbb{E}\left[R_{t+1} + \gamma V^{\pi}(S_{t+1}) | S_t = s\right] \\ &\leq \mathbb{E}\left[R_{t+1} + \gamma Q^{\pi}(S_{t+1}, \pi'(S_{t+1})) | S_t = s\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 V^{\pi}(S_{t+1}) | S_t = s\right] \\ &\leq \dots \\ &= V^{\pi'}(s) \end{aligned}$$

□

Policy improvement creates a new policy that enhances an initial policy by adopting a greedy approach based on the value function.

2.4.1 Policy Iteration

Policy Iteration allows to estimate an optimal policy in finite time. It is the combination of Policy Evaluation and Policy Improvement: once a policy π_t, has been improved using V_{t-1} to yield a better policy π_{t+1}, we can then compute V_{t+1} and start again.

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \dots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*,$$

By previous results we can thus obtain a sequence of monotonically improving policies and value functions unless we reach the optimal policy. To show convergence to the optimal policy, along with monotone improvement, we need to show that if there is no improvement in the value function at any state, then we are at optimality. Consider k such that $V^{\pi_{k+1}}(s) = V^{\pi_k}(s), \forall s \in S$. We can show using eq. (2.6) that such V^{π_k} satisfies the Bellman Optimality equation 2.3, and hence $V^{\pi_k} = V^*$.

The algorithm is summarized in 1.

Algorithm 1 Policy Iteration

Require: A finite MDP with states S, actions A, transition probabilities T, rewards R, and discount factor $\gamma \in [0, 1]$

Ensure: Optimal policy π^*

Initialize an arbitrary policy π

repeat

Policy Evaluation: Compute the state-value function V^π

Solve $V^\pi(s) = \sum_{a \in A} \pi(a|s) \sum_{s' \in S} \mathcal{T}(s'|s, a) \left[\mathcal{R}(s, a, s') + \gamma V^\pi(s')\right]$ for all $s \in S$

Policy Improvement: Compute a new policy π'

for all $s \in S$ **do**

$\pi'(s) \leftarrow \arg\max_{a \in A} \sum_{s' \in S} \mathcal{T}(s'|s, a) \left[\mathcal{R}(s, a, s') + \gamma V^\pi(s')\right]$

end for

until $\pi' = \pi$ ▷ Stop when policy converges

return π^*

2.4.2 Value Iteration

The main drawback of Policy Iteration is the need to perform the time consuming policy evaluation step at each iteration. Value Iteration is a simpler alternative that works by truncating the policy evaluation step without loosing convergence guarantees, truncating it after a single sweep, results in the following update rule for the value function:

$$\begin{aligned} V_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma V_k(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma V_k(s')\right], \end{aligned}$$

or in the equivalent form $V_{t+1} = BV_t$ where B is the *Bellman Optimality Operator* defined as:

$$(B^*V)(s) = \max_a \sum_{s', r} p(s', r|s, a) \left[r + \gamma V(s')\right] \tag{2.7}$$

The foundamental theoretical result is that the sequence of value functions $\{V_k\}$ converges to the optimal value function V^* as $k \to \infty$.

Lemma 2.4.2. The Bellman Optimality operator B^* is a contraction mapping with respect to the supremum norm.

Proof. Let V, W be any two value functions. Then, for all $s \in S$:

$$\begin{aligned}
|(B^*V)(s) - (B^*W)(s)| &= \left| \max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma V(s') \right] - \max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma W(s') \right] \right| \\
&\leq \max_a \left| \sum_{s',r} p(s', r|s, a) \left[r + \gamma V(s') \right] - \sum_{s',r} p(s', r|s, a) \left[r + \gamma W(s') \right] \right| \\
&\leq \max_a \sum_{s',r} p(s', r|s, a) \left| \left[r + \gamma V(s') \right] - \left[r + \gamma W(s') \right] \right| \\
&\leq \max_a \sum_{s'} p(s'|s, a) \left| \gamma V(s') - \gamma W(s') \right| \\
&\leq \gamma \max_a \sum_{s'} p(s'|s, a) \left| V(s') - W(s') \right| \\
&\leq \gamma \max_a \left\| V - W \right\|_\infty
\end{aligned}$$

where $\left\| V - W \right\|_\infty = \max_{s \in S} |V(s) - W(s)|$ is the supremum norm. □

Theorem 2.4.3. *(Value Iteration Convergence) The sequence of value functions $\{V_k\}$ generated by the value iteration algorithm converges to the optimal value function V^* as $k \to \infty$.*

Proof. The result is direct consequence of the contraction property of the Bellman Optimality operator and the Banach Fixed-Point Theorem.

$$\begin{aligned}
\left\| V_{k+1} - V^* \right\|_\infty &= \left\| B^* V_k - B^* V^* \right\|_\infty \\
&\leq \gamma \left\| V_k - V^* \right\|_\infty
\end{aligned}$$

By induction, we have that $\left\| V_k - V^* \right\|_\infty \leq \gamma^k \left\| V_0 - V^* \right\|_\infty$. Since $\gamma \in [0, 1)$, the sequence $\{V_k\}$ converges to V^*.

□

The algorithm is summarized in 2.

Algorithm 2 Value Iteration

Require: A finite MDP with states S, actions A, transition probabilities T, rewards R, discount factor $\gamma \in [0, 1]$, and a small threshold $\theta > 0$

Ensure: Optimal policy π^*

Initialize value function $V(s) \leftarrow 0$ for all $s \in S$

repeat

$\Delta \leftarrow 0$

for all $s \in S$ **do**

$v \leftarrow V(s)$

$V(s) \leftarrow \max_{a \in A} \sum_{s' \in S} \mathcal{T}(s, a, s') \left[\mathcal{R}(s, a, s') + \gamma V(s')\right]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

end for

until $\Delta < \theta$ ▷ Stop when value function converges

Extract Policy:

for all $s \in S$ **do**

$\pi^*(s) \leftarrow \arg\max_{a \in A} \sum_{s' \in S} \mathcal{T}(s, a, s') \left[\mathcal{R}(s, a, s') + \gamma V(s')\right]$

end for

return π^*

2.4.3 Generalized Policy Iteration

A very important underlying mechanism, common to most methods, is the so-called Generalized Policy Iteration (GPI) principle. It consists of two interacting processes.

First the policy evaluation step estimates the utility of the current policy, computing V^π, directly through the use of the model (if available) or iteratively by sampling trajectories interacting with the environment. Second the policy improvement step computes an improved policy from the current one using the information in V.

Both the evaluation and the improvement steps can be implemented and interleaved in several distinct ways. The underliyng mechanism is that there is a policy that drives value learning, i.e. it determines the value function, but in turn there is a value function that can be used by the policy to select good actions.

2.5 Monte Carlo Methods

Learning methods do not necessarily need to rely on the model of the environment. Model free techniques are based on the idea of learning from actual or simulated[2] experience, in the form of sample sequence of states, actions and rewards. Following

[2] Although a model is needed for simulated experience, a sample model is sufficient and is way easier to obtain than a full probabilistic specification of the environment dynamics

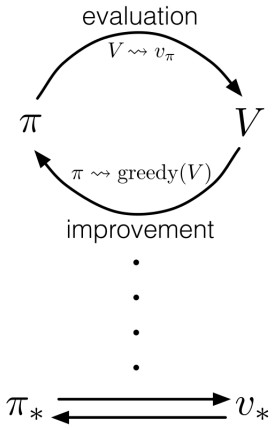


Figure 2.3: Generalized Policy Iteration

the GPI framework we briefly explore the main ideas behind Monte Carlo (MC) Methods.

Policy Evaluation The most intuitive strtegy for estimating the value function without a model is to average rewards after visiting a state. In particular, if we had to estimate $V_\pi(s)$, given a set of episodes obtained by following π after s, since s may be visited multiple times in the same episode; denoting the first time it is visited in an episode the *first visit* to s, two options are available:

- First-visit MC method estimates V(s) as the average of the returns following first visits to s.
- Every-visit MC method averages the returns following all visits to s.

Convergence of both methods is guaranteed by the law of large numbers.

Policy improvement It is worth noting that the absence of the model, state values alone are not sufficient to determine a greedy policy w.r.t. V. For control tasks MC methods are therefore used to estimate the action-value function $Q^\pi(s, a)$. The concept of visits is redefined to apply to state-action pairs, rather than states alone: a state-action pair (s, a) is considered visited if action a was selected while in state s

Monte Carlo Control In policy evaluation, we need to be cautious and ensure that all states will be visited, otherwise convergence is not guaranteed. To accomplish this we can generate the episodes with *exploring starts*, that is, all episodes begin with state-action pairs randomly selected to cover all possibilities. The second underlying assumption we made is that policy evaluation could be done over an

infinite number of episodes. Under theses assumptions, the value function converges to the true value function by *Policy Improvement Theorem* 2.4.1.
Both assumptions can actually be relaxed by giving up on deterministic policies, only search over ϵ-soft policies 2.8, and by allowing the value function to be updated after each episode and not when convergence is met.

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|} & \text{if } a = \text{argmax}_{a'} Q^{\pi}(s, a') \\ \frac{\epsilon}{|A|} & \text{otherwise} \end{cases} \tag{2.8}$$

Algorithm 3 Monte Carlo Control with Exploring Starts

Require: A finite MDP with states S, actions A, rewards R, discount factor $\gamma \in [0, 1]$, and an exploring-starts assumption
Ensure: Optimal policy π^*
  Initialize arbitrary action-value function $Q(s, a)$ for all $s \in S$, $a \in A$
  Initialize returns list $\text{Returns}(s, a) \leftarrow \emptyset$ for all s, a
  Initialize policy π arbitrarily
  repeat
    Generate an episode: $(s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_T)$ following π
    $G \leftarrow 0$
    for $t = T - 1$ to 0 **do** ▷ Process the episode in reverse
      $G \leftarrow \gamma G + r_{t+1}$
      if (s_t, a_t) is the first occurrence in the episode **then**
        Append G to $\text{Returns}(s_t, a_t)$
        $Q(s_t, a_t) \leftarrow \frac{1}{|\text{Returns}(s_t, a_t)|} \sum \text{Returns}(s_t, a_t)$
        Update policy: $\pi(s_t) \leftarrow \arg\max_{a \in A} Q(s_t, a)$
      end if
    end for
  until Q-values converge
  return π^*

2.6 Temporal Difference Learning

Combining the ideas of DP and MC methods, Temporal Difference (TD) learning is a model-free method that learns from experience (like MC) but updates estimates based on other learned estimates (like DP), a technique known as bootstrapping.
TD prediction methods update the value function estimates based on the difference between the current estimate and a target value, which is a sum of the reward and the value of the next state. The simplest form of TD prediction is the TD(0) algorithm, also known as the one-step TD algorithm. The update rule for the value function is given by:

$$V(S_t) \leftarrow V(S_t) + \alpha \Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \Big]$$

Convergence results for TD(0) is found in Sutton (1988), For any fixed policy π, TD(0) has been proved to converge to V_π, in the mean for a constant step-size parameter if it is sufficiently small, and with probability 1 if the step-size parameter decreases according to the usual stochastic approximation conditions:

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty \qquad \text{and} \qquad \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty. \tag{2.9}$$

We present two TD control methods that are widely used in practice: Sarsa and Q-Learning.

2.6.1 Sarsa

Sarsa is an on-policy method that updates the action-value function Q based on the current policy π and the action taken in the next state. The update rule is given by:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, a) - Q(S_t, A_t)\Big].$$

The convergence properties of the Sarsa algorithm depend on the nature of the policy's dependence on Q. For example, one could use ϵgreedy or ϵ-soft policies. Sarsa converges with probability 1 to an optimal policy and action-value function as long as all state–action pairs are visited an infinite number of times and the policy converges in the limit to the greedy policy.

2.6.2 Q-Learning

A similar update for an off-policy method is given by the Q-Learning algorithm by Watkins and Dayan (1992):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big].$$

In this case, the learned action-value function, Q, directly approximates the optimal action-value function Q^*, independent of the policy being followed. This dramatically simplifies the analysis of the algorithm and enabled early convergence proofs. The policy still has a crucial role in that it determines which state-action pairs are visited, and it is required for convergence that all pairs continue to be updated.
Q-learning is one of the most well known and widely used algorithms in RL and has been extended and adapted in many ways. The convergence of Q-learning has been proved for many cases, including linear function approximation, and it has been shown to be a very effective and robust method in practice.

2.7 Policy Gradient Methods

We now consider methods that directly learn the policy, rather than the value function. Policy gradient methods are a class of algorithms that search for the optimal policy by updating the policy parameters in the direction that increases the expected return making use of its gradients.
The first concept we will present is that of Function Approximation in the context of RL, than we'll present the foundamental theoretical result of Policy Gradient Methods: the *Policy Gradient Theorem* 2.7.1. Finally we'll explore some of the most common algorithms based on this principle.

2.7.1 Function Approximation

Up to now we have considered the case in which the value function or the policy are represented exaclly as tables. This is hardly done in practice, as prohibitively large state and action spaces require a more compact representation. Function approximation comes in handy as a technique used to store function information and generalize to unseen transitions efficiently. The most common function approximators are Neural Networks, followed by kernel functions, linear models, decision trees, and others. In such cases the value function or the policy are represented by a parameterized function, $V(s;\theta)$ or $\pi(a|s;\theta)$, where θ is the parameter vector of the function approximator. Once the function approximator is defined, the goal is to find the optimal parameters θ^* that maximize the expected return, most of the times by Stochastic Gradient Descent[3].
Consider the problem of finding the value function corresponding to a policy π, thus far we implemented the update rule for the value function by moving the value prediction $V(s)$ closer to the return G_t, which represents the target:

$$V(s) = V(s) + \alpha(G_t - V(s))$$

Function approximation allows to perform the same update on the parameter vector θ as follows:

$$\theta_{t+1} \doteq \theta_t + \alpha(U_t - V(s_t;\theta_t))\nabla V(s_t;\theta_t)$$

where U_t is the unbiased estimate for the target value V_π, and $\nabla V(s_t;\theta_t)$ is the gradient of the value function with respect to the parameter vector θ.

[3]Stochastic Gradient Descent (SGD) was first introduced by Robbins and Monro (1951)

2.7.2 Policy Gradient Theorem

Now consider a parameterized policy π_θ, which is differentiable almost everywhere, and the following objective function J for maximizing the expected episodic return:

$$\begin{aligned} J(\theta) &= \mathbb{E}_{S_0 \sim p_0, \pi_\theta}[G_0] \\ &= \mathbb{E}_{S_0 \sim p_0}\Big[\mathbb{E}_{\pi_\theta}[G_t \mid S_t = S_0]\Big] \\ &= \mathbb{E}_{S_0 \sim p_0}\Big[V_{\pi_\theta}(S_0)\Big] \end{aligned}$$

In this section we treat the episodic case, for which we define the performance measure as the value of the start state of the episode. We can simplify the notation without losing any meaningful generality by getting rid of the initial state distribution p_0 assuming that every episode starts in some particular (non-random) state s_0:

$$J(\theta) = V^{\pi_\theta}(s_0)$$

The idea of policy gradient algorithms is to maximize $J(\theta)$ over the parameters θ by performing gradient ascent Sutton and Barto (2018). Hence, we require the gradients $\nabla_\theta J(\theta)$, however it is a priori not obvious how the right-hand side $\mathbb{E}_{S_0 \sim p_0, \pi_\theta}[G_0]$ depends on θ as changes in the policy π also affect the state distribution d^π. The Policy Gradient Theorem Sutton et al. (1999); Marbach and Tsitsiklis (2001) yields an analytic form of $\nabla_\theta J(\theta)$ from which we can sample gradients that does not involve the derivative of d^π. Here, we focus on the undiscounted case, i.e. $\gamma = 1$. Note that any discounted problem instance can be reduced to the undiscounted case by letting the reward function absorb the discount factor Schulman et al. (2018).

Theorem 2.7.1. *(Policy Gradient Theorem) For a given MDP, let π_θ be differentiable w.r.t. θ and $\nabla_\theta \pi_\theta$ be bounded, let Q^{π_θ} be differentiable w.r.t. θ and $\nabla_\theta Q^{\pi_\theta}$ be bounded for all s and a. Then, there exists a constant η such that*

$$\nabla_\theta J(\theta) = \eta\, \mathbb{E}_{S \sim d^{\pi_\theta}, A \sim \pi_\theta}\Big[Q^{\pi_\theta}(S, A)\, \nabla_\theta \ln \pi_\theta(A \mid S)\Big]. \tag{2.10}$$

Proof. We follow the proof by Sutton and Barto (2018) the detailed one by Lehmann (2024). To simplify the notation, we omit subscripts θ for the policy π and all gradients ∇ but both always depend on the parameters θ.
Starting from the definition of the objective function, use the relationship between value and action-value function, $V^\pi(s) = \sum_a \pi(a \mid s)\, Q^\pi(s, a)$ and the product rule.

$$\nabla\, V^\pi(s) \;= \nabla \sum_a \pi(a \mid s)\, Q^\pi(s, a)$$

$$= \sum_a \nabla\pi(a \mid s)\, Q^\pi(s,a) \; + \sum_a \pi(a \mid s)\, \nabla Q^\pi(s,a). \tag{2.11}$$

Now, consider the recursive formulation of the action-value function

$$Q^\pi(s,a) = \sum_{s'} \sum_r p(s', r \mid s, a)\, \big(r + V^\pi(s')\big).$$

Due to the identity $\sum_r p(s', r \mid s, a) = p(s' \mid s, a)$ and since realized rewards r and environment transitions for a given action no longer depend on the policy, we can reformulate the gradients of Q^π w.r.t. θ.

$$\begin{aligned}
\nabla Q^\pi(s,a) &= \nabla \sum_{s'} \sum_r p(s', r \mid s, a)\, \big(r + V^\pi(s')\big) \\
&= \sum_{s'} \sum_r p(s', r \mid s, a)\, \nabla\big(r + V^\pi(s')\big) \\
&= \sum_{s'} \sum_r p(s', r \mid s, a)\, \nabla V^\pi(s') \\
&= \sum_{s'} p(s' \mid s, a)\, \nabla V^\pi(s')\,.
\end{aligned} \tag{2.12}$$

By using (2.11) and (2.12), we can transform (2.11) into a recursive form, which we are then going to unroll subsequently to yield an explicit form. In the following, we simply notation by defining

$$\phi(s) \coloneqq \sum_a \nabla\pi(a \mid s)\, Q^\pi(s,a). \tag{2.13}$$

Applying (2.13) and (2.12) to (2.11) in order and rearranging the integrals gives

$$\begin{aligned}
\nabla J(\theta) &= \sum_a \nabla\pi(a \mid s)\, Q^\pi(s,a) da + \sum_a \pi(a \mid s)\, \nabla Q^\pi(s,a) \\
&= \phi(s) + \sum_a \pi(a \mid s)\, \nabla Q^\pi(s,a)\, ds \\
&= \phi(s) + \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s, a)\, \nabla V^\pi(s') \\
&= \phi(s) + \sum_{s'} \sum_a \pi(a \mid s)\, p(s' \mid s, a)\, \nabla V^\pi(s')
\end{aligned} \tag{2.14}$$

To unroll Equation (2.14) across time, we introduce notation for multi-step transition probabilities. Let $\rho_\pi(s \to s', k)$ be the probability of transitioning from state s to s' after k steps under policy π. We have that

$$\rho_\pi(s \to s', 0) \coloneqq \begin{cases} 1 & \text{if } s = s', \\ 0 & \text{else} \end{cases}$$

and $\rho_\pi(s \to s', 1) \coloneqq \sum_a \pi(a \mid s)\, p(s' \mid s, a)$. Now, we can recursively write

$$\rho_\pi(s \to s'', k+1) = \sum_{s'} \rho_\pi(s \to s', k)\, \rho_\pi(s' \to s'', 1)\,.$$

Using this notation, iteratively substituting in (2.11) and (2.12), we can unroll (2.14):

$$\begin{aligned}
\nabla J(\theta) &= \phi(s) + \sum_{s'} \sum_{a} \pi(a \mid s)\, p(s' \mid s, a)\, \nabla V^\pi(s') \\
&= \phi(s) + \sum_{s'} \rho_\pi(s \to s', 1)\, \nabla V^\pi(s') \\
&= \phi(s) + \sum_{s'} \rho_\pi(s \to s', 1)\, \phi(s') + \sum_{a} \pi(a \mid s')\, \nabla Q^\pi(s', a) \\
&= \phi(s) + \sum_{s'} \rho_\pi(s \to s', 1)\, \phi(s') + \sum_{s''} \rho_\pi(s' \to s'', 1)\, \nabla V^\pi(s'')\,' \\
&= \phi(s) + \sum_{s'} \rho_\pi(s \to s', 1)\, \phi(s') \\
&\quad + \sum_{s''} \sum_{s'} \rho_\pi(s \to s', 1)\, \rho_\pi(s' \to s'', 1)\ \ \nabla V^\pi(s'')\,' \\
&= \phi(s) + \sum_{s'} \rho_\pi(s \to s', 1)\, \phi(s')\ + \sum_{s''} \rho_\pi(s \to s'', 2)\, \nabla V^\pi(s'')\,' \\
&= \rho_\pi(s \to s, 0)\phi(s) + \sum_{s'} \rho_\pi(s \to s', 1)\, \phi(s') \\
&\quad + \sum_{s''} \rho_\pi(s \to s'', 2)\, \phi(s'')\,' + \sum_{s'''} \rho_\pi(s \to s''', 3)\, \nabla V^\pi(s''')\,'' \\
&\ \ \vdots \\
&= \sum_{s'} \sum_{t=0}^{T} \rho_\pi(s \to s', t)\, \phi(s')
\end{aligned}$$

We set $\eta_s(s') \coloneqq \sum_{t=0}^{T} \rho_\pi(s \to s', t)$ to obtain

$$\begin{aligned}
\nabla_\theta J(\theta) &= \sum_{s'} \sum_{t=0}^{T} \rho_\pi(s \to s', t)\, \phi(s') \\
&= \sum_{s'} \eta_s(s')\, \phi(s') \\
&= \frac{\sum_{s''} \eta_s(s'')\ '}{\sum_{s''} \eta_s(s'')\ '} \sum_{s'} \eta_s(s')\ \phi(s') \\
&= \sum_{s''} \eta_s(s'')\ ' \sum_{s'} \frac{\eta_s(s')}{\sum_{s''} \eta_s(s'')\ '}\ \phi(s') \\
&= \sum_{s''} \eta_s(s'')\,' \sum_{s'} d^\pi(s')\, \phi(s')\,. \qquad (2.15)
\end{aligned}$$

In the final step, we used the identity

$$d^\pi(s') = \frac{\eta_s(s')}{\sum_{s''} \eta_s(s'')\ '},$$

which can be seen as $\eta_s(s')$ is the accumulate sum over probabilities of reaching s' after any number of steps for a given starting state.
Finally, we can derive the canonical form of the Policy Gradient Theorem from (2.15) by using the definition of $\phi(s)$, setting

$$\eta \coloneqq \sum_{s''} \eta_s(s'')\ '$$

ields:

$$\begin{aligned}
\nabla J(\theta) &= \sum_{s''} \eta_s(s'')\ ' \sum_{s'} d^\pi(s')\, \phi(s') \\
&= \eta \sum_{s'} d^\pi(s') \sum_a (\nabla \pi(a \mid s'))\, Q^\pi(s', a) \\
&= \eta \sum_{s'} d^\pi(s') \sum_a \pi(a \mid s')\, \frac{\nabla \pi(a \mid s')}{\pi(a \mid s')}\, Q^\pi(s', a) \\
&= \eta \sum_{s'} d^\pi(s') \sum_a \pi(a \mid s')\, (\nabla \ln \pi(a \mid s'))\, Q^\pi(s', a) \\
&= \eta\, \mathbb{E}_{S \sim d^\pi} \mathbb{E}_{A \sim \pi} \Big[Q^\pi(S, A)\, \nabla \ln \pi(A \mid S) \Big].
\end{aligned}$$

□

The Policy Gradient Theorem provides an explicit form of the policy gradients from which we can sample gradients. SGD requires the sample gradients only to be proportional to the gradient because any constant of proportionality can be absorbed into the step size α, which is otherwise arbitrary. The absence of the derivative of the state distribution d^π in the gradient expression, allows for the use of sample-based estimates of the gradient without the need for the exact knowledge of the state distribution.

2.7.3 REINFORCE

The REINFORCE algorithm, which is an acronym for "REward Increment = Non-negative Factor × Offset Reinforcement × Characteristic Eligibility." by Williams (1992), is a simple policy gradient method, while it precedes the formulation of the Policy Gradient Theorem, REINFORCE can be seen as a straightforward application of that, as it estimates the policy gradient by sampling trajectories ($Q1\pi(s, a)$ is not known and must be estimated). The algorithm makes use of the following

update rule:

$$\theta_{t+1} = \theta_t + \alpha\gamma^t G_t \nabla \log \pi(a_t|s_t;\theta_t)$$

where G_t is the return of the trajectory starting at time t, α is the step-size parameter, γ is the discount factor and $\nabla \log \pi(A_t|S_t;\theta_t)$ is the score function.
A common problem with REINFORCE is the high variance of the gradient estimates, which can be reduced by subtracting a baseline $b(s)$ from the return. Since the baseline does not depend on the action, it does not affect the expectation of the gradient and can be chosen freely. The idea is based on the following identity:

$$\nabla_\theta J(\theta) = \eta\, \mathbb{E}_{\pi_\theta}\Big[(Q^{\pi_\theta}(S,A) - b(s))\, \nabla_\theta \ln \pi_\theta(A \mid S)\Big].$$

since

$$\begin{aligned}
\sum_s d^\pi(s) \sum_a (Q^\pi(s,a) - b(s)) \nabla_\theta \pi_\theta(a|S_t) &= \\
&= \sum_s d^\pi(s) \sum_a (Q^\pi(s,a)) \nabla_\theta \pi_\theta(a|S_t) - \sum_s d^\pi(s) b(s) \sum_a \nabla_\theta \pi_\theta(a|S_t) \\
&= \nabla_\theta J(\theta) - \sum_s d^\pi(s) b(s) \nabla_\theta \sum_a \pi_\theta(a|s) \\
&= \nabla_\theta J(\theta)
\end{aligned}$$

A common choice as baseline is the value function $V(s)$, which can be itself estimated by MC sampling, leading to the following generalization of the REINFORCE algorithm:

$$\theta_{t+1} = \theta_t + \alpha(G_t - \hat{v}_{\pi_\theta}(S_t)) \frac{\nabla_\theta \pi_\theta(A_t|S_t)}{\pi_\theta(A_t|S_t)}$$

The quantity $G_t - V(S)$ can be interpreted as an estimate of the adavntage function

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{2.16}$$

A^π measures whether or not the action is better or worse than the policy's default behavior. There are in fact many unbiased estimators of the policy gradient that can be constructed to produce smaller variance in the more general form:

$$\nabla J = \mathbb{E}_\pi\Big[\Psi\, \nabla \ln \pi(A \mid S)\Big]$$

2.7.4 Actor Critic

Instead of estimating Q^π directly via sampling, we can learn such an estimate via function approximation. Algorithms that use this approach to learn a parameterized

Q^π or V^π function (the critic) in addition to learning the policy π (the actor) are referred to as actor-critic algorithms Mnih et al. (2016).

2.7.5 Trust Region Policy Optimization

Excessively large changes in the policy can result in instabilities during the training of RL algorithms. This is enforced by the fact that small changes in the parameter space do not necessary lead to small changes in the policy space. Small step sizes during gradient ascent cannot fully remedy this problem and would impair the sample efficiency of the algorithm. Trust Region Policy Optimization (TRPO) Schulman et al. (2017a) mitigates these issues by imposing a trust region constraint by KL Divergence on the size of policy update at each iteration while trying to improve sample efficiency of policy gradient methods.
In TRPO, since rollout workers and optimizers are running in parallel asynchronously, we have to use the actual policy that was used to collect the data, which is the old policy. As in off-policy RL, the policy used for collecting trajectories on is different from the policy to optimize. To account for the discrepancy between the distribution of collected training data and that of the target policy, an importance sampling estimator is employed:

$$\begin{aligned} J(\theta) &= \sum_{s\in\mathcal{S}} d^{\pi_{\theta_{\text{old}}}} \sum_{a\in\mathcal{A}} \left(\pi_\theta(a|s)\hat{A}_{\theta_{\text{old}}}(s,a)\right) \\ &= \sum_{s\in\mathcal{S}} d^{\pi_{\theta_{\text{old}}}} \sum_{a\in\mathcal{A}} \left(\beta(a|s)\frac{\pi_\theta(a|s)}{\beta(a|s)}\hat{A}_{\theta_{\text{old}}}(s,a)\right) \\ &= \mathbb{E}_{s\sim d^{\pi_{\theta_{\text{old}}}},a\sim\beta}\left[\frac{\pi_\theta(a|s)}{\beta(a|s)}\hat{A}_{\theta_{\text{old}}}(s,a)\right] \end{aligned}$$

where θ_{old} is the policy parameters before the update and β is the behavior policy followed while collecting trajectories. The objective function becomes:

$$J(\theta) = \mathbb{E}_{s\sim\rho^{\pi_{\theta_{\text{old}}}},a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}\hat{A}_{\theta_{\text{old}}}(s,a)\right] \doteq \mathcal{L}(\theta_k,\theta)$$

$\mathcal{L}(\theta_k,\theta)$ is the surrogate advantage, a measure of how policy π_θ performs relative to the old policy π_{θ_k} using data from the old policy. The theoretical TRPO update becomes:

$$\theta_{k+1} = \arg\max_\theta \mathcal{L}(\theta_k,\theta) \tag{2.17}$$

$$\bar{D}_{KL}(\theta||\theta_k) \leq \delta \tag{2.18}$$

The *trust region constraint* which enforces the distance between old and new policies

is measured by $\bar{D}_{KL}(\theta||\theta_k)$, an average KL-divergence between policies across states visited by the old policy:

$$\bar{D}_{KL}(\theta||\theta_k) = \mathbb{E}_{s\sim\pi_{\theta_k}}\Big[D_{KL}\left(\pi_\theta(\cdot|s)||\pi_{\theta_k}(\cdot|s)\right)\Big]$$

This approach ensures that the updated policy remains close to the previous one, while TRPO still guarantees a monotonic improvement over policy iteration (Schulman et al. (2017a)).
Still we have to solve the optimization problem, which is not trivial. Usually this is done by approximating the objective 2.17 to the first order and the constraint 2.18 to the second order.

2.7.6 Proximal Policy Optimization

Proximal Policy Optimization (PPO) by Schulman et al. (2017b) is a simplified version of TRPO that uses different objective function to ensure that the policy update is not too large, while being computationally more efficient. The clipped surrogate objective function is defined as follows:

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a),\ \ \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon\right)A^{\pi_{\theta_k}}(s,a)\right)$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio between the new and old policy, $\hat{A}_t$ is the advantage function, and ϵ is a hyperparameter that controls the size of the trust region. The clipped surrogate objective function is then optimized using stochastic gradient ascent.
The main difference between PPO and TRPO is that PPO solves the problem using a first order method, which is much simpler to implement and faster to train, as we don't need to compute any inverse Hessian-gradient product.

Chapter 3

Partially Observable Markov Decision Processes

Until now we have considered the case of fully observable MDPs, where the agent has access to the complete state of the environment. However, this is not always the case, and in many real-world scenarios, the agent has only partial information about its surroundings. As MDPs are controlled Markov Chains, POMDPs are controlled Hidden Markov Models. Figure 3.1 shows how the underlying process states are hidden from the agent, which can only observe the environment through its sensors.

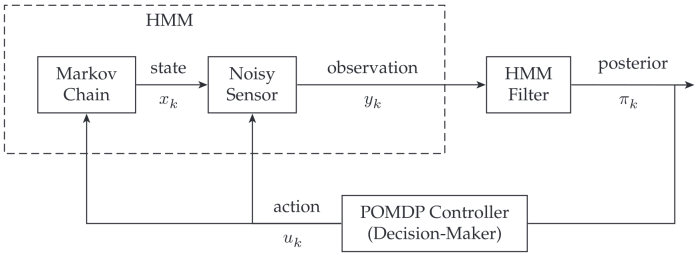


Figure 3.1: Partially Observable Markov Decision Process (Krishnamurthy, 2016)

As such, POMDPs are an extension of MDPs to the case where the full observability assumption is relaxed in favour of a more general and realistic assumption that the agent only has partial knowledge of the state of the system but there exist observations/signals which yield probabilistic beliefs about the hidden state.
The next sections will be based on Spaan (2012) and Hauskrecht (2000) analysis of POMDPs, originally introduced by the work of Drake (1962) Smallwood and Sondik (1973), Åström (1965) and Lovejoy (1991).

Definition 3.0.1. A *Partially Observable Markov Decision Process* is a tuple $P =$

$(S, A, \Omega, \mathcal{T}, \mathcal{R}, \mathcal{O})$ where:

- S is a finite set of states
- A is a finite set of actions
- Ω is a finite set of observations
- $\mathcal{T}$ is the transition function $\mathcal{T} : S \times A \times S \to [0, 1]$
- $\mathcal{R}$ is the reward function $\mathcal{R} : S \times A \times S \to \mathbb{R}$
- $\mathcal{O}$ is the observation function $\mathcal{O} : S \times A \times \Omega \to [0, 1]$

Example 3.0.1. Consider the two-state infinite-horizon POMDP by Jaakkola et al. (1994) depicted in 3.2. The POMDP is defined by two states and two possible actions in each state. The agent receives an observation independent of the state, and the reward is r if the agent changes state and $-r$ otherwise.
The optimal policy in the underlying MDP is to take action a^2 in state s^1 and action a^1 in state s^2, yielding a reward of r in each time step and producing an optimal value function $V^*(s) = \frac{r}{1-\gamma}$ by geometric series convergence.
In the POMDP, the agent receives the same observation in both states, and as a result, there are only two memoryless stationary deterministic policies possible: always taking action a^1 or always taking action a^2. Both policies yield a maximum expected reward of $r + \frac{\gamma r}{1-\gamma}$ if fortunate enough to take the first action that leads to a different state. The best stationary stochastic policy would otherwise be able to achieve an expected discounted reward of 0, when it chooses either action 50% of the time, still far from the optimal value of the MDP.
However, if the agent could remember what actions it had executed, it could execute a policy that alternates between the two actions, achieving $\frac{\gamma r}{1-\gamma} - r$ in the worst case.

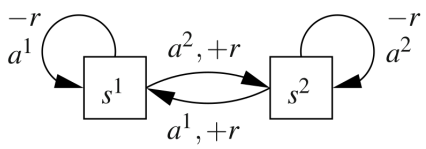


Figure 3.2: The need for Memory

This example shows how the lack of memory can severely limit the agent's ability to solve the problem. In the next sections, we will introduce the main concepts and algorithms used to solve POMDPs based on the formalization of the concept of memory.

3.1 Belief State MDPs

Since in POMDPs, the underlying process states are unknown, the action choices will be based on the information available to the agent in the form of past actions and observations, giving rise to the concept of *Information State*.

Definition 3.1.1. (Complete Information State) The complete information state at time t, denoted I_t, consists of:

- prior belief b_0 on states in S at time $t = 0$;
- a complete history of actions and observations $\{o_0, a_0, o_1, ..., o_{t-1}, a_{t-1}, o_t\}$

A sequence of information states defines a controlled Markov process called information-state MDP. In this context, a policy $\pi : I \to A$ is a mapping of information states to actions (possibly distributions over actions). The new information state I_{t+1} is obtained by applying an update function τ to the the previous state I_t, previous action a_{t-1} and observation o_t:

$$\tau : \mathcal{I} \times A \times O \to \mathcal{I}$$

It's easy to see that a POMDP can be cast into the information-state MDP by using complete information states, revisiting the POMDP such that states become information-states while transitions and rewards are updated according to:

$$\mathcal{R}(I, a) = \sum_{ss'} \mathcal{T}(s, a, s')\mathcal{R}(s, a, s')P(s|I)$$

$$\mathcal{T}(I, a, I') = \sum_{o} \tau(I, a, o)P(o|I, a)$$

It's also important to note that the information available can be also summarized in the so-called *sufficient information states*. Such states must preserve the necessary information content as well as the Markov property of the information-state.

Definition 3.1.2. (Sufficient information state process). Let $\mathcal{I}$ be an information state space and $\tau : \mathcal{I} \times A \times \Theta \to \mathcal{I}$ be an update function defining an information process $I_t = \tau\left(I_{t-1}, a_{t-1}, o_t\right)$. The process is sufficient with regard to the optimal control if for every time step t, it satisfies:

$$P\left(s_t \mid I_t\right) = P\left(s_t \mid I_t^C\right)$$

$$P\left(o_t \mid I_{t-1}, a_{t-1}\right) = P\left(o_t \mid I_{t-1}^C, a_{t-1}\right)$$

where I_t^C and I_{t-1}^C are complete information states.

Being able to define a sufficient information state process is crucial since it allows us to avoid the curse of dimensionality of enlarging complete information states.
This brings us to the most common concept of *belief state* as a form of sufficient information state Åström (1965).
A belief is a probability distribution over the states of the system which summarizes the history of actions and observations. Each POMDP assumes an initial belief b_0, then at each time step the belief is updated by the observation and the action taken through Bayes' rule:

$$\tau(b, a, o) = b^{ao}(s') = \frac{p(o|s', a)}{p(o|b, a)} \sum_{s \in S} p(s'|s, a)b(s) \tag{3.1}$$

where $p(s'|s, a) = T(s, a, s')$ and $p(o|s', a) = O(s, a, o)$, and

$$p(o|b, a) = \sum_{s' \in S} p(o|s', a) \sum_{s \in S} p(s'|s, a)b(s)$$

.
It is thus possible to define a MDP over belief states. This transformation requires the transition and observation functions to be known to the agent, and hence can be applied only in model-based methods.
The key point is that belief-state MDPs are fully observable even though the original problem involves hidden quantities. This formulation effectively turns the problem into a planning one in the space of beliefs.
Belief-state MDPs are the primary object of study in the field of POMDPs.

3.2 Policies and Value Functions

As in the fully observable context, the goal of the agent is to find a policy that maximizes the expected return. In the POMDP case, the policy is a mapping from the set of probability distributions over S to a probability distribution over actions $\pi : \Delta(S) \to \Delta(A)$ and the value function $V^\pi : \Delta(S) \to \mathbb{R}$. In the infinite-horizon discounted case, the value function is defined as:

$$V^\pi(b) \doteq \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^k \mathcal{R}(b_t, \pi(b_t)) | b_0 = b \right]$$

where $\mathcal{R}(b_t, \pi(b_t)) = \sum_s \mathcal{R}(s, \pi(b_t))b_t(s)$.
Revisiting equations in the previous section, the Bellman optimality equation for the value function in a belief space MDP is given by:

$$V^*(b) = \max_a \left[\sum_s R(s, a)b_t(s) + \gamma \sum_o p(o|b, a)V^*(b'_{a,o}) \right]$$

or alternatively, $V^* = HV^*$ where H is the Bellman backup operator for POMDPs defined as:

$$HV(b) = \max_a \left[\sum_s \mathcal{R}(s,a)b(s) + \gamma \sum_o p(o|b,a)V(b'_{a,o}) \right]$$

The properties of the H operator are sufficient to guarantee the existence of a unique optimal value function given the Banach contraction theorem. *Exact Value Iteration* is therefore formulated as the iterative application of the H operator to an initial value function until convergence - practically until the difference between the value function at two consecutive iterations is below a given threshold.

serve una dimostrazione H contrazione

$$V^{n+1} = HV_n = H^n V_0$$

The major difficulty in applying exact value iteration is that the belief space is continuous, and updates over the whole space are infeasible. Fortunately, the value function can be parameterized by a finite set of vectors; the following result is originally from Smallwood and Sondik (1973).

Theorem 3.2.1. *(Piecewise linear and convex Value function). Let V_0 be an initial value function that is piecewise linear and convex. Then the i-th value function obtained after a finite number of update steps for a belief-state MDP is also finite, piecewise linear and convex, and is equal to:*

$$V_i(b) = \max_{\alpha_i \in \Gamma_i} \sum_{s \in S} b(s)\alpha_i(s)$$

where b and α_i are vectors of size $|S|$ and Γ_i is a finite set of vectors (linear functions) α_i.

Proof. It follows from induction. The result holds for the horizon one value function $V_0 = \sum_s \mathcal{R}(s,a)b(s) = \sum_i b_i \alpha_0^i(s) = b \cdot \alpha_0$ Assuming the hypothesis holds for V_n, we can show that it holds for V_{n+1} as well.

$$\begin{aligned}
V_{n+1}(b) &= \max_a \Big[\sum_s \mathcal{R}(S,a)b(s) + \gamma \sum_o p(o|a,b)V_n(b_a^o) \Big] \\
&= \max_a \Big[b \cdot r_a + \gamma \sum_o p(o|a,b) \max_{\{\alpha_n^i\}_i} \sum_{s'} b_a^o(s')\alpha_n^i(s') \Big] \\
&= \max_a \Big[b \cdot r_a + \gamma \sum_o \max_{\{\alpha_n^i\}_i} \sum_{s'} p(o|s',a) \sum_s p(s'|s,a)b(s)\alpha_n^i(s') \Big] \\
&= \max_a \Big[b \cdot r_a + \gamma \sum_o \max_{\{g_{a,o}^i\}_i} b \cdot g_{a,o}^i \Big], \\
&= \max_a \Big[b \cdot r_a + \gamma b \cdot \sum_o arg \max_{\{g_{a,o}^i\}_i} b \cdot g_{a,o}^i \Big],
\end{aligned}$$

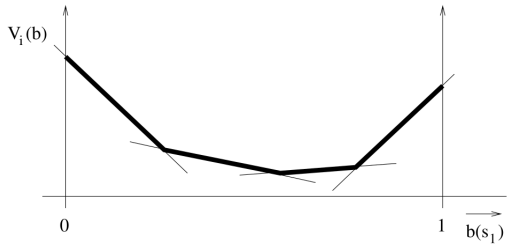


Figure 3.3: Convex Value function for a two states POMDP.

$$= \max_{g_a^b} b \cdot g_a^b$$

denoting

$$g_{a,o}^i(s) = \sum_{s'} p(o|s', a)p(s'|s, a)\alpha_n^i(s').$$

and

$$g_a^b = r_a + \gamma \sum_o arg \max_{\{g_{a,o}^i\}_i} b \cdot g_{a,o}^i$$

therefore one can express the value function as stated.

□

Remark 3.2.1. Even though the optimal value function for an infinite-horizon POMDP is convex, it may not be piecewise linear. Nonetheless, it can be approximated arbitrarily closely by a piecewise linear and convex value function.

The main idea behind most value iteration algorithms is that for a given value function V_n and a single belief point b, we can easily compute the vector α_{n+1}^b of HV_n such that:

$$\alpha_{n+1}^b = \arg \max_{\{\alpha_{n+1}^i\}_i} b \cdot \alpha_{n+1}^i$$

where $\{\alpha_{n+1}^i\}_{i=1}^{|HV_n|}$ is the (unknown) set of vectors for HV_n. We will denote this operation $\alpha_{n+1}^b = \text{backup}(b)$. It computes the optimal vector for a given belief b by back-projecting:

$$backup(b) = \arg \max_{g_a^b} b \cdot g_a^b \tag{3.2}$$

Due to the convexity of the value function, the expected discounted reward typically increases as the entropy of b decreases Figure 3.3. This remark is the basis for many entropy-regularized algorithms in POMDPs.
The total number of all its possible linear functions is $|A|\Gamma_i|^{|O|}$ (one for every combination of actions and permutations of α_i vectors of size $|O|$).

Policy Similarly to the MDP case, in POMDPs, the optimal policy with respect to the Value function V is:

$$\pi_V(b) = \underset{a}{\operatorname{argmax}} \left[\mathcal{R}(b,a) + \gamma \sum_{o \in \Omega} P(o \mid b,a)\, V(b^{a,o}) \right]$$

Computing a policy for the current belief state b using the above equation requires computing all the $|A| \times |\Omega|$ successors of b, with a cost of $|S|^2$ for each successor. Then, computing the value at each successor requires $|S| \times |\Gamma|$ operations (using the α-vector representation).
However, we can label the vector resulting from the point-based backup operation with the action associated with it. Then, all we need is to find the best α-vector for the current belief state and execute the corresponding action, with a computation cost of only $|S| \times |\Gamma|$ for finding the best action when following V.

3.3 Exact Value Iteration

The process of *exact value iteration* Sondik (1978) Monahan (1982), starts by enumerating all possible linear functions:

$$H_{POMDP} V_n = \bigcup_a G_a, \quad \text{with} \quad G_a = \bigoplus_o \left\{ \frac{1}{|\Omega|} r_a + \gamma g^i_{a,o} \right\}_i$$

Since the complete set of linear functions is rarely needed (some of the linear functions are dominated by others, therefore not affecting the value function), redundant vectors are then pruned:

$$V_{n+1} = prune(H_{POMDP} V_n)$$

Incremental pruning strategies by Cassandra et al. (1997) can be adopted to reduce the computational cost of the pruning linear program. Still, the pruning steps are usually expensive and seem to make a difference only in the constant factors rather than the order of growth of needed vectors.
In each iteration, the value function is updated across the entire belief space, vectors G_a are $|\Gamma| \times |A| \times |\Omega|$ and each one is computed in $|S|^2$ operations. Then, we create $|\Gamma|^{|\Omega|}$ new vectors for each action, with a complexity of $|S|$ for each new vector. The overall complexity of a single iteration is $O\left(|\Gamma| \times |A| \times |\Omega| \times |S|^2 + |A| \times |S| \times |\Gamma|^{|\Omega|}\right)$.
An alternative to this approach relies on the idea of computing a useful linear function for a single belief state, and then defining a set of constraints over the belief space where this vector is guaranteed to be dominant (Sondik, 1971; Smallwood Sondik, 1973).

The main challenge is identifying all belief points that seed valuable linear functions. Approaches to address this include Sondik's one- and two-pass algorithms Sondik (1978), Cheng's methods Cheng (1988), and the Witness algorithm Littman (1994); Cassandra et al. (1994).

3.4 Point Based Value Iteration

Approximate solution methods are necessary to solve POMDPs in practice, since even modest sized problems are intractable with exact methods. The main assumption behind Point Based Value Iteration (PBVI) by Pineau et al. (2003) is that it is unlikely to reach most of the points in the belief simplex. Computing solutions only for those parts of the belief simplex that are reachable, i.e., that can be actually encountered by interacting with the environment, can drastically reduce the computational complexity of the problem.
The PBVI algorithm solves a POMDP for a finite set of belief points B initializing an α-vector for each $b \in B$, and repeatedly updating (via value backups) the value of that α-vector, preserving the piecewise linearity and convexity of the value function, and defining a value function over the entire belief simplex.

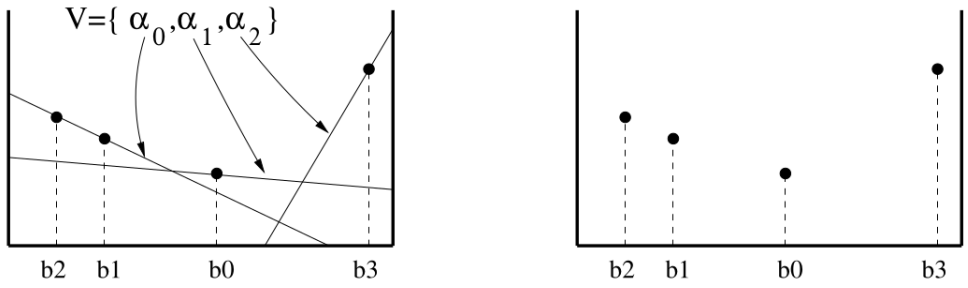


Figure 3.4: Point-Based Value Iteration

The point-based update follows exactly Eq. 3.2 for each belief point in the set B. This is repeated h times for finite h-horizon problems, or a predefined number of iterations in the infinite case.
The next step is to enlarge the set of belief points by selecting for each belief b in B a successor b' that is the most distant from the set B. That is, let L be a distance metric, then define:

$$|b' - B|_L = \min_{b \in B} |b - b'|_L$$

and focus on candidate successors generated using forward simulation, thus:

$$b' = \max_{a,o} |b^{a,o} - B|_L$$

The set of successor points, one b for each b in B_{i-1} , are added into Bi (along with the previous points in Bi) to create the new set Bi+1 . Experiments by various researchers show little, if any, sensitivity to the distance metric chosen, and L 1 has useful theoretical properties for the convergence of PBVI. The full procedure is formally described in Algorithm
As per previous analysis, the complexity of the point-based backup on a single belief is $O(|A| \times |\Omega| \times |V| \times |S|^2 + |A| \times |S| \times |\Omega|$). However, a full backup for the subset B will not require $|B|$ times the complexity of a single point-based backup, because the $g^{a,o}$ are independent of the current belief point b. Hence, executing a backup for $|B|$ belief points over a single value function V, where we compute every $g^{a,o}$ only once, requires $O(|A| \times |\Omega| \times |V| \times |S|^2 + |B| \times |A| \times |S| \times |\Omega|$), as compared with the $O(|A| \times |\Omega| \times |V| \times |S|^2 + |A| \times |S| \times |V|^{|\Omega|})$ of a single iteration of the exact backup.
For any belief set B and horizon n, PBVI produces an estimate V_n^B such that the error between V_n^B and the true value function V_n^* is bounded, and the bound depends on how dense B is in the Belief space Pineau et al. (2003).

PBVI represents a class of algorithms based on the above ideas. They differ mainly in in how new belief points are selected, in how belief points are updated in each backup iteration, and in other minor optimizations, such as pruning techniques or better initial value functions. In general, these algorithms always interleave belief point collection and belief point updates.

3.4.1 Perseus

Perseus by Spaan and Vlassis (2005) introduces a method to reduce the number of belief updates required at each iteration, while still improving the value function for a large number of beliefs. With fewer belief point updates per step, there is room for many more iterations of belief point updates, for a given time constraint. It starts with running random exploration in belief space, collecting belief points until a (relatively) large number of points are collected.
Then, given a previous value function V_n, the Perseus backup stage is performed:

$$V_{n+1} = \tilde{H}_{Perseus} V_n$$

The operator is built such that V_{n+1} improves the value of all $b \in B$: upper bounds V_n over B (but not necessarily over $\Delta(S)$ which would require linear programming):

$$V_{n+1}(b) \geq V_n(b) \quad \forall b \in B$$

$\tilde{H}_{Perseus}$ defines a set $\tilde{B} = B$ and a new empty value function $V_{n+1} = \emptyset$. A belief

point $b \in \tilde{B}$ is then selected uniformly at random and used to compute a new vector $\alpha_{new} = backup(b, V_n)$. If $\alpha_{new} \cdot b > V_n(b)$, then α_{new} is added into V_{n+1}, and we remove from $\tilde{B}$ all b' such that $\alpha_{new} \cdot b' > V_{n+1}(b')$. That is, all belief points whose value was improved by α_{new} will not be backed up at the current iteration. If $\alpha_{new} \cdot b \leq V_n(b)$, we add $\alpha_{old} = argmax\alpha \in V_n \alpha \cdot b$ into V_{n+1}. The iteration ends when $\tilde{B} = \emptyset$, and V is set to V'.

Note however that in this algorithm, it is assumed that if $\alpha_{new} \cdot b > V_m(b)$, then $|\alpha_{new} \cdot b - V^*(b)| < |V_n(b) - V^*(b)|$. This is only true if we initialize V to a lower bound on V^* . Therefore, unlike PBVI, Perseus requires a lower bound initialization $V_0 = \{ \alpha_0 \}$ such that:

$$\alpha_0 = \frac{1}{1-\gamma} min_{s,a} \mathcal{R}(s, a)$$

The key advantages over PBVI are that the belief collection step is fast, and the value function update tends to be represented by a number of α-vectors much smaller than $|B|$. Thus, Perseus can collect a set of belief points much larger than PBVI.

This comes at some cost; first, the random belief gathering assumes that it is relatively simple to obtain good belief points, which also represents a practical limitation in domains that require long trajectories. Second, even though the value for every point $b \in B$ gets closer to $V^*(b)$ with every iteration, many belief points will not get backed up and the value function may remain far from optimal for many beliefs.

3.5 MDP based Approximations

To address challenges arising from the curse of dimensionality (the belief space is continuous and high-dimensional) and the curse of history (decisions must account for past observations), approximate solution methods that leverage techniques from fully observable MDPs have been proposed. MDP-based approximations exploit the underlying state transition structure of the problem, aiming to balance computational efficiency with solution quality by: Reducing complexity – treating the POMDP as an MDP (e.g., assuming full observability), avoiding costly belief-space computations. Leveraging Standard MDP Solvers – Many efficient algorithms exist for solving MDPs, and approximations allow their application to POMDPs. Providing Practical Policies – While not optimal, these approximations yield reasonable decision policies that work well in many real-world scenarios.

MDP approximation The simplest way to approximate the value function for a POMDP is to assume that states of the process are fully observable:

$$\hat{V}(b) = \sum_s b(s) \cdot V^*_{MDP}(s)$$

where V^*_{MDP} is the optimal value function for the underlying MDP. The update rule for the value function can be written as:

$$\begin{aligned} \widehat{V}_{i+1}(b) &= \sum_{s\in S} b(s) \max_{a\in A} \left[\mathcal{R}(s,a) + \gamma \sum_{s'\in S} P(s'|s,a) \alpha_i^{MDP}(s') \right] \\ &= (H_{MDP}\widehat{V}_i)(b). \end{aligned}$$

where α_i^{MDP} is the α-vector of the MDP value function at iteration i.

QMDP A variant, more precise approximation by (Littman, Cassandra, Kaelbling, 1995):, is the QMDP algorithm which approximates the value function by the Q-function of the underlying MDP:

$$\hat{V}(b) = \max_a \sum_s b(s) Q^*_{MDP}(s,a)$$

Ielding the following update rule:

$$\begin{aligned} \widehat{V}_{i+1}(b) &= \max_{a\in A} \sum_{s\in S} b(s) \left[\mathcal{R}(s,a) + \gamma \sum_{s'\in S} p(s'|s,a) \max_{\alpha_i\in\Gamma_i} \alpha_i(s') \right] \\ &= (H_{QMDP}\widehat{V}_i)(b). \end{aligned}$$

FIB Both the MDP and the QMDP approaches ignore partial observability and use the fully observable MDP as a surrogate. FIB by Hauskrecht (1997) tryes to improve on these approximations and account (at least to some extent) for the partial observability.

$$\begin{aligned} \left(H\widehat{V}_i\right)(b) &= \max_{a\in A} \left\{ \sum_{s\in S} \mathcal{R}(s,a)\, b(s) + \gamma \sum_{o\in\Theta} \max_{\alpha_i\in\Gamma_i} \sum_{s'\in S} \sum_{s\in S} p(s',o|s,a)\, b(s)\, \alpha_i(s') \right\} \\ &\leq \max_{a\in A} \left\{ \sum_{s\in S} \mathcal{R}(s,a)\, b(s) + \gamma \sum_{o\in\Theta} \sum_{s\in S} \max_{\alpha_i\in\Gamma_i} \sum_{s'\in S} p(s',o|s,a)\, b(s)\, \alpha_i(s') \right\} \\ &= \max_{a\in A} \sum_{s\in S} b(s) \left[\mathcal{R}(s,a) + \gamma \sum_{o\in\Theta} \max_{\alpha_i\in\Gamma_i} \sum_{s'\in S} p(s',o|s,a)\, \alpha_i(s') \right] \\ &= \max_{a\in A} \sum_{s\in S} b(s)\, \alpha^a_{i+1}(s) \\ &= \left(H_{FIB}\widehat{V}_i\right)(b). \end{aligned}$$

Remark 3.5.1. A solution for the fast informed bound approximation can be found

by solving an MDP with $|S|A||O|$ states, $|A|$ actions (Hauskrecht (2000)).

UMDP On the contrary, discarding all observations available to the decision maker leads to the so called called unobservable MDP update:

$$\begin{aligned}
(H\widehat{V}_i)(b) &= \max_{a\in A}\left\{\sum_{s\in S}\mathcal{R}(s,a)b(s)+\gamma\sum_{o\in\Theta}\max_{\alpha_i\in\Gamma'_i}\sum_{s'\in S}\sum_{s\in S}p(s',o|s,a)b(s)\alpha_i(s')\right\}\\
&\geq \max_{a\in A}\left\{\sum_{s\in S}\mathcal{R}(s,a)b(s)+\gamma\max_{\alpha_i\in\Gamma'_i}\sum_{o\in\Theta}\sum_{s\in S}\sum_{s'\in S}p(s',o|s,a)b(s)\alpha_i(s')\right\}\\
&= \max_{a\in A}\left\{\sum_{s\in S}\mathcal{R}(s,a)b(s)+\gamma\max_{\alpha_i\in\Gamma'_i}\sum_{s\in S}\sum_{s'\in S}p(s'|s,a)b(s)\alpha_i(s')\right\}\\
&= \left(H_{U\,MDP}\widehat{V}_i\right)(b).
\end{aligned}$$

Each backup operator presented is a contraction preserving linearity and convexity, moreover we have that:

$$H_{MDP}V \geq H_{QMDP}V \geq H_{FIB}V \geq H_{POMDP}V \geq H_{UMDP}V$$

At the cost of a more complex update rule, the upper bound to the true value function is tighter, as the inequalities also hold for their fixed points by the following result.

Theorem 3.5.1 (Value-Function Comparison)**.** *Let H_1 and H_2 be two value-function mappings defined on $\mathcal{V}_1$ and $\mathcal{V}_2$ such that:*

1. *H_1, H_2 are contractions with fixed points V_1^*, V_2^*;*
2. *$V_1^* \in \mathcal{V}_2$ and $H_2V_1^* \geq H_1V_1^* = V_1^*$;*
3. *H_2 is an isotone mapping.*

Then $V_2^* \geq V_1^*$ holds.

Proof. We have that:

$$\begin{aligned}
H_2V_1^* &\geq H_1V_1^* = V_1^*\\
H_2H_2V_1^* &\geq H_2V_1^* \geq H_1V_1^* = V_1^*\\
H_2^nV_1^* &\geq V_1^*
\end{aligned}$$

In the limit, by contraction property, we have that $V_2^* \geq V_1^*$. □

The presented upper bounds do present some limitations, in particular they are are likely to fail when the belief has a complex shape and when many information gathering actions are needed to reduce the uncertainty of the belief state.

3.6 Entropy based Heuristics

Cassandra (1998) showed how the dual problem of taking the action that ields the greatest rewards and reduce the uncertanty on the information state could be tackled using the entropy of the belief to go from pure exploitive policies to information gathering policies. Still these heuristics may be misleading if the minimum of $V(b)$ does not occur at the point of greatest entropy, that is, the uniform belief state. Based on the same ideas, Infotaxis by Vergassola et al. (2007) greedily minimizes uncertainty: at each step, an infotactic agent chooses the action that maximizes the expected information gain:

$$a^* = \arg\max_a \sum_o H(b|a) - H(b)$$

where $H(b) = -\sum_s b(s) \log b(s)$ is the entropy of the belief state and $H(b|a)$ is the expected entropy of the belief state after taking action a:

$$H(b|a) = -\sum_o p(b'|b,a,o)p(o|a,b)H(b') = \sum_{b^{a,o}} p(o|a,b)H(b^{a,o})$$

Liu et al. (2024) proposed an improvement over the Infotaxis algorithm, called Space-Aware Infotaxis, combining information gain with spatial information to improve the efficiency of the search. The algorithm uses a spatial map of the environment to guide the search, taking into account the distance between the agent and the target, as well as the spatial distribution of information. By incorporating spatial information, Space-Aware Infotaxis can optimize the search process and reduce the time required to locate the target.

Part II

Experiments

Chapter 4

Methods

Problem Description Surgical robot navigation presents unique challenges, particularly when operating on soft, deformable organs. Unlike rigid environments, where a robot can rely on predefined maps and fixed obstacle positions, deformable organs continuously change shape due to external forces, physiological processes, and surgical interactions. This introduces uncertainty in navigation, making it difficult to plan and execute precise movements.

Key Challenges

1. **Deformation of Organs:** Soft tissues can shift, stretch, or compress unpredictably, leading to non-static obstacles and pathways.
2. **Partial Observability:** Due to the limited field of view of endoscopic cameras, occlusions, and the dynamic nature of the surgical site, the robot does not have full knowledge of the environment at any given time.
3. **Real-Time Adaptation:** The robot must continuously update its navigation strategy based on incomplete and evolving sensory data.
4. **Safety Constraints:** Unlike traditional robotic navigation, errors in surgical settings can result in severe patient harm, requiring ultra-precise movement planning.

POMDPs for Surgical Robot Navigation Given the challenges of deformable environments and partial observability, POMDPs provide a suitable framework for modeling surgical robot navigation. A POMDP extends a traditional MDP by incorporating uncertainty in both state transitions and observations, making it ideal for scenarios where the robot does not have full knowledge of the environment at any given moment.

In a surgical context, the robot must estimate the underlying state of the organ based on noisy and limited sensory data. The POMDP framework allows for probabilistic reasoning over possible states and dynamically updating beliefs based on new observations. This enables more robust decision-making under uncertainty, improving navigation efficiency and patient safety through uncertainty quantification.

Relation to a 2D Deformable Maze A simplified analogy can be drawn between surgical robot navigation and a robot navigating a 2D deformable maze. In this case:

- The **maze walls** represent anatomical structures that move and deform in real-time.
- The **robot's sensors** are analogous to a surgical camera with a limited and dynamic field of view.
- The **algorithm** must adapt to continuous environmental changes, rather than relying on a static map.
- The **goal** is to reach a target point while avoiding dynamically shifting obstacles and ensuring efficient movement.

Relation to 3D Simulation of Surgical Manipulation of Soft Objects A crucial aspect of developing and validating surgical robot navigation algorithms is the use of 3D simulations of soft object manipulation. These simulations provide a controlled environment where the behavior of deformable tissues can be accurately modeled using finite element methods. By simulating realistic tissue deformations, forces, and interactions, we can test and refine robotic control strategies before deploying them in real-world surgical settings. Additionally, 3D simulations enable the integration of sensory feedback loops, allowing for the evaluation of different perception and decision-making strategies under conditions of uncertainty.

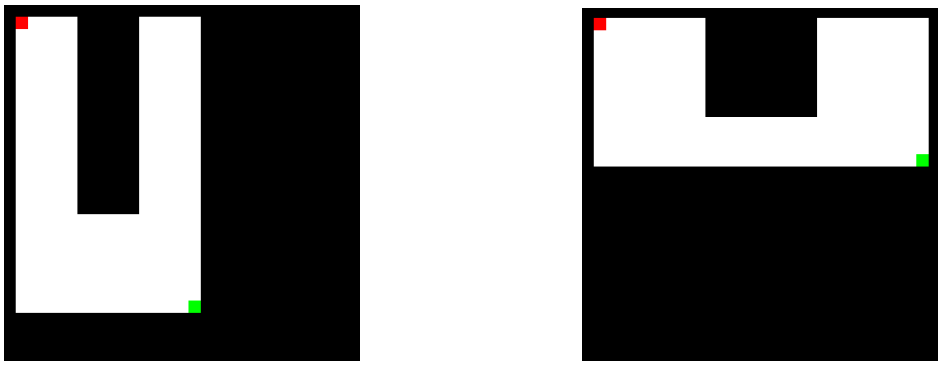
This analogy highlights the importance of real-time sensing, dynamic path planning, and uncertainty management in both surgical robotics and general deformable environment navigation. Future advancements in soft-tissue modeling, AI-driven prediction, and sensor fusion will be crucial in improving the efficacy of surgical robot navigation under these constraints.
We adopted a bottom up approach, increasing the complexity of the task in increasingly complex environments. Our objective is to explore the applicability of RL and POMDPs to navigation and manipulation in deformable environments related to the surgical tasks.

In the following we assumied the model of the environment to be known or learnable, given the preoperative imaging medical data. This would allow us to model the problem as a POMDP and apply the methods described in previous chapters.

4.1 Gridworld

The first task we consider is a simple gridworld, where the agent had to navigate from a random starting position to a goal position. The twist is given by the deformable property of the environment.

State Space The environment is a 2D grid of size 30×30 where each cell can be either empty or occupied by an obstacle. the state is represented as a tuple (x, y, ϕ, θ) where x and y are the discrete coordinates of the agent in the grid, ϕ is the orientation of the agent (it can point up, down, right, or left) and θ is a parameter controlling the linear scale deformation relative to xy axis for a total of 81 discrete deformations. We assume perfect observability upon the agent's position and orientation while the deformable property of the environment has to be estimated through observations. This is done to reflect the clinical setting where the preoperative imaging data is available but the intraoperatory setting might be different due to the deformation of the organs and tissues.



(a) θ = (5,8) (b) θ = (9,4)

Figure 4.1: Gridworld Environment for $s = (1, 1, 0, \theta)$

Notice that the deformable property of the environment directly affects the target position (fig. 4.1).

Action Space The agent can move in four directions: up, down, left, right. The action space is therefore $A = \{0, 1, 2, 3\}$.

Observation Space Every time the agent moves, it receives an observation which corresponds to the type of adjacent cells in the grid $O = \{ o \in \{ 0, 1 \}^4 \}$ where 0 refers to an empty cell and 1 to an obstacle (fig. 4.2).

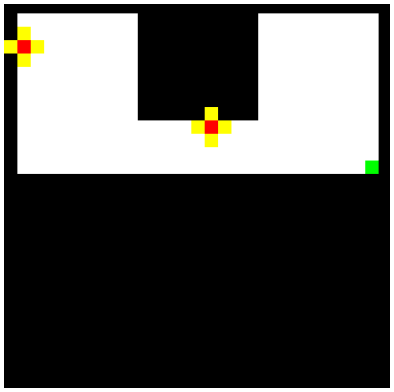


Figure 4.2: Observations in Gridworld

Transition Function Always assuming deterministic transitions, the transition function is defined as follows:

$$\mathcal{T}(s'|s,a) = \begin{cases} 1 & \text{if } s' \text{ is the result of applying action } a \text{ to state } s, \\ 0 & \text{otherwise.} \end{cases}$$

In this simulation we assume that every transition that ends up inside the grid is a valid one. This is done to simplify the problem and focus on the deformable property only without the need to consider an elaborate transition model. The agent will nonetheless avoid obstacles if possible thanks to the reward structure.

Reward Function The reward function $\mathcal{R}(s,a,s')$ is defined as follows:

$$\mathcal{R}(s,a,s') = \begin{cases} \frac{-0.1}{mapsize} & s' \neq s_{goal} \wedge \text{free cell}, \\ \frac{-0.2}{mapsize} & s' \neq s_{goal} \wedge \text{hit wall}, \\ 1 & s' = s_{goal}. \end{cases}$$

Observation Function The conditional observation probabilities $O(o|s,a)$ are also deterministic:

$$O(o|s,a) = O(o|s) = \begin{cases} 1 & \text{if } (x,y) \text{ adjacent cells for map } f_\theta(M) \text{are compatible with } o, \\ 0 & \text{otherwise.} \end{cases}$$

where adjacent cells are defined as the cells in the 4 directions of the agent's orientation as in fig. 4.2.

Belief State Because the agent does not directly observe the environment's state, the agent must make decisions under uncertainty of the true environment state. The general update rule for the belief state is given by the Bayes' rule. Adapting it to the gridworld case, given the only source of uncertanty is only on the deformation parameter, the belief state is a probability distribution over possible deformations. Denoting with $\psi = (x, y, \phi)$ the observable part of the state, the belief update collapses to belief over deformations since:

$$b(s) = b(x, y, \phi, \theta) = b(\psi, \theta) = \delta_\psi \cdot b(\theta)$$

where $\delta_{x,y,\phi}$ is 1 if the agent is in position (x, y) and orientation ϕ and 0 otherwise. By eq. (3.1) the belief update is then given by:

$$b^{a,o}(\psi', \theta') = \eta O(o \mid \psi', \theta') \delta_{\psi'} b(\theta) \tag{4.1}$$

4.1.1 Tests

Given the discrete nature of the deformations, we tested exact belief updates. Traditional Value Iteration and Perseus Infotaxis
Deep q-learning Discrete update of the belief state is performed by the agent at each time step.
STATES: 236196 ACTIONS: 4 OBSERVATIONS: 16
maze size 30x30

Algorithm	**Avg. Reward**	**Avg. Steps**
MDP (upper bound)	-18.454	18.188
QMDP	-24.715	23.75
TS	-28.01	25.75
MLS	-32.55	26.25

Table 4.1: Performance comparison of different algorithms on the Gridworld task. MDP represents the theoretical upper bound with perfect information.

4.2 Deformable Maze

The second task was structured just like the first one increasing the environment complexity. In this case we considered a continuous space and a continuous deformation parameter, keeping the discrete action space and observation space. This was done to provide a more insightful analysis of the possible scaling of the problem and techniques used to solve it.

State Space The environment is a continous 2D square of side length 1. the state is represented as a tuple (x, y, θ) where x and y are the Cartesian coordinates of the agent position, and θ is a parameter controlling the affine transformation:

$$M_\theta = \begin{bmatrix} \theta_{0,0} & \theta_{0,1} \\ \theta_{1,0} & \theta_{1,1} \end{bmatrix}$$

where the diagonal elements range is $0.4 \leq \theta_{ii} \leq 1$ and the off-diagonal elements range is $-0.2 \leq \theta_{i \neq j} \leq 0.2$. We assume perfect observability upon the agent's position while the deformable property of the environment has to be estimated through observations.

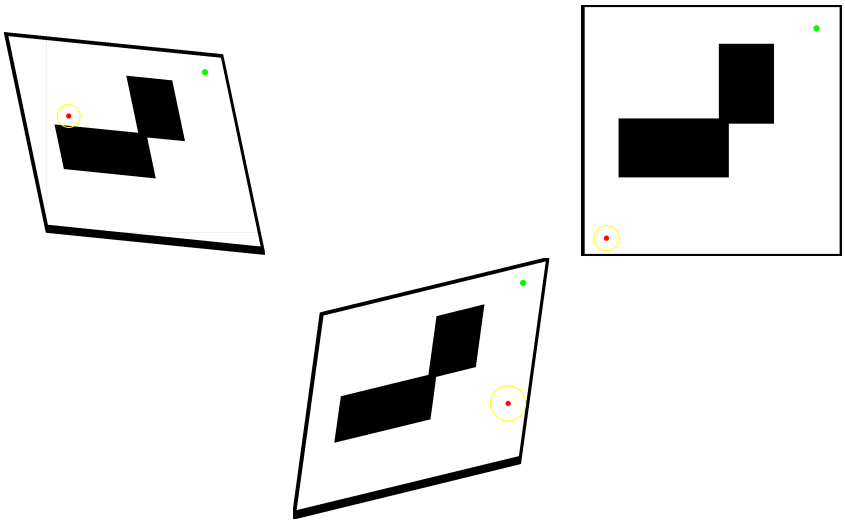


Figure 4.3: Maze Environment for different θ

Figure 4.3 shows the agent (red) that observes the presence of obstacles (black) inside its observtion radius (yellow) while trying to reach the goal (green).

Action Space Actions are the same as in the gridworld case, discrete movements of $stepsize = 0.02$ in the four cardinal directions.

Observation Space Observations are defined as elements of $O = \{\, o \in \{\, 0, 1 \,\}^4 \,\}$ where 0 refers to an empty cell and 1 to an obstacle. Obstacle presence is checked along cardinal directions N, S, E, W at any distance in the observation radius of the agent $r = 0.05$.

Transition Function Always assuming deterministic transitions, the transition function is defined as follows:

$$\mathcal{T}(s'|s,a) = \begin{cases} 1 & \text{if } s' \text{ is the result of applying action } a \text{ to state } s, \\ 0 & \text{otherwise.} \end{cases}$$

Analogous considerations to the gridworld case apply.

Reward Function The reward function $\mathcal{R}(s,a,s')$ is defined as follows:

$$\mathcal{R}(s,a,s') = \begin{cases} -1 & s' \neq s_{goal} \wedge \text{free cell}, \\ -2 & s' \neq s_{goal} \wedge \text{hit wall or outside bounds}, \\ 1 & s_{goal} \text{ in observation radius.} \end{cases}$$

Observation Function The conditional observation probabilities $O(o|s,a)$ are also deterministic:

$$O(o|s,a) = O(o|s) = \begin{cases} 1 & \text{if } (x,y) \text{ adjacent cells for map } f_\theta(M) \text{are compatible with } o, \\ 0 & \text{otherwise.} \end{cases}$$

4.2.1 Tests

Given the continuous nature of the environment, we didn't consider value iteration or Perseus. We tested PPO and DQN on the belief space without success
DQN - PPO Belief update: dicrete variational inference laplace approximation particle filters Different observation informativity.
TS con 1000 particles Mean episode Reward: -29.4585 Mean number of steps: 43.746

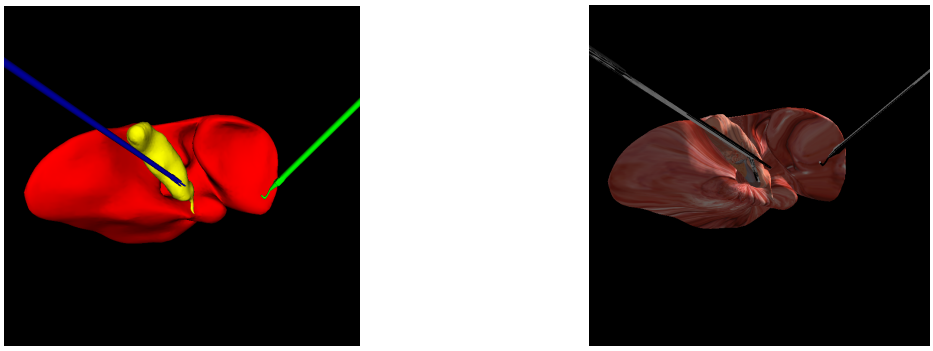
Algorithm	**Avg. Reward**	**Avg. Steps**
MDP (upper bound)	-27.8435	34.534
TS	-34.8965	46.342

Table 4.2: Performance comparison of different algorithms on the Maze task.

4.3 Surgical Task

We now move on to a different task that is more related to the surgical setting and deformable object manipulation.
The environment is based on Scheikl et al. (2023a) and models a sub-task from laparoscopic cholecystectomy, i.e., the minimally invasive removal of the gallbladder.

During dissection of the yellow gallbladder from the red liver, the blue grasper has to grasp the distal end (infundibulum) of the partially resected gallbladder. Afterwards, the grasper retracts the gallbladder, exposing a visual marker, which represents the point that should be cut next. The green electrocautery hook then navigates to the visual marker and activates in order to cut the tissue. The task is complete when the target is visible to the camera and the cauter activates while touching it (fig. 4.4). The task combines the challenges of grasping a deformable object, coordinating heterogeneous instruments, and performing multiple sequential steps to solve the task. Although this task models cholecystectomy, the task of exposing and then interacting with a region of interest often appears in surgical contexts.



(a) Segmented (b) Real

Figure 4.4: Surgical Environment

State Space The state space accounts for 59 real values, including Poses, TPSD states, jaw angle/activation state of instruments; current phase id; and 10 tracking points on the gallbladder. Target position is otherwise treated as unknown and has to be estimated through observations.

Action Space The action space is continous and represents the activation of the instruments.

Observation Space Observations are modelled as visual inputs from the camera, given the only source of uncertanty is the target position, we can assume perfect observability upon the instrument's position and orientation.

Reward Function Rewards are given based on the distance between the target and the instrument tip, the activation of the cauter, and the completion of sub-tasks and any other factors are considered in the reward function to model the surgical task; see **??**.

Transition Function Transition function wasn't considered in this case as getting it from the physics engine would be too complex and not necessary for the task at hand.

Observation Function

4.3.1 Tests

Belief space PPO

Chapter 5

Results and Future Directions

Simulation-based pretraining to reduce real-world data collection costs. Transfer learning from simpler robotic tasks (e.g., grasping) to more complex surgical tasks. Safety-aware RL methods (constrained policies) to minimize surgical risk. Domain adaptation techniques to handle variations in patient anatomy and real-time changes. Hybrid approaches that blend model-based and model-free RL for adaptive decision-making and uncertainty handling.

ASSUMPTIONS THROUHGOUT THE EXPERIMENTS PRIMI DUE ENV - deformation of the environment is the only source of uncertainty - the agent doesn't affect deformation - deformation are topologically sound LAST ENV

Bibliography

Daniele Amparore, Enrico Checcucci, Pietro Piazzolla, Federico Piramide, Sabrina De Cillis, Alberto Piana, Paolo Verri, Matteo Manfredi, Cristian Fiori, Enrico Vezzetti, and Francesco Porpiglia. Indocyanine green drives computer vision based 3d augmented reality robot assisted partial nephrectomy: The beginning of "automatic" overlapping era. *Urology*, 164:e312–e316, 2022. ISSN 0090-4295. doi: https://doi.org/10.1016/j.urology.2021.10.053. URL https://www.sciencedirect.com/science/article/pii/S0090429522000292.

Karl Johan Åström. Optimal control of markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10:174–205, 1965. URL https://api.semanticscholar.org/CorpusID:121222106.

Richard Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America*, 38(8):716–719, 1952. ISSN 00278424, 10916490. URL http://www.jstor.org/stable/88493.

Rokas Bendikas, Valerio Modugno, Dimitrios Kanoulas, Francisco Vasconcelos, and Danail Stoyanov. Learning needle pick-and-place without expert demonstrations. *IEEE Robotics and Automation Letters*, 8(6):3326–3333, 2023. doi: 10.1109/LRA.2023.3266720.

Anthony Cassandra, Michael L. Littman, and Nevin L. Zhang. Incremental pruning: a simple, fast, exact method for partially observable markov decision processes. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, UAI'97, page 54–61, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. ISBN 1558604855.

Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth AAAI National Conference on Artificial Intelligence*, AAAI'94, page 1023–1028. AAAI Press, 1994.

Anthony Rocco Cassandra. *Exact and approximate algorithms for partially observable markov decision processes*. PhD thesis, USA, 1998. AAI9830418.

Hsien-Te Cheng. *Algorithms for partially observable Markov decision processes*. PhD thesis, University of British Columbia, 1988. URL https://open.library.ubc.ca/collections/ubctheses/831/items/1.0098252.

Karthik Dharmarajan, Will Panitch, Muyan Jiang, Kishore Srinivas, Baiyu Shi, Yahav Avigal, Huang Huang, Thomas Low, Danyal Fer, and Ken Goldberg. Automating vascular shunt insertion with the dvrk surgical robot. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6781–6788, 2023. doi: 10.1109/ICRA48891.2023.10160966.

Alvin W Drake. *Observation of a Markov process through a noisy channel*. PhD thesis, Massachusetts Institute of Technology, 1962.

M. Hauskrecht. Value-function approximations for partially observable markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–94, August 2000. ISSN 1076-9757. doi: 10.1613/jair.678. URL http://dx.doi.org/10.1613/jair.678.

Milos Hauskrecht. Incremental methods for computing bounds in partially observable markov decision processes. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence*, AAAI'97/IAAI'97, page 734–739. AAAI Press, 1997. ISBN 0262510952.

Tommi Jaakkola, Satinder Singh, and Michael Jordan. Reinforcement learning algorithm for partially observable markov decision problems. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7. MIT Press, 1994. URL https://proceedings.neurips.cc/paper_files/paper/1994/file/1c1d4df596d01da60385f0bb17a4a9e0-Paper.pdf.

Ji Woong Kim, Tony Z. Zhao, Samuel Schmidgall, Anton Deguet, Marin Kobilarov, Chelsea Finn, and Axel Krieger. Surgical robot transformer (srt): Imitation learning for surgical tasks, 2024. URL https://arxiv.org/abs/2407.12998.

Vikram Krishnamurthy. *Partially Observed Markov Decision Processes: From Filtering to Controlled Sensing*. Cambridge University Press, 2016.

Matthias Lehmann. The definitive guide to policy gradients in deep reinforcement learning: Theory, algorithms and implementations, 2024. URL https://arxiv.org/abs/2401.13662.

Michael L. Littman. The witness algorithm: Solving partially observable markov decision processes. Technical report, USA, 1994.

Shiqi Liu, Yan Zhang, and Shurui Fan. Adaptive space-aware infotaxis ii as a strategy for odor source localization. *Entropy*, 26(4), 2024. ISSN 1099-4300. doi: 10.3390/e26040302. URL https://www.mdpi.com/1099-4300/26/4/302.

William S. Lovejoy. A survey of algorithmic methods for partially observed markov decision processes. *Ann. Oper. Res.*, 28(1–4):47–66, June 1991. ISSN 0254-5330. doi: 10.1007/BF02055574. URL https://doi.org/10.1007/BF02055574.

P. Marbach and J.N. Tsitsiklis. Simulation-based optimization of markov reward processes. *IEEE Transactions on Automatic Control*, 46(2):191–209, 2001. doi: 10.1109/9.905687.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016. URL https://arxiv.org/abs/1602.01783.

George E. Monahan. State of the art—a survey of partially observable markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1): 1–16, 1982. doi: 10.1287/mnsc.28.1.1. URL https://doi.org/10.1287/mnsc.28.1.1.

Yafei Ou and Mahdi Tavakoli. Sim-to-real surgical robot learning and autonomous planning for internal tissue points manipulation using reinforcement learning. *IEEE Robotics and Automation Letters*, 8(5):2502–2509, 2023. doi: 10.1109/LRA.2023.3254860.

Yafei Ou, Abed Soleymani, Xingyu Li, and Mahdi Tavakoli. Autonomous blood suction for robot-assisted surgery: A sim-to-real reinforcement learning approach. *IEEE Robotics and Automation Letters*, 9(8):7246–7253, 2024. doi: 10.1109/LRA.2024.3421191.

Alberto Piana, Daniele Amparore, Michele Sica, Gabriele Volpi, Enrico Checcucci, Federico Piramide, Sabrina De Cillis, Giovanni Busacca, Gianluca Scarpelli, Flavio Sidoti, Stefano Alba, Pietro Piazzolla, Cristian Fiori, Francesco Porpiglia, and Michele Di Dio. Automatic 3d augmented-reality robot-assisted partial nephrectomy using machine learning: Our pioneer experience. *Cancers*, 16(5), 2024. ISSN 2072-6694. doi: 10.3390/cancers16051047. URL https://www.mdpi.com/2072-6694/16/5/1047.

Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: an anytime algorithm for pomdps. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI'03, page 1025–1030, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc.

Herbert Robbins and Sutton Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400 – 407, 1951. doi: 10.1214/aoms/1177729586. URL https://doi.org/10.1214/aoms/1177729586.

Paul Maria Scheikl, Balázs Gyenes, Rayan Younis, Christoph Haas, Gerhard Neumann, Martin Wagner, and Franziska Mathis-Ullrich. Lapgym: an open source framework for reinforcement learning in robot-assisted laparoscopic surgery. *J. Mach. Learn. Res.*, 24(1), January 2023a. ISSN 1532-4435.

Paul Maria Scheikl, Eleonora Tagliabue, Balázs Gyenes, Martin Wagner, Diego Dall'Alba, Paolo Fiorini, and Franziska Mathis-Ullrich. Sim-to-real transfer for visual reinforcement learning of deformable object manipulation for robot-assisted surgery. *IEEE Robotics and Automation Letters*, 8(2):560–567, 2023b. doi: 10.1109/LRA.2022.3227873.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017a. URL https://arxiv.org/abs/1502.05477.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017b. URL https://arxiv.org/abs/1707.06347.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018. URL https://arxiv.org/abs/1506.02438.

Amin Abbasi Shahkoo and Ahmad Ali Abin. Deep reinforcement learning in continuous action space for autonomous robotic surgery. *International Journal of Computer Assisted Radiology and Surgery*, 18(3):423–431, 2023. ISSN 1861-6429. doi: 10.1007/s11548-022-02789-8. URL https://doi.org/10.1007/s11548-022-02789-8.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017. ISSN 1476-4687. doi: 10.1038/nature24270. URL https://doi.org/10.1038/nature24270.

Richard D. Smallwood and Edward J. Sondik. The optimal control of partially observable markov processes over a finite horizon. *Operations Research*, 21(5):1071–1088, 1973. ISSN 0030364X, 15265463. URL http://www.jstor.org/stable/168926.

Edward J. Sondik. The optimal control of partially observable markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2):282–304, 1978. ISSN 0030364X, 15265463. URL http://www.jstor.org/stable/169635.

M. T.J. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for pomdps. *Journal of Artificial Intelligence Research*, 24:195–220, August 2005. ISSN 1076-9757. doi: 10.1613/jair.1659. URL http://dx.doi.org/10.1613/jair.1659.

Matthijs T. J. Spaan. Partially observable Markov decision processes. In Marco Wiering and Martijn van Otterlo, editors, *Reinforcement Learning: State of the Art*, pages 387–414. Springer Verlag, 2012.

Richard S. Sutton. Learning to predict by the methods of temporal differences. *Mach. Learn.*, 3(1):9–44, August 1988. ISSN 0885-6125. doi: 10.1023/A:1022633531479. URL https://doi.org/10.1023/A:1022633531479.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

Vignesh Manoj Varier, Dhruv Kool Rajamani, Nathaniel Goldfarb, Farid Tavakkolmoghaddam, Adnan Munawar, and Gregory S Fischer. Collaborative suturing: A reinforcement learning approach to automate hand-off task in suturing for surgical robots. In *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1380–1386, 2020. doi: 10.1109/RO-MAN47096.2020.9223543.

Massimo Vergassola, Emmanuel Villermaux, and Boris I Shraiman. 'infotaxis' as a strategy for searching without gradients. *Nature*, 445(7126):406—409, January 2007. ISSN 0028-0836. doi: 10.1038/nature05464. URL https://doi.org/10.1038/nature05464.

Christopher Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992. URL https://api.semanticscholar.org/CorpusID:208910339.

Marco Wiering and Martijn Van Otterlo. *Reinforcement Learning: State of the Art*. Springer, 2012. ISBN 978-3-642-27644-6. doi: 10.1007/978-3-642-27645-3.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

K.J Åström. Optimal control of markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10(1):174–205, 1965. ISSN 0022-247X. doi: https://doi.org/10.1016/0022-247X(65)90154-X. URL https://www.sciencedirect.com/science/article/pii/0022247X6590154X.