UNIVERSITY OF TRIESTE

Department of Mathematics, Informatics and Geoscience

Master's Degree in
Data Science & Scientific Computing

Reinforcement Learning applications to Minimally invasive Robotic Surgery

trova titolo decente

February 18, 2025

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

A.Y. 2023/2024

Abstract

Lung cancer remains a leading cause of cancer-related deaths worldwide, with surgical resection offering the best chance for cure in early-stage cases. However, the intraoperative environment presents significant challenges, including dynamic anatomical changes, limited visibility, and high stakes for patient safety. Assistive technologies capable of real-time decision-making and adaptability could greatly enhance surgical outcomes. In this context, this thesis explores the potential of Partially Observable Markov Decision Processes and Reinforcement Learning as frameworks for developing intelligent, adaptive, and safe intraoperative AI driven systems to support surgeons in Robotic Assisted Surgery.

Thesis Outline and Summary

There are two major areas that contribute to the advancement of automating surgical procedures: perception and task automation. Perception aims at understanding the surgical situation based on a variety of inputs during the surgery (endoscopic cameras, tactile feedback etc). This may also include the detection, and tracking of surgical tools, as well as 3D reconstruction of the deformed tissue. Task automation, on the other hand, involves automating the surgical robot to accomplish specific tasks, given the knowledge of the surgical situation.
Our work focuses on brigding the gap between these two areas, by developing a framework that integrates perception and task automation in a single pipeline through the use of Partially Observable Markov Decision Processes (POMDPs) and Reinforcement Learning (RL).

Contents

List of Figures

List of Tables

Acronyms

AI Artificial intelligence.

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.

RATS robotic-assisted thoracic surgery.

RL Reinforcement Learning.

TD Temporal Difference.

VATS video-assisted thoracic surgery.

Chapter 1

Introduction

Artificial intelligence (AI) applications in robotic surgery represent a transformative frontier in the medical field, enhancing precision, safety, and efficiency in surgical procedures. The integration of such technologies with robotic systems has significantly advanced 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.

Notable robotic systems, such as the da Vinci Surgical System, exemplify the evolution of surgical practice through this innovative marriage of AI and robotics, paving the way for improved patient outcomes across various specialties, including urology, gynecology, and cardiothoracic surgery.

AI enhances several aspects of surgical practice, including preoperative planning, real-time decision support, and postoperative analysis, thereby transforming the surgical landscape.

These advancements underscore AI's potential to refine surgical techniques and foster a culture of continuous improvement in surgical care. As the field progresses, the future of AI in robotic surgery promises further innovations, such as remote surgical capabilities and enhanced imaging technologies, which could redefine healthcare access and delivery on a global scale.

The intersection of AI and robotic surgery not only marks a significant evolution in surgical practices but also holds the potential to fundamentally reshape patient care and outcomes in the coming years. Reinforcement Learning (RL) in thoracic microinvasive surgery represents a pioneering intersection of artificial intelligence and surgical practice, aiming to enhance surgical precision and decision-making processes. As robotic-assisted surgical techniques gain traction, particularly through methods like video-assisted thoracic surgery (VATS) and robotic-assisted thoracic surgery (RATS), the integration of RL algorithms into these workflows has become notable for its potential to transform patient outcomes. This application is increasingly relevant as the demand for minimally invasive approaches grows, driven by their associated benefits such as reduced postoperative pain and shorter recovery times.

The development of RL techniques in this field leverages the ability of these algorithms to learn from complex interactions within dynamic surgical 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 thoracic microinvasive surgery faces challenges, including the need for robust regulatory frameworks and ongoing ethical scrutiny. Discussions surrounding the implications of AI in surgical contexts emphasize the importance of ensuring that healthcare providers remain central in the decision-making process while leveraging AI technologies to enhance patient care.

Overall, the ongoing research and development in reinforcement learning for thoracic microinvasive 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

Over the last decade, 3D models have entered oncologic surgery as a means to achieve better outcomes in renal and hepatic surgery [1, 2]. Nevertheless, the integration of 3D models into the operative field has been lacking due to three main reasons. Firstly, proper model alignment with the intraoperative anatomy has proven to be a major challenge due to shifting of organs during surgery and different patient positioning in surgery ver- sus during computed tomography [2, 3]. Secondly, automated organ registra-tion in a continuously moving surgical video has been another major challenge for many years [4]. Thirdly, 3D model overlay obscures the surgical field, including sharp surgical instruments which are manipulated, hence creating a possible hazardous situation rather than facilitating surgery. The latter occlusion problem has been a longstanding study topic [5] which, if solved, would further advance various surgical domains and applications [6]. Already in 2004, Fischer et al. [7]

Navigation in deformable environments

1.2 Related Work

- Instrument occlusion
- DL recognition of anatomical structures
- Elastic Regristration
- RL for simple surgical tasks

While robotic systems have revolutionized minimally invasive surgery, they lack the ability to adapt to unexpected intraoperative changes autonomously. Partially Observable Markov Decision Process (POMDP)s, with their capacity to model uncertainty, and RL, with its learning-based adaptability, present a promising synergy for addressing these challenges. However, their application to real-time intraoperative decision-making remains unexplored.

RL in Surgical Robotics **preso da** Ou et al. 2024 ALarge number of recent research in surgical robotics focuses on the automation of surgical sub-tasks, such as needle manipulation [1], [2], [3], suturing [4], [5], cutting [6], [7], [8], vessel manipulation [9], tissue retraction and deformation [10], [11], [12], [13]. However, most of these studies focus on the manipulation of rigid and soft objects, although fluid-related tasks are also common in surgeries, due to the presence of body fluids, especially blood

Part I

Theoretical Background

Chapter 2

Reinforcement Learning

controlla itemize

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 Process (MDP)s is mostly based on Wiering and Van Otterlo 2012 and Sutton and Barto 2018

Chapter 3

Markov Decision Processes

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 given time) of the problem that is modelled. 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 $s \in S$.

Trasition 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 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 $T(s, a, s')$. It is required that for all actions a, and all states s and s',

$T(s, a, s') \geq 0$ and for all states s and actions a,

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

such that 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 araise 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 R is usually defined as

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

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

With the above components, we can define a MDP as in 3.1 .

Definition 3.0.1. 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 $\mathcal{T} : S \times A \times S \rightarrow [0, 1]$
- $\mathcal{R}$ is the reward function $\mathcal{R} : S \times A \times S \rightarrow \mathbb{R}$

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

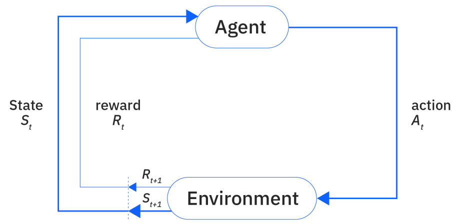

Figure 3.1: Markov Decision Process

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 an ordinary deterministic function of four arguments and the '|' in the middle of it comes from the notation for conditional probability, note that the following holds:

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

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

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

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

these formulations are useful for the analysis of the MDP and the development of algorithms. In such context, the goal of the agent is to learn an optimal behaviour. Thus, let's define behaviours and optimality.

3.1 Policy and Value Function

Policy The formalization of the naive concept of behaviour 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 $\sum_{a\in A} \pi(a|s) = 1$. Such policy represents 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.

add non-stationary definition of policy

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 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 infinite horizon problems.

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

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

$$\begin{aligned}
&= \sum_a \pi(a|s)\sum_{s'}\sum_r\sum_{g_{t+1}} p(s', r|a, s)p(g_{t+1}|s', r, a, s)(r + \gamma g_{t+1}) \\
&= \sum_a \pi(a|s)\sum_{s'}\sum_r p(s', r|a, s)\sum_{g_{t+1}} p(g_{t+1}|s')(r + \gamma g_{t+1}) \\
&= \sum_a \pi(a|s)\sum_{s'}\sum_r p(s', r|a, s)(r + \gamma\sum_{g_{t+1}} p(g_{t+1}|s')g_{t+1}) \\
&= \sum_a \pi(a|s)\sum_{s'}\sum_r p(s', r|a, s)\left(r + \gamma V^\pi(s')\right)
\end{aligned} \tag{3.1}$$

Eq 3.1 is a famous recoursive relationship know as the *Bellman equation* for V^π, forming the basis to learn, compute and approximate the value function.
Similarly one can define the state-action value function, giving 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 thereafter following policy π :

$$\begin{aligned} Q^\pi(s,a) &\doteq \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right] \\ &= \mathbb{E}_\pi\left[R_t + \gamma V^\pi(S_{t+1}) | S_t = s, A_t = a\right] \end{aligned} \tag{3.2}$$

it follows naturally that

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

3.2 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), there always[1] is a deterministic, stationary (time independent) 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

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

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

[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 it is required $\gamma \in (0,1)$, and costs (negative rewards) to be bounded $|c(s,a)| < M$ for all s and a in a continous setting

$$
\begin{aligned}
V^*(s) &= \max_{a \in A(s)} Q^{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma V^*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma V^*(s') \right]. \qquad (3.3)
\end{aligned}
$$

H (for MDPs) is an isotone mapping and that it is a contraction under the supremum norm (see (Heyman Sobel, 1984; Puterman, 1994)). An analogous result holds for the optimal state-action value function:

isotone why? non mi sembra che puterman lo richieda

$$
\begin{aligned}
Q^*(s,a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a] \\
&= \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma \max_{a'} Q^*(s', a') \right]. \qquad (3.4) \\
&= \sum_{s',r} p(s', r|s, a) \left[r + \gamma V^*(s') \right] \qquad (3.5)
\end{aligned}
$$

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

3.3 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 upside to having a model is that it allows the agent to plan by thinking ahead, seeing what would happen for a range of possible choices, and explicitly deciding between its options. Agents can then distill the results from planning ahead into a learned policy. A particularly famous example of this approach is AlphaZero.Silver et al. 2017 When this works, it can result in a substantial improvement in sample efficiency over methods that don't have a model. Achiam 2018

da spinningup

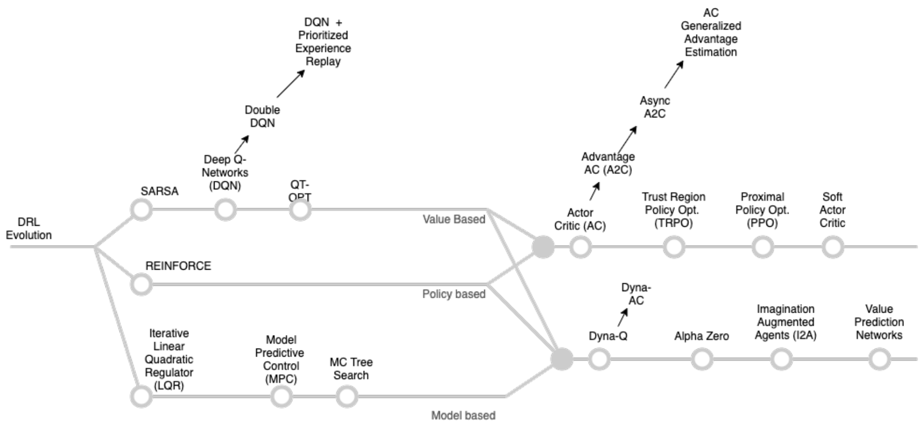

Figure 3.2: Evolution of Deep Reinforcement Learning Algorithms

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. Modern solution methods - entirely based on Deep Reinforcement Learning (DRL) - presented in fig. 3.2, rely on Neural Networks to approximate the value function, the policy, or both.

Before diving into these methods, we will introduce some of the most common and foundational algorithms used to solve MDPs that are the building blocks of more advanced techniques.

3.4 Dynamic Programming

Dynamic Programming (DP) is a class of algoritms of little practical use, nontheless 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. Two main steps are involved: policy evaluation and policy improvement.

Policy Evaluation Recall that policy evaluation or prediction is the task of determining the value function of a given policy. If the environment's dynamics are known, equation 3.1 represents a linear system in the $|S|$ unknowns $V^{\pi}(s)$, which can be solved exactly or iteratively. Due to the computational burden of exact solutions, iterative methods are 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 $\gamma < 1$.

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 $Q^\pi(s, a)$ as in equation [manca cit equation]: 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'.

$$\begin{aligned}
\pi'(s) &\doteq arg\max_a Q^\pi(s, a) \\
&= \mathbb{E}[R_{t+1} + \gamma V^\pi(S_{t+1}) \mid S_t = s, A_t = a] \\
&= arg\max_a \sum_{s',r} p(s', r\,|s, a)\Big[r + \gamma V^\pi(s')\Big]
\end{aligned} \tag{3.6}$$

This is motivated by the following theorem:

Theorem 3.4.1. *(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.

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

□

Policy improvement creates a new policy that enhances an initial policy by adopting a greedy approach based on the value function. Assuming π' is equally effective as π but not superior, then $V^{\pi} = V^{\pi'}$ ensures that for all states $s \in \mathcal{S}$.
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. The proof sketch is as follows. We consider k such that $V^{\pi_{k+1}}(s) = V^{\pi_k}(s), \forall s \in S$. We can show that such V^{π_k} satisfies the Bellman optimality equation [3.3], and hence $V^{\pi_k} = V^*$.

3.4.1 Policy Iteration

3.4.2 Value Iteration

Theorem 6.2.3. of puterman 1994 serve a provare la convergenza assieme a banach fixed point theorem e dimostrazione che l'operatore di bellman soddisfa le proprietà di contrazione e (isotonica?)

3.4.3 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.
The policy evaluation step estimates the utility of the current policy, that is, it computes V^{π}, directly through the use of the model (if available) or iteratively by sampling trajectories interacting with the environment
The policy improvement step. This step computes an improved policy from the current one using the information in V.
Both the evaluation and the improvement steps can be implemented in various ways, and interleaved in several distinct ways. The bottom line 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.

3.5 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 experience - actual or simulated[2] - in the form of sample sequence of states, actions and rewards. Following

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

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

Policy Evaluation The most simple 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 wrt V, MC methods are therefore used to estimate the action-value function $Q_\pi(s, a)$ for control tasks. 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 While Policy Evaluation and Improvement, 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 thanks to the *Policy Improvement Theorem* 3.4.1.
Both assumptions can actually be relaxed by giving up on deterministic policies and only search over ϵ-soft policies and by using allowing the value function to be updated after each episode and not when convergence is met.

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

3.6 Temporal Difference Learning

Combining the ideas of Dynamic Programming (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.

3.7 Policy Gradient Methods

Chapter 4

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. This is not always the case, and in many real-world scenarios, the agent has only partial information about the state of the environment. As MDPs are controlled Markov Chains, POMDPs are controlled Hidden Markov Models 4.1.

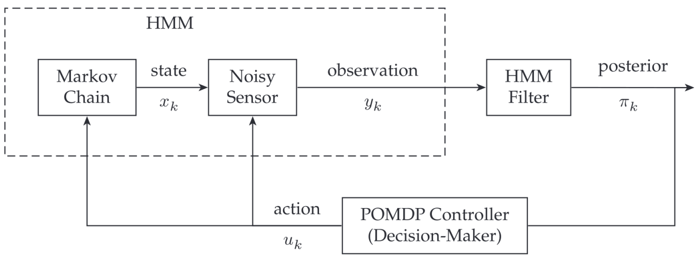

Figure 4.1: Partially Observable Markov Decision Process Krishnamurthy 2016

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 M. T. J. Spaan 2012 and Hauskrecht 2000 analysis of POMDPs, originally introduced by Drake, 1962; Astrom, 1965; Sondik, 1971; Lovejoy, 1991b. aggiusta citazioni

Definition 4.0.1. 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 4.0.1. the need for memory (Singh et al, 1994).

In the next sections we will introduce the main concepts and algorithms used to solve POMDPs.

4.1 Belief State MDPs

Since in POMDPS the underliyng 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 4.1.1. (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 mapping of information states to actions (possibly distributions over actions). The new information state I_{t+1} is obtained 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 accordingly

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

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

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 4.1.2. (Sufficient information state process). Let $\mathcal{I}$ be an information state space and $\tau : \mathcal{I} \times A \times \Theta \rightarrow \mathcal{I}$ 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 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 citestrm1965OptimalCO.
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 Baye's rule.

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

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 an 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

4.2 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 as the value function $V^\pi : \Delta(S) \rightarrow \mathbb{R}$. In the infinite-horizon discounted case the value function is defined as:

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

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

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

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

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

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 parametrized by a finite set of vectors, the following result is originally from Smallwood and Sondik 1973.

Theorem 4.2.1. *(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 $V_0 = \sum_s R(S,a)b(s) = \sum_i b_i\alpha_0^{i}(s) = b\cdot\alpha_0$ Assuming the hypothesis holds for V_n, we can show that it holds for V_{n+1} as well.

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

denoting

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

and

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

therefore one can express the value function as stated.

□

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

$$backup(b) = \arg\max_{g_a^b} b \cdot g_a^b \tag{4.1}$$

Convexity of the value function the expected discounted reward generally grows as the entropy of b decreases 4.2.
Even though the optimal value function for an infinite-horizon POMDP is convex, it may not be piecewise linear. Nontheless, it can be approximated arbitrarily closely by a piecewise linear and convex value function.
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 $|O|$).

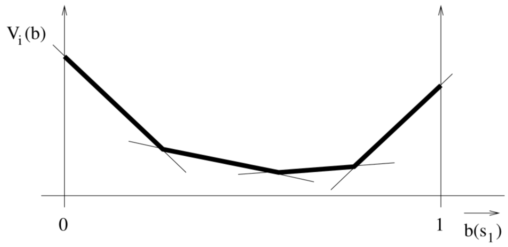

Figure 4.2: Convex Value function for a two states POMDP.

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

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

defined over the belief space.
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 $|S| \times |V|$ 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 $|S| \times |V|$ for finding the best action when following V.

4.3 Exact Value Iteration

The complete set of linear functions is rarely needed: some of the linear functions are dominated by others therefore not affecting the value function. This approach - Sondik (1971) and Monahan (1982) - enumerates all possible linear functions and then remove (prune) all redundant vectors.

aggiusta O()

This process is known as exact value iteration. In each iteration, the value function is updated across the entire belief space. There are $|V| \times |A| \times |\Omega|$ vectors generated at Eq. 21, and computing each of these vectors takes $|S|^2$ operations. In Eq. 19 we create $|V|^{|\Omega|}$ new vectors for each action, with a complexity of $|S|$ for each new vector. Hence, the overall complexity of a single iteration is $O\left(|V| \times |A| \times |\Omega| \times |S|^2 + |A| \times |S| \times |V|^{|\Omega|}\right)$.

Monahan's Enumeration Algorithm

Incremental Pruning An alternative approach builds on Sondik's idea of computing a useful linear function for a single belief state (Sondik, 1971; Smallwood Sondik, 1973), which can be done efficiently. The key problem here is to locate all belief points that seed useful linear functions. Methods that implement this idea are Sondik's one- and two-pass algorithms (Sondik, 1971), Cheng's methods (Cheng, 1988), and the Witness algorithm (Kaelbling, Littman, Cassandra, 1999; Littman, 1996; Cassandra,1998).

4.4 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, Gordon, and Thrun 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 a α-vector for each $b \in B$, and repeatedly updating (via value backups) the value of that α-vector, preserving the piece-wise linearity and convexity of the value function, and defining value function over the entire belief simplex.

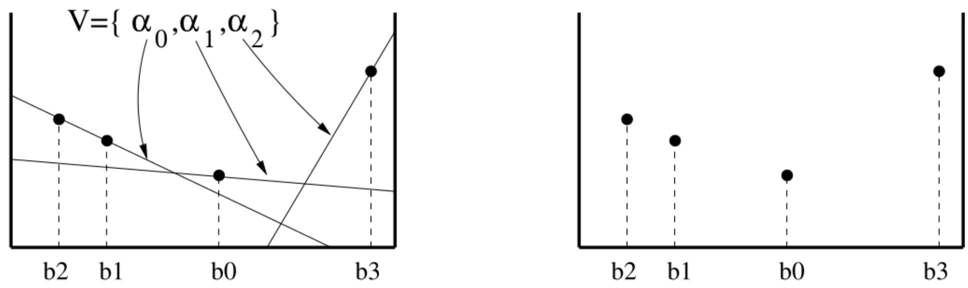

Figure 4.3: Point-Based Value Iteration

The point Based update follows exactly Eq. 4.1 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 infinte 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 Bi , 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

The complexity of computing $g^i_{a,o}$ is $O\left(|S|^2\right)$, and it is done for every $\alpha \in V$, hence computing all $g^{a,o}$ requires $O\left(|A| \times |\Omega| \times |V| \times |S|^2\right)$. Computing α^b_a (Eq. 34) requires the computation of all the relevant $\alpha^{a,o}$, but then the summation and inner products require only $O(|S| \times |\Omega|)$ operations and another $O(|S|)$ operations for adding the reward vector. Finally, the backup operation requires for each α^b_a another $O(|S|)$ operations for the inner product. Hence, the full complexity of the point-based backup requires $O(|A| \times |\Omega| \times |V| \times |S|^2 + |A| \times |S| \times |\Omega|)$.

However, a full backup for the subset B will not require $|B|$ times the complexity of a single point-based backup, because the $\alpha^{a,o}$ 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 $\alpha^{a,o}$ only once and cache the result, requires only $O(|A| \times |\Omega| \times |V| \times |S|^2 + |B| \times |A| \times |S| \times |\Omega|)$, 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.

Proof of convergence bounds Pineau, Gordon, and Thrun 2003

4.4.1 Perseus

M. T. Spaan and Vlassis 2005 This scheme is a heuristic to let Bt cover a wide area of the belief space, but comes at a cost as it requires computing distances between all b Bt . By backing up all b Bt the PBVI algorithm generates at each stage approximately vectors, which can lead to slow performance in domains requiring large Bt . In the next section we will present a point-based POMDP value iteration method which does not require backing up all b B. We compute backups for a subset of B only, but seeing to it that the computed solution will be effective for the complete set B. As a result we limit the growth of the number of vectors in the successive value function estimates, leading to significant speedups.

4.5 Heuristics Approximations

4.5.1 QMDP

4.5.2 action voting

4.5.3 most likely state

4.5.4 entropy heuristics

Cassandra [1998] shows how to use the entropy of b to switch between information gathering policies and exploitive policies. The entropy can also be used to weight two policies that trade off information gathering and exploitation. These heuristics may be misleading if the minimum of (b) does not occur at the point of greatest entropy, that is, the uniform belief state.

4.6 Model Free et al

2. Monte Carlo and Sampling-Based Methods
These approaches approximate belief updates or policies using stochastic simulations.
Monte Carlo Tree Search (MCTS): Expands belief nodes based on sampled trajectories (e.g., POMCP – Partially Observable Monte Carlo Planning). DESPOT (Determinized Sparse Partially Observable Trees): Uses sparse sampling to reduce tree expansion complexity.
3. Policy-Based Approximation
Instead of solving for a full value function, these methods approximate policies directly.
Finite-State Controllers (FSCs): Compact representation of policies as a finite-state machine. Gradient-Based Policy Search: Uses reinforcement learning techniques to optimize policies.
4. Online Solvers
Rather than solving the entire problem offline, these solvers focus on finding good actions given the current belief.
POMCP (Partially Observable Monte Carlo Planning): Uses Monte Carlo sampling with an upper confidence bound exploration strategy. ABT (Adaptive Belief Tree): Builds a belief tree dynamically using online lookahead search.
5. Function Approximation Methods
These methods use machine learning techniques to approximate value functions or policies.

Deep Reinforcement Learning (DRL) for POMDPs: Uses neural networks to approximate value functions or policies. Deep Q-Networks (DQN) with Recurrent Layers: Uses LSTMs to track hidden states in partially observable environments.

Part II

Experiments

Chapter 5

Methods

We adopted a bottom up approach, increasing the complexity of the task in increasingly complex environments. Recall that our objective was to explore the applicability of RL to navigation in deformable environment related to the surgical task of thoracic surgery. We therefore adopted a mixed approach assuming the model of the environment to be known or learnable, given the preoperative imaging data available in the clinical setting. This would allow us to model the problem as a POMDP and apply the methods described in the previous sections.

5.1 Gridworld

The first task we considered was 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 state space is a 2D grid of size 10×10 where each cell can be either empty or occupied by an obstacle. the state is represented as a tuple (x, y, ϕ) where x and y are the coordinates of the agent in the grid. and ϕ is the orientation of the agent.

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.
The **conditional observation probabilities** $O(o|s, a)$ are also deterministic.

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

Reward Function The **reward function** $R(s, a, s')$ is defined as follows:

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

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

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

Observation Function

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 belief function is a probability distribution over the states of the environment.

$$b : S \to [0,1] \text{ and } \sum_s b(s) = 1$$

By interacting with the environment and receiving observations, the agent may update its belief in the true state by updating the probability distribution of the current state

$$b'(s') = \eta O(o \mid s', a) \sum_{s \in S} T(s' \mid s, a) b(s)$$

where $\eta = \frac{1}{Pr(o|b,a)}$ is a normalizing constant with

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

Discrete update of the belief state is done by the agent at each time step.

5.2 Deformable Maze

State Space

Action Space

Observation Space

Reward Function

Transition Function

Observation Function

5.3 Surgical Task

State Space

Action Space

Observation Space

Reward Function

Transition Function

Observation Function

Chapter 6

Results

Presentation of results... GraspLiftAndTouchEnv models a sub-task from laparoscopic chole- cystectomy, i.e., the minimally invasive removal of the gallbladder. Dur- ing dissection of the yellow gallbladder from the red liver, the blue grasper has to grasp the distal end (infundibulum) of the partially re- sected 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.

Chapter 7

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.

Bibliography

Achiam, Joshua (2018). “Spinning Up in Deep Reinforcement Learning”. In.

Bellman, Richard (1952). “On the Theory of Dynamic Programming”. In: *Proceedings of the National Academy of Sciences of the United States of America* 38.8, pp. 716–719. ISSN: 00278424, 10916490. URL: http://www.jstor.org/stable/88493 (visited on 02/11/2025).

Hauskrecht, M. (Aug. 2000). “Value-Function Approximations for Partially Observable Markov Decision Processes”. In: *Journal of Artificial Intelligence Research* 13, pp. 33–94. ISSN: 1076-9757. DOI: 10.1613/jair.678. URL: http://dx.doi.org/10.1613/jair.678.

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

Ou, Yafei et al. (2024). “Autonomous Blood Suction for Robot-Assisted Surgery: A Sim-to-Real Reinforcement Learning Approach”. In: *IEEE Robotics and Automation Letters* 9.8, pp. 7246–7253. DOI: 10.1109/LRA.2024.3421191.

Pineau, Joelle, Geoff Gordon, and Sebastian Thrun (2003). “Point-based value iteration: an anytime algorithm for POMDPs”. In: *Proceedings of the 18th International Joint Conference on Artificial Intelligence*. IJCAI’03. Acapulco, Mexico: Morgan Kaufmann Publishers Inc., pp. 1025–1030.

Silver, David et al. (2017). “Mastering the game of Go without human knowledge”. In: *Nature* 550.7676, pp. 354–359. ISSN: 1476-4687. DOI: 10.1038/nature24270. URL: https://doi.org/10.1038/nature24270.

Smallwood, Richard D. and Edward J. Sondik (1973). “The Optimal Control of Partially Observable Markov Processes Over a Finite Horizon”. In: *Operations Research* 21.5, pp. 1071–1088. ISSN: 0030364X, 15265463. URL: http://www.jstor.org/stable/168926 (visited on 02/11/2025).

Spaan, M. T.J. and N. Vlassis (Aug. 2005). “Perseus: Randomized Point-based Value Iteration for POMDPs”. In: *Journal of Artificial Intelligence Research* 24, pp. 195–220. ISSN: 1076-9757. DOI: 10.1613/jair.1659. URL: http://dx.doi.org/10.1613/jair.1659.

Spaan, Matthijs T. J. (2012). "Partially Observable Markov Decision Processes". In: *Reinforcement Learning: State of the Art*. Ed. by Marco Wiering and Martijn van Otterlo. Springer Verlag, pp. 387–414.

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

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

Chapter A

Appendix A

A.1 Operators and Contractions