Situated Decision-Making in Uncertain Environments

Matthew Budd

Pembroke College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Hilary 2025

Acknowledgements

To start, I am sincerely grateful for the support and guidance from my supervisors, Nick and Bruno. As well as being an enthusiastic mentor, Nick has consistently sourced interesting and challenging projects to explore throughout the DPhil. Bruno has always been generous with his time and support, and greatly strengthened every publication with his careful review. I'd also like to thank Paul for his supervision and contributions during my Master's and the first half of my DPhil. I very much appreciate the time and care my examiners, Stephen Roberts and Zachary Sunberg, took to read my thesis and provide detailed feedback.

GOALS is a fantastic environment to work in, and it's been great to learn from, and be friends with, so many talented people. I'd like to thank Charlie, Marc, Michael, Anna, Mohamed, Alex St, Lara, Alex R, Carl, Michal, Branton, Alex Sc, Roland and An. Thanks also to Ricardo, Kim Tien and my other Team ORIon compatriots.

I am grateful to Prof. Alice Cicirello and Prof. Daniel Eakins for introducing me to the world of engineering research. This, and a thoroughly enjoyable Master's research project with GOALS, were the catalysts for embarking on a DPhil.

I'd like to thank the Decision & Control group at Ericsson, and Dr. Magnus Lindhé and Dr. Fernando dos Santos Barbosa in particular, for an excellent internship and a friendly introduction to a new city. I'd also like to thank Amazon Web Services (AWS) for supporting my DPhil with a generous scholarship and AWS compute credits.

Thank you especially to Tom and Gill for your full support and belief in me, and for always being there to help. Thank you to Rory and Lachlan for being not just great brothers, but also great friends. Thank you, of course, to Pumpkin the cat for her company and general administrative assistance from almost the very beginning of the DPhil. Finally, a huge thank you to Catherine for helping me keep things in perspective when under pressure, and for being a constant source of support and encouragement. I'm sure the next few years will be just as exciting and rewarding as the journey so far.

Abstract

Mobile robots are highly suited to large-scale data-gathering, monitoring and exploration tasks in challenging and poorly understood environments such as oceans and hazardous unmapped areas. To operate reliably in these settings, they need decision-making methods that can function under a lack of knowledge about the environment. This lack of knowledge is known as epistemic uncertainty. A robot situated in such an environment faces several challenges: it has only local information about its environment, its actions may be affected by unknown environment dynamics such as water currents, and it may only have access to its own on-board computational resources and sensors.

In this thesis, we aim to answer the question of how an autonomous system such as a mobile robot can make the best possible decisions, given its bounded computational resources and limited knowledge of its deployment environment. We do this by contributing new decision-making methods for control of robot actions, and new metareasoning methods for control of reasoning processes.

In one strand of work, we design three new decision-making methods for mobile robots operating in epistemically uncertain environments. These range from an *offline* method that pre-plans robot policies, to *online* methods that better adapt to the environment but require more computational resources. We demonstrate our methods in domains including underwater and hazardous environment mobile robotics, and two real field deployments. In each case our algorithms achieve their goals while incurring less cost than previous approaches, or are able to tackle problem settings that previous approaches could not.

In an orthogonal research direction, we develop learning-based metareasoning methods that automatically adapt a robot's decision-making strategy to the deployment environment, the robot's reasoning capabilities, and the problem at hand. We do this by proposing two metareasoning frameworks, which apply to offline and online decision-making methods respectively. We evaluate our methods in environments specifically designed to test how metareasoning improves decision-making in situated systems, and demonstrate a significant performance increase over fixed strategies and prior metareasoning methods.

Contents

1 Introduction

Mobile robots offer the promise of revolutionising scientific, industrial and commercial fields over the coming decades, and are therefore being deployed in increasingly unstructured and uncertain environments. Advancements in tasks such as delivery logistics (Srinivas et al., 2022) and human service robotics (Brohan et al., 2022) have required reasoning methods capable of operating under this uncertainty, unlike those in the highly structured environments of traditional industrial robotics. Both of these examples require an agent to operate in environments designed for humans, meaning that they have some level of structure and predictability, and large-scale datasets are available at moderate cost. However, deploying robots in even more unstructured natural environments can be more challenging still (Zereik et al., 2018; Tranzatto et al., 2022).

This thesis focuses on sequential decision-making in environments with high levels of *epistemic uncertainty*: that is to say, the agent's knowledge of the environment is incomplete and uncertain. Rather than *local* uncertain effects, such as noisy sensors and uncertain dynamics model coefficients (Slade et al., 2017; Ramos et al., 2019), we consider *global* uncertainty about the true state or dynamics of the whole deployment environment. Example settings with high environmental uncertainty include underwater and surface vehicles operating in the ocean (Ho et al., 2020), robots deployed in hazardous environments with unknown hazard distributions (Tsitsimpelis et al., 2019), and unmanned aerial vehicle navigation with unpredictable spatial distributions of wind or communication signal quality (Ladosz et al., 2019). To achieve their goals, our agents must make long-term plans that are resilient or adaptive with respect to this uncertainty in the environment.

In particular, we emphasise the *situated* nature of the agent in the environment (Kinny and George, 1991; Wooldridge, 1997; Parashar et al., 2018). The agent's actions are impacted by the environment, it generally has only local information about the environment based on its sensors, and it likely cannot achieve globally optimal behaviour. What it can achieve is *bounded optimality* (Russell and Subramanian, 1994): the best possible decisions given its limited computational resources and the uncertainty in the environment. In the situated setting, the agent's embodiment, resource constraints and its reasoning process itself should be first-class entities in the decision-making problem.

For the first three methods described in this thesis, the situated nature of the agent is taken into account by decision-making methods that are appropriate for the problem setting. For example, some settings call for more adaptation to the true environment dynamics (Chapter 5), and some require only general resilience to possible environment instantiations (Chapter 3). All methods in this thesis are based around the Markov decision process (MDP) framework (Puterman, 1994), which is a standard tool for decision-making under uncertainty.

However, in many cases, it is difficult to determine in advance a decision-making strategy that is optimal or even sufficiently well-performing. The optimal strategy may depend on the agent's current and future states, the environment, or some cumulative budget such as remaining battery life. This is particularly important for mobile robots: they must share their battery capacity between sensing, computing and actuation demands. For each of the three aforementioned decision-making methods, we identify cases where an inability to consider its own reasoning process causes the agent to make suboptimal decisions. Therefore, the final two algorithms presented in this thesis are automated *metareasoning* (Russell and Wefald, 1991) methods which learn how best to use an agent's reasoning capabilities given the situation it finds itself in.

1.1 Concepts and Context

Planning with models vs. reinforcement learning Planning is the task of determining a policy – a mapping from states to actions – with a given model. Value-based planning algorithms such as Value Iteration (VI) (Bellman, 1966) solve for a policy by estimating the expected future cumulative reward from each state, and selecting the action that maximises this value. Search algorithms such as Monte Carlo Tree Search also estimate the value of states, but do so by sampling trajectories using the model rather than calculating values for all states. By contrast, in the reinforcement learning (RL) setting, the only information available is either an interface to interact with the environment (online RL) or a fixed dataset of interactions (offline RL). One common high-level approach to RL is value-based model-free deep RL, which uses deep neural networks to approximate the value function based on the agent's experience. By contrast, model-based RL methods use the agent's experience to learn a model of the environment that can then be used for planning or for sampling to train a model-free RL agent (Sutton et al., 1998).

Chapters 3 and 4 use variants of VI, and the algorithm in Chapter 5 is based on MCTS. The model in Chapter 3 is learned using simulation *before* the mission, whereas in Chapters 4 and 5 the model is learned *online* from execution data. Chapters 4 and 5 are therefore model-based RL methods, as the agent learns a model of the environment dynamics from data and uses a planning algorithm to make decisions based on the model. Chapters 6 and 7 use model-free deep RL to solve the metareasoning task, due to its complexity and the difficulty of constructing a good model of reasoning processes.

Epistemic vs. aleatoric uncertainty Conceptually, uncertainty can be categorised as either epistemic or aleatoric (Sullivan, 2015). Aleatoric uncertainty is noise or stochasticity considered to be inherent to the environment, and cannot be reduced by collecting more data. Epistemic uncertainty arises from the agent's incomplete knowledge of the environment, such as unknown model parameters or environment dynamics that cannot be captured by the model's structure. Of

course, the distinction between the two types of uncertainty is often a modelling choice, made based on which sources of uncertainty may be practically reducible, rather than a fundamental property of the environment. In principle, epistemic uncertainty can be decreased by making the agent's models more accurate, which would typically require collecting more data.

Frequently, both types of uncertainty are present: consider a simple example in a 1-dimensional continuous state space. The ground-truth transition outcome of an action is given by a Gaussian distribution. The agent does not know the mean and variance of the distribution, so its model of the environment contains epistemic uncertainty. Even if the agent's model of the environment is perfect, the action outcome is still stochastic, so there is aleatoric uncertainty in the action outcome.

The types of environments we consider in this thesis are generally much larger than the agent, and difficult to model even without the inherent limitations of a single mobile robot's sensors and computational resources. Although the agent can gain some knowledge throughout its mission, it will never have a fully accurate model of the environment. Therefore, our decision-making methods must be able to reason about the uncertainty in the agent's models. In this thesis we take a Bayesian approach to epistemic uncertainty (Soize, 2017), where probability distributions, or *belief distributions*, are used and updated to represent the agent's changing beliefs about the environment. All problem settings in this thesis include aleatoric uncertainty, which is captured in the standard MDP model by the probabilistic transition function.

We refer to methods that aim to optimise expected performance across the belief distribution as *resilient* methods. This contrasts with methods which are *robust* to epistemic uncertainty, which aim to optimise the worst-case performance (Rigter et al., 2021a; Badings et al., 2023). Due to the minimax objective of robust methods, they typically employ uncertainty sets, rather than belief distributions, to characterise epistemic uncertainty. Finally, methods that are *adaptive* to epistemic uncertainty are those that change their behaviour based on data collected during deployment. They are therefore able to improve their performance in the updated

belief distribution given the new data. Several existing RL methods achieve adaptive behaviour in the online (Ross et al., 2007; Guez et al., 2013) and offline (Pong et al., 2022; Ghosh et al., 2022) RL settings.

Algorithm modes of operation One classification scheme for decision-making methods – based on *when* decision-making computation is performed – is what we refer to as the *mode of operation*. *Offline* methods compute a policy before the agent is deployed, and the policy is then executed without further computation other than that required to determine the current state and look up the corresponding action in the policy. *Online* methods carry out computation to synthesise or improve the current policy *during* the agent's deployment. *Replanning* methods can be considered a hybrid of the two, where the agent alternates between executing a policy (for one or multiple timesteps) and recomputing the whole policy.

In this taxonomy, online methods use a single policy which is frequently updated to incorporate new information or improve expected performance based on the currently available information. The amount of policy improvement computation carried out per action selection timestep need not be fixed, although this is a common approach.

In contrast, replanning methods recompute new policies from scratch, and use those policies until the next recomputation. For example, we would label model-predictive control (MPC) (Morari and Lee, 1999) as a replanning method, as it typically recomputes a new fixed-horizon plan at each action-selection timestep. More generally, policy recomputation may occur based on a schedule, new data, or unexpected events in the environment (Yoon et al., 2007; Cashmore et al., 2019). If new data has been collected, the agent's models may be updated before replanning in order to incorporate the new information.

Metareasoning. Metareasoning is the study of reasoning about the reasoning process, and has an extensive history in artificial intelligence and cognitive science (Horvitz, 1988; Russell and Wefald, 1991; Griffiths et al., 2019). In the context of automated decision-making, we are generally interested in resource-bounded

(a) Examples of timelines for offline metareasoning. The lower timeline shows a shorter planning period followed by a longer execution period. If the objective is to minimise the *total* time taken, the lower timeline is preferable despite the execution stage taking longer.

(b) Examples of timelines for online metareasoning. The lower timeline contains three short planning periods, compared to two longer periods in the upper timeline. The makespan of the lower timeline is shorter, so this planning approach is preferable.

Figure 1.1: Illustrative example timelines of offline and online metareasoning methods controlling the planning time of an object-level algorithm. For these examples, it is assumed that the metareasoning objective is to minimise the total planning and execution time to complete a task.

metareasoning: optimising for an objective that includes the agent's reasoning process as a component. For example, a metareasoning agent might decide how long to spend planning before executing a plan, which planning algorithm to use, or how frequently to update its models based on new data already collected.

Specifically, the methods in Chapters 6 and 7 take a *metalevel control* approach (Cox and Raja, 2011). This setting posits a separate *metalevel* agent that controls the reasoning process of an *object-level* agent or algorithm. The metalevel agent may observe the object-level agent's state, and can take actions to control the object-level agent's reasoning process. One key challenge in the metalevel control setting is scalable prediction of reasoning dynamics. Benefit is only gained from metareasoning if the metalevel agent has a low-overhead method for predicting the outcomes of different reasoning actions, otherwise more computation is spent on metareasoning than is saved by the improved reasoning process (Lin et al., 2015).

In this thesis we distinguish between *offline* vs. *online* metareasoning methods. The distinction refers to the type of reasoning that is being controlled, rather than the mode of operation of the metalevel agent itself: *offline metareasoning* is metareasoning to control offline reasoning, and *online metareasoning* is metareasoning to control online reasoning. The offline metareasoning setting assumes a single period of computation followed by a single period of execution, as illustrated in Figure 1.1a. In the online setting, the metalevel agent controls the reasoning process

throughout execution. This is illustrated in Figure 1.1b. Online metareasoning methods are therefore necessary to control object-level algorithms with online or replanning modes of operation.

Separately, the mode of operation of the metalevel agent itself may be online or offline. In this thesis we design metalevel agents which are trained offline, in order to avoid metareasoning overhead during deployment. Some metareasoning methods carry out significant online computation during the reasoning process (Lin et al., 2015; Svegliato et al., 2018), but generally ignore the computational overhead.

1.2 Thesis Structure

Chapter 2 covers preliminaries and a unified discussion of related work relevant to the following chapters. Chapters 3-7 are *paper chapters*, each consisting mainly of a paper that has been published or is currently under review. Paper chapters begin with a review of the setting and assumptions of the work, and description of the work's contributions. They conclude with further analysis, including the limitations of the work and potential future research avenues. Each paper is self-contained with its own reference list, while sources outside of the paper content are collected at the end of the thesis. Apart from Chapter 4, where the appendices are integrated into the main text, paper appendices are collected at the end of the thesis. Citations for the published versions of papers included in each chapter are provided at the end of this section in Table 1.2, and before each paper. Page numbering for papers has been overridden for continuous numbering across the thesis. Other than this, the formatting of the paper chapters is unchanged from the published versions.

Table 1.1 gives an overview of the paper chapters in this thesis, comparing the problem settings, assumptions and key method details from the following:

- **Epistemic uncertainty**: different problem settings focus on different sources of epistemic uncertainty. As the thesis progresses, the focus shifts from epistemic uncertainty in the environment dynamics to epistemic uncertainty in the reasoning process of an agent situated in an environment.

	Chapter 3	Chapter 4	Chapter 5	Chapter 6	Chapter 7
Epistemic uncertainty	Environment dynamics	Environment dynamics	Environment dynamics	Planner dynamics	Planner dynamics
Mode	Offline	Replanning	Online	Offline	Online
Model	Timed MDP	EstMDP	BAMDP	Metalevel MDP	Metalevel MDP
Solution method	VI (max reward)	VI (reachability)	MCTS	Deep RL	Deep RL
Data regime	Small/ medium	Small	Small	Large	Large

Table 1.1: Settings, methods and features for each chapter. This table is repeated at the start of each chapter.

- **Mode**: the offline/online mode of operation of the algorithm, generally a design requirement for the algorithm given the problem setting. For the two metareasoning methods, their mode of operation must match the mode of operation of the reasoning algorithm they are controlling.
- **Model** and **Solution method** refer to the core formulation and corresponding solution method used in each chapter. For the first three chapters, this is the decision-making formulation and solution method used by the agent. For the two metareasoning methods, this is the formulation and solution method for the metalevel agent.
- **Data regime**: different methods have different data requirements. As the first three chapters are designed for decision-making in partially unknown environments, they are designed to operate with little or no prior data about the deployment environment. A small data regime consists of tens of data points, while a large data regime involves thousands or even millions.

1.3 Contributions

In the situated setting, an agent's embodiment, resource constraints and its reasoning process itself should be first-class entities in the decision-making problem. Therefore, in this thesis, we aim to address the following core research question:

Core research question

How can a situated agent make the best possible decisions given its limited computational resources and knowledge of the environment?

We answer this question by breaking it down into a series of sub-questions. First, we separate the setting by whether the agent has the on-board computational resources to carry out decision-making during execution. For agents which cannot, we take an offline approach to decision-making. For agents with sufficient compute ability, we take a replanning or fully online approach. For each case, we develop both decision-making and metareasoning methods. Decision-making methods are designed to be resilient or adaptive to epistemic uncertainty in the deployment environment. The metareasoning methods are designed to optimise the decision-making process itself, and are therefore adaptive to epistemic uncertainty about the reasoning process.

RQ1: Offline decision-making. How can we pre-plan decision-making for an agent to be deployed in an epistemically uncertain environment?

RQ2: Offline metareasoning. How can a reasoning system optimise the process of pre-planning and deploying a plan?

RQ3: Online decision-making. How can an agent with on-board computational resources adapt its behaviour to the true environment dynamics?

RQ4: Online metareasoning. How can an agent with on-board computational resources optimise its own reasoning process during deployment?

In Figure 1.2, we illustrate how the overall research question is broken down into sub-questions, and show where each thesis chapter fits into this breakdown.

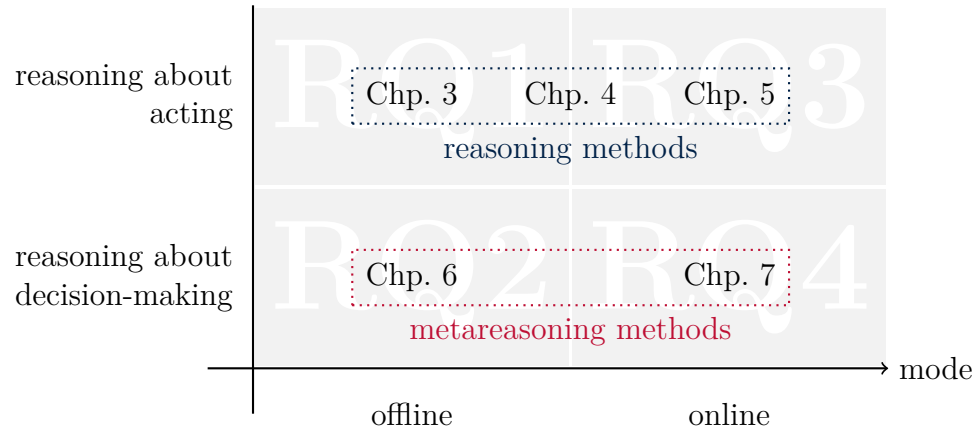


Figure 1.2: Research sub-questions RQ1 to RQ4, separated by mode of operation and what is being reasoned about. Paper chapter methods are placed according to which question they address.

The first three paper chapters in this thesis propose new methods for decision-making under uncertainty for agents situated in epistemically uncertain environments. As chapters progress, the decision-making is increasingly adaptive to the true environment dynamics, and require increasing online reasoning capabilities. Methods are evaluated in real-world deployments (Chapters 3 and 4) or in high-fidelity simulation (Chapter 5).

Chapter 3 addresses **RQ1** by developing and analysing a new method, based on parameterised simulators, for decision-making with limited onboard computational resources. We apply this concept to develop a first-of-its-kind probabilistic decision-making method for underwater data retrieval with autonomous underwater vehicles. The offline decision-making approach enables deployment with a low-cost underwater vehicle with only limited computational capability. Chapter 6 addresses **RQ2** by developing a metareasoning method for offline decision-making. The metareasoning method is applicable to any offline probabilistic planning method, whereas previous methods only applied to deterministic planning or made strong simplifying assumptions that degrade performance.

To address **RQ3**, we progress to decision-making using Bayesian belief models that incorporate environment measurements made by the agent during execution.

Chapter 4 details a replanning-based method which limits online compute requirements by replanning only when the agent's belief about the environment changes in a way that impacts the agent's current plans. The task we consider is safe exploration of unknown environments with unknown hazards. Our method expands the class of environments which can be safely explored compared to previous methods, and outperforms existing methods in exploration efficiency. As a replanning method, Chapter 4 shares some properties with the offline setting and some with the online setting, so is placed between categories in Figure 1.2.

Chapter 5 takes a fully online approach, making use of more online computation to adapt to the true environment dynamics with a theoretically optimal exploration/exploitation trade-off. Our method outperforms previous algorithms for belief-based decision-making in epistemically uncertain environments.

Chapter 7 addresses **RQ4** by extending the framework in Chapter 6 to the online setting. To illustrate the benefits of carrying out online metareasoning, one experiment domain contains a simple didactic example of epistemic uncertainty in the environment dynamics. Quantitative and qualitative results demonstrate our method outperforming fixed reasoning time strategies, the offline metareasoning method from Chapter 6, and the previous state-of-the-art metareasoning method for online probabilistic planning.

Chapter	Paper reference
Chapter 3	**Matthew Budd**, Georgios Salavasidis, Izzat Kamarudzaman, Benjamin Sherlock, Jeffrey Neasham, Bruno Lacerda, Nick Hawes, Alexander B. Phillips, and Catherine Harris. AUV-based data harvesting from underwater sensor networks using probabilistic planning. *Manuscript under review, Journal of Field Robotics*, 2025. The following conference publication covers the same method, but with much less detail. Therefore it is not included in this thesis: **Matthew Budd**, Georgios Salavasidis, Izzat Karnarudzaman, Catherine A Harris, Alexander B Phillips, Paul Duckworth, Nick Hawes, and Bruno Lacerda. Probabilistic planning for AUV data harvesting from smart underwater sensor networks. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.
Chapter 4	Alex Stephens[†], **Matthew Budd**[†], Michal Staniaszek, Benoit Casseau, Paul Duckworth, Maurice Fallon, Nick Hawes, and Bruno Lacerda. Planning under uncertainty for safe robot exploration using Gaussian process prediction. *Autonomous Robots*, 48(7):18, 2024. [†]joint first authorship
Chapter 5	**Matthew Budd**, Paul Duckworth, Nick Hawes, and Bruno Lacerda. Bayesian reinforcement learning for single-episode missions in partially unknown environments. In *Conference on Robot Learning*. PMLR, 2023. URL `https://proceedings.mlr.press/v205/budd23a/budd23a.pdf`.
Chapter 6	**Matthew Budd**, Bruno Lacerda, and Nick Hawes. Stop! Planner Time: metareasoning for probabilistic planning using learned performance profiles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20053–20060, 2024.
Chapter 7	**Matthew Budd**, Bruno Lacerda, and Nick Hawes. Think Fast! Learning to Control Online Reasoning in Stochastic Environments. *Manuscript under review*, 2025.

Table 1.2: Citations for the published versions of papers included in each chapter, or provisional citations for papers currently under review.

2 Background

2.1 Decision-Making under Uncertainty

In this thesis we take a probabilistic approach to sequential decision-making, and make use of Markov decision process (MDP) models to represent environments and agent reasoning processes. These are stochastic models which satisfy the Markov property: evolution from the current state to the next state depends only on the current state.

MDPs are discrete-time processes. We also define and use continuous-time Markov decision processes (CTMDPs) in Chapter 3. As these continuous-time models only appear in Chapter 3, we do not define or discuss them here.

2.1.1 Foundational Models

Markov Decision Processes

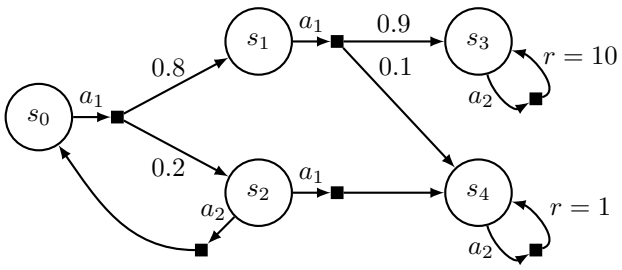


Figure 2.1: A simple example of a fully defined finite MDP with 5 states and 2 actions.

A Markov decision process (MDP) is a tuple of the form $\mathcal{M} = \langle S, \overline{s}, A, T, R \rangle$, where:

- S is a finite set of states,

- $\overline{s} \in S$ is the initial state. In some cases we denote this as s_0, or define instead an initial state distribution $init : S \to [0, 1]$,
- A is a finite set of actions,
- $T : S \times A \times S \to [0, 1]$ is the probabilistic transition function, where $T(s, a, s')$ is the probability of transitioning from state s to state s' after taking action a. In some cases we denote this as $\delta(s, a, s')$ instead of T, and
- $R : S \times A \to \mathbb{R}_{\geq 0}$ is a reward function, where $R(s, a)$ is the reward received after taking action a in state s. Alternatively and equivalently, we can define a cost function $C : S \times A \to \mathbb{R}_{\geq 0}$, where $C(s, a)$ is the cost incurred after taking action a in state s.

In some cases, we also define goal states $G \subseteq S$. This makes the MDP a stochastic shortest path (SSP) MDP, where the goal is to reach a goal state while minimising the expected cost to do so. An example of a finite MDP with rewards is shown in Figure 2.1. In this thesis we focus on discrete, finite MDPs, but it is possible to define MDPs with continuous state and action spaces. Where MDPs are defined with a cost function, the objective is usually to minimise the expected cost incurred by the agent. If defined with a reward function, the objective is usually to maximise the expected reward received by the agent.

MDPs can be separated into three classes: finite-horizon MDPs, infinite-horizon discounted-reward MDPs, and indefinite-horizon MDPs (Mausam and Kolobov, 2012). In Chapter 3 the MDP is defined by construction without loops, meaning that the state space forms a directed acyclic graph (DAG). This is effectively a finite-horizon MDP with the horizon implicitly defined by the structure of the MDP. Other chapters generally use an SSP MDP formulation, which is a type of indefinite-horizon MDP that terminates upon reaching a goal state. Infinite-horizon discounted-reward MDPs also define a discount factor $\gamma \in [0, 1)$ to ensure convergence of the expected reward, but we do not study infinite-horizon discounted-reward MDPs in this thesis.

A *policy* π defines which action(s) to take in each state. We are generally interested in *deterministic*, *stationary* policies, which define a single action to take

in each state: $\pi : S \to A$. A *complete* policy defines an action for every state in the MDP. A *partial* policy defines an action for a subset of states in the MDP. For SSP MDPs, a policy is *proper* in state s if following the policy from state s leads to reaching a goal state with probability 1. A policy is *proper* if it is proper in every state $s \in S$. As a defining condition for an SSP MDP, there must exist a proper policy from the initial state (Mausam and Kolobov, 2012). All *improper* policies incur infinite expected cost.

A history $h_t \in \mathcal{H}^{\mathcal{M}}$ is a sequence of states and actions up to time t, where $h = (s_0, a_0, s_1, a_1, \ldots, s_t)$. $\mathcal{H}^{\mathcal{M}}$ is the set of all histories in the MDP $\mathcal{M}$. *History-dependent* policies map histories to actions: $\pi : \mathcal{H}^{\mathcal{M}} \to A$. History-dependent policies can enable more complex adaptive behaviour in partially observable or uncertain models (Ghosh et al., 2021). They have a much larger input space than stationary policies, so are harder to optimise or learn. In problems with an indefinite or infinite horizon, the history length is unbounded, so the input space is infinite.

We now define value functions and optimal policies for MDPs. We focus on the SSP MDP formulation because it is commonly used in this thesis, and because SSP MDPs are strictly more general than the other MDP classes (Mausam and Kolobov, 2012).

The *value function* $V^\pi(s)$ of a policy π in state s is the expected cost incurred by following the policy from state s:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} C(S_t, A_t) \mid S_0 = s \right]. \tag{2.1}$$

Q-values can also be defined to represent the value of taking action a in state s and following π thereafter:

$$Q^\pi(s, a) = C(s, a) + \sum_{s'} T(s, a, s') V^\pi(s'). \tag{2.2}$$

An optimal policy π^* is a policy that minimises the expected cost from the initial state s_0, and the optimal value function $V^*(s)$ is the value function of

the optimal policy or policies:

$$\pi^* = \arg\min_{\pi} V^{\pi}(s_0), \tag{2.3}$$

$$V^*(s) = \min_{\pi} V^{\pi}(s). \tag{2.4}$$

It can be shown that in SSP MDPs, there exist optimal policies that are both deterministic and stationary (Mausam and Kolobov, 2012). The value of a policy π in state s can be estimated using Monte Carlo (MC) methods (Sutton et al., 1998):

$$V^{\pi}(s) \approx \frac{1}{N} \sum_{i=1}^{N} \left[\sum_{t=0}^{\infty} C(S_t, A_t) \right], \tag{2.5}$$

where the estimate converges to the true value function as $N \to \infty$ with standard deviation $\propto \frac{1}{\sqrt{N}}$. In practice, evaluating an improper policy may cause Monte Carlo evaluations to fail to terminate, preventing the estimate from converging.

Partially Observable MDPs and Belief MDPs

The Partially Observable MDP (POMDP) is a generalisation of the MDP, where the state is not fully observable (Kaelbling et al., 1998). POMDPs are not utilised in this thesis, but they are closely related to the uncertain parameter models we study. A POMDP definition includes a set of possible observations and an observation function that defines which observations may be stochastically emitted, based on the action taken and the outcome state. At a high level, decision-making in POMDPs requires maintaining a belief distribution (or simply "belief"), which is a probability distribution over the possible current states. Decision-making in POMDPs can be carried out by explicitly or implicitly constructing a *belief MDP*, which is an MDP where the states are the possible belief states (Kaelbling et al., 1998). The belief MDP is a continuous state space MDP, where transitions between belief states are a combination of the transition function of the POMDP and a *belief update* to the new belief state at the new node, given the observation received. Acting optimally in the belief MDP optimally trades off information gathering and reward gathering, so causes the agent to act optimally with respect to its beliefs in the original POMDP. An equivalent MDP to the belief MDP can be constructed by

augmenting the state space with histories of observations and actions, because the belief state is a sufficient statistic for the history (Kaelbling et al., 1998). Due to the continuous state space, belief-space planning suffers from the "curses of dimensionality and history" (Pineau et al., 2006), making pre-planning for partially observable problems of practical size very computationally expensive. We discuss a specific type of belief MDP for epistemically uncertain models in Section 2.1.3.

2.1.2 Algorithms for Solving MDPs

Value Iteration

Value iteration (VI) is a standard dynamic programming algorithm for solving MDPs (Bellman, 1966). VI stores the value function for each state in a table, and iteratively updates the value function for each state using the Bellman equation. For cost minimisation problems, the Bellman update equation is:

$$V_{k+1}(s) = \min_a \left[C(s,a) + \sum_{s'} T(s,a,s') V_k(s') \right]. \tag{2.6}$$

The value function converges to the optimal value function V^* as $k \to \infty$. In practice, VI is run for a fixed number of iterations, or until the value function converges to within a specified tolerance. VI is a full state space method, so produces a policy for every state in the MDP.

Some states in the MDP are unreachable from the initial state, or unreachable from the initial state under any optimal policy. This means that VI may waste computational resources on unimportant states. Topological VI methods (Dai et al., 2011) identify structure in the MDP to reduce the number of backups required. In Chapter 3 we use a topological VI method to efficiently solve MDPs where a timestep value is included in the state. As every transition in the MDP increments the timestep by at least one, this leads to a natural topological ordering of the state space. Also in Chapter 3, we mention point-based value iteration (PBVI) (Pineau et al., 2003), which is a value iteration method for POMDPs based on sampling representative belief points from the continuous belief state space.

Trial-based Heuristic Tree Search

Trial-based heuristic tree search (THTS) (Keller and Helmert, 2013) is a framework that subsumes several families of algorithms for solving MDPs. What the algorithms have in common is that they carry out *trials*, simulated sequences of actions and states up to a specified depth or termination condition, in order to estimate a value function or create a policy. THTS categorises algorithms by several properties, including which type of backups are used, and how actions are selected during trials.

One family of THTS algorithms is real-time dynamic programming (RTDP) algorithms (Barto et al., 1995). As the name suggests, these algorithms carry out full-width dynamic programming backups, but only for states visited during the trial. One specific algorithm is Bounded RTDP (BRTDP) (McMahan et al., 2005), which maintains two value functions for each state: a lower bound and an upper bound. The lower bound is used to select actions during trials, and the inter-bound difference is used to preferentially sample outcomes with high value uncertainty. In Chapter 6 we build a variant of BRTDP with multiple lower-bound value functions.

Another family of THTS algorithms is Monte Carlo Tree Search, which is used extensively in many decision-making problems (Świechowski et al., 2023), sometimes in combination with learning-based methods (Schrittwieser et al., 2020). MCTS algorithms iteratively construct a search tree that estimates the Q-value function for states represented in the tree. During each trial, up to the point that a new state is added to the tree, the algorithm selects actions according to a *tree* policy. The most common tree policy is Upper Confidence Bound for Trees (UCT) (Kocsis and Szepesvári, 2006), which selects actions according to an estimate of the upper or lower bound of the value function, based on the number of visits to the state $N(s)$ and the number of times $N(s, a)$ that action a has been tried in state s:

$$a_{tree} = \arg\max_{a \in A} \left[Q(s, a) - c\sqrt{\frac{\ln N(s)}{N(s, a)}} \right], \tag{2.7}$$

where c is a constant that controls the exploration-exploitation trade-off. This tree policy is typically shown with an upper bound rather than a lower bound as shown here, because the reward maximisation setting is more common than the

cost minimisation setting. We make use of MCTS-based methods with UCT exploration in Chapter 5.

After a new state is added to the tree, the algorithm uses a *rollout* policy to estimate its value. Learning variants commonly use a value function approximator instead of a rollout policy (Schrittwieser et al., 2020). Q-values and node statistics are updated after the trial is complete, using Monte Carlo (MC) backups. Given that the algorithm uses MC backups, it does not need access to a closed-form model of the MDP: sample-based black box access is sufficient.

MCTS variants applicable to continuous state and action spaces are also available. Progressive widening (Couëtoux et al., 2011) provides a method for controlling branching factors, specifying *when* to sample new actions and/or outcomes. Other methods focus on *which* actions to sample, such as Voronoi partitioning (Kim et al., 2020). In large state and action spaces, deep RL is another common approach to decision-making (Mnih et al., 2015).

Many MCTS-based algorithms exist for solving POMDPs, usually in an online setting. A well-known approach is POMCP (Silver and Veness, 2010), which builds a search tree consisting of histories of actions and observations. An unweighted particle filter over histories is used as the belief state representation. POMCP introduces a concept known as *root sampling*, where a possible state is sampled from the MCTS tree root belief node and used as if it were the true underlying state throughout one MCTS trial. This can be shown to result in the same rollout distribution as if full belief distributions were maintained at every tree node, and new nodes were created using full belief updates at every step. The method improves computational efficiency, as explicit belief updates can be computationally expensive.

Recent algorithms further increase the search performance of POMCP, such as by regularising policies (Ye et al., 2017). POMCPOW (Sunberg and Kochenderfer, 2018) extends POMCP to continuous state, action and observation spaces by employing a weighted particle filter and progressive widening. LCEOPT (Hoerger et al., 2024) is another algorithm which applies to continuous state and action space POMDPs.

THTS algorithms are commonly used online, where search is run for a fixed time limit or number of trials before taking an action. If possible, the root node of the search tree is updated with the result of the action taken, allowing some of the search tree to be carried forward to the next time step.

2.1.3 Uncertain Parameter Models

MDPs with Uncertain Parameters

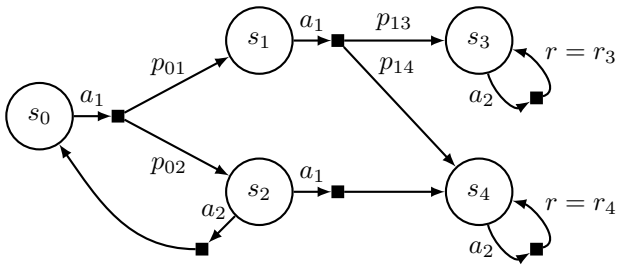


Figure 2.2: A simple example of an uncertain, i.e. underdefined, finite MDP. Some structure is known (e.g. which states are connected), but the transition probabilities and rewards are not known for every state and action.

In this thesis we study MDPs with uncertain parameters. An example of an uncertain MDP is shown in Figure 2.2. We focus on the case where the transition function is unknown, and treat reward functions as known *a priori* because they are frequently used to specify the task or intended behaviour.

The MDP in Figure 2.2 has two unknown parameters in the transition function, p_{01} and p_{13}. The other two parameters are simply $1 - p_{01}$ and $1 - p_{13}$, as transition probabilities must sum to 1. Different approaches to decision-making with uncertain models make different assumptions about the nature of the uncertainty, and aim to optimise different metrics. One key distinction is whether parameters are correlated or independent. If parameters are assumed to be correlated, experience can be *generalised* rather than having to independently retry all actions in all states to learn the transition function.

Reinforcement learning (RL) algorithms learn policies for MDPs with unknown parameters by interacting with the environment. Most commonly, they assume interactive access to the environment to collect large amounts of transition outcome

data, or a large dataset of transition outcomes (the offline RL setting). Most commonly, RL algorithms optimise for expected reward, but some algorithms optimise for other metrics such as risk measures (Chow and Ghavamzadeh, 2014).

Contrastingly, robust MDP methods are planning methods that do not learn from experience. They assume that transition probabilities belong to some uncertainty set, and aim to find a policy that performs well under the worst-case transition probabilities in the set (Suilen et al., 2024). Interval MDPs (Givan et al., 2000) assume that transition probabilities are bounded by known upper and lower bounds, and assume independence between different transitions. Sample-based uncertain MDPs (Adulyasak et al., 2015) posit a finite set of possible transition functions, thereby allowing generalisation by tracking which samples are consistent with the observed transition outcomes so far.

Bayesian Model-Based RL

Bayesian model-based reinforcement learning (Bayesian MBRL) (Ghavamzadeh et al., 2015) is a family of problem settings and algorithms where Bayesian methods are used to learn the transition function of an MDP. In the Bayesian MBRL setting, the agent maintains a belief distribution over the transition function parameters, and updates the belief distribution based on observed transition outcomes.

In Bayesian MBRL, *Bayes-optimal* policies represent the theoretical optimal behaviour of the agent, achieving optimal exploration/exploitation performance with respect to the prior belief over the transition function. Many algorithms, however, employ approximations that trade off optimality for computational efficiency. The performance of a Bayes-optimal policy is upper bounded by the performance of the optimal policy in the MDP with the true transition function (Ghavamzadeh et al., 2015), which requires full knowledge of the transition function to calculate.

Depending on the form of the belief, there may or may not be correlation between transition parameters. For discrete state and action spaces, one option is to assume fully independent transitions where separate multinomial distributions give the output probabilities for each transition. In this case, the belief distribution

for each transition is a Dirichlet distribution, which is a conjugate prior for the multinomial distribution. The Dirichlet belief distribution is updated using the counts of observed transition outcomes, which are multinomial samples (Duff, 2002). In Chapters 4 and 5 we use belief models based on Gaussian processes (Section 2.2), which results in correlation between transition probabilities at different states, enabling generalisation of experience.

The Bayes-Adaptive MDP

The Bayes-adaptive MDP is a special case of a belief MDP, where the belief is over the transition function parameters (Duff, 2002). Rather than POMDP observations, the agent uses the outcomes of actions taken in the MDP to update the belief distribution. Acting optimally in this MDP achieves Bayes-optimal behaviour, in the same way that acting optimally in a POMDP belief MDP leads to optimal exploration/exploitation behaviour in that POMDP.

We define BAMDPs in terms of histories rather than belief states, again because a belief is a sufficient statistic for the history. Given an MDP $\mathcal{M} = \langle S, \overline{s}, A, T, C \rangle$ with an unknown transition function T, and a prior $p(T)$ over transition functions, the corresponding BAMDP is defined as the tuple $\mathcal{M}^+ = \langle S^+, \overline{s}^+, A, T^+, C^+ \rangle$, where:

- $S^+ = S \times \mathcal{H}^{\mathcal{M}}$, i.e. the state space is augmented with histories of actions and states,
- $\overline{s}^+ = (\overline{s}, h_0)$, where h_0 is the empty history,
- A is the same action space as in the MDP,
- $T^+ : S^+ \times A \times S^+ \to [0, 1]$ is the transition function, where $T^+((s, h), a, (s', h'))$ is the probability of transitioning from state s with history h to state s' with history $h' = (h, a, s')$ after taking action a, and
- $C^+ : S^+ \times A \to \mathbb{R}_{\geq 0}$ is the cost function, where $C^+((s, h), a) = C(s, a)$.

T^+ can be calculated by marginalising over possible transition functions T according to the posterior belief distribution $p(T \mid h)$, which is the belief distribution over transition functions given the history h:

$$T^+((s,h),a,(s',h')) = \begin{cases} \int_T T(s,a,s')p(T \mid h)dT, & \text{if } h' = (h,a,s'), \\ 0, & \text{otherwise.} \end{cases} \tag{2.8}$$

A policy in the BAMDP is a mapping from histories to actions, therefore Bayes-optimal policies are history-dependent policies.

In some sense, a BAMDP is easier to solve than a POMDP. In the equivalent POMDP formulation, the environment state is fully observable, and the hidden component of the state is the true transition function. This hidden component is fixed during an episode, as the transition function does not change. In the case where there are a finite number of possible transition functions, it is possible to define an "information horizon" for the BAMDP (Arumugam and Singh, 2022). If this horizon exists for the given BAMDP and policy class, it represents the latest timestep by which the agent is guaranteed to have fully resolved all uncertainty about the environment and to have exactly identified the transition function. However, in practice BAMDPs are still highly complex to solve due to the continuous belief state space, and practical algorithms are generally based on online MCTS.

Bayes-adaptive POMDPs (BA-POMDPs) formulate the problem of learning a POMDP model from experience as a BAMDP, where belief over possible observation functions must also be maintained (Ross et al., 2007).

2.1.4 Algorithms for Solving Uncertain Models

BAMCP

BAMCP is a modification of POMCP (Silver and Veness, 2010) for the BAMDP setting. Where POMCP root samples the true unobservable state from the belief state, BAMCP root samples (the parameters of) the transition function from the belief state over transition functions. This transition function is then used for that MCTS trial. The transition functions used when passing through search tree nodes are stored in that node, forming a particle filter over transition functions that could

have caused the observed history. This particle filter can be used as the belief over transition functions when the root node is updated after taking an action and observing the outcome. Alternatively, an exact belief update may be used if the belief representation is sufficiently tractable to update after real actions, even if it is not tractable enough to update for every new node added to the MCTS tree during planning. Carrying out exact belief updates after real actions avoids potential issues with particle filtering, such as particle depletion. We base our algorithm in Chapter 5 on BAMCP, and use exact Gaussian process belief updates after receiving real measurements from the environments, which are equivalent to action outcomes in our formulation of Bayesian MBRL.

As an MCTS method based on histories, BAMCP (and POMCP) cannot generalise between histories that correspond to the same or similar belief states. A history where an agent tries action a_1 before action a_2 is represented in a different branch of the MCTS tree to a history where the agent tries action a_2 before action a_1, even if the same outcomes are observed. A successor to BAMCP, Bayes-Adaptive planning with Function Approximation (BAFA) (Guez et al., 2014a), uses value function approximation to generalise between histories and better scale to larger or continuous state spaces. However, it removes the MCTS structure and so must use simpler ϵ-greedy exploration strategies rather than UCT exploration.

Katt et al. (2017) extend BAMCP to apply to BAPOMDPs, carrying out root sampling of the observation function as well as the transition function.

RL algorithms based on posterior sampling have had empirical success and shown good regret performance (Osband et al., 2013; Osband and Van Roy, 2017) in the episodic setting. However, for single-episode settings where an optimal exploration/exploitation trade-off is required, true Bayesian decision-making methods are more effective (Guez et al., 2014b).

Methods Based on Deep RL

Meta-RL methods aim to enable agents to adapt to new tasks or environments quickly (Nagabandi et al., 2018). Some meta-RL methods aim to learn approximately

Bayes-optimal policies using deep RL, without incurring the computational cost of online inference with the belief model. One well-known method is VariBAD (Zintgraf et al., 2020), which defines a stochastic latent variable that represents which MDP transition/reward function the agent is operating under. A variational autoencoder (Kingma et al., 2013) is used to infer the distribution over the latent variable given the observed history. The learned policy is conditioned on this posterior over the latent variable. Some more recent model-based offline RL methods use VariBAD-inspired latent variable models (Dorfman et al., 2020; Chen et al., 2021). This enables adaptation at deployment time after training with only the offline dataset, leading to better performance than offline RL methods that avoid uncertainty rather than adapting to it (Yu et al., 2020).

As deep RL methods, meta-RL methods require access to a training distribution covering the space of possible transition and reward functions. In the offline case, this is a dataset of interactions from the training distribution.

2.2 Gaussian Processes

Gaussian processes (GPs) are a non-parametric Bayesian modelling approach, commonly used for regression and classification (Rasmussen and Williams, 2006). A GP models a function f as a collection of random variables, where any finite subset of the random variables has a joint Gaussian distribution. A GP is fully specified by its mean function $m(x)$ and covariance function $k(x, x')$: $f(x) \sim \mathcal{GP}(m(x), k(x, x'))$. By conditioning on observed data, GPs can be used to make predictions about the function f at unobserved points.

GPs are particularly useful where uncertainty quantification is desired, and the number of data points is limited (Rasmussen and Williams, 2006). GPs provide a closed-form posterior over functions in the standard setting, assuming a Gaussian likelihood and a Gaussian prior over functions. Though the posterior is closed-form, a naive implementation of exact GPs requires inverting a covariance matrix, which scales as $\mathcal{O}(n^3)$ in the number of data points n. In certain cases, structure in the covariance matrix can be exploited to reduce this complexity significantly. Adding

a single new data point to a GP can be achieved with $\mathcal{O}(n^2)$ complexity using a Cholesky rank-1 update (Seeger et al., 2004). Approximate methods such as sparse GPs (Snelson and Ghahramani, 2005; Titsias, 2009) and string and membrane GPs (Samo and Roberts, 2016) can further improve scalability, allowing GPs to be used with datasets up to millions of data points. For inputs on a fixed grid, Toeplitz structure can be exploited to reduce the complexity to $\mathcal{O}(n \log n)$ (Wilson et al., 2015). These scalability methods tend to focus on big data analytics rather than mobile robotics, where the number of data points is generally smaller but inference speed with limited computational resources is more important.

GPs are well-studied in the contexts of Bayesian optimisation (BO) (González et al., 2016; Morere et al., 2017), motion planning (Mukadam et al., 2016; Meng et al., 2022), and non-RL control of large unknown MDPs (Imani et al., 2018). In the RL setting, GPs have been used to model value functions directly (Kuss and Rasmussen, 2003). Gaussian Process Dynamical Models (GPDMs) were used as dynamics models for early model-based RL methods (Dallaire et al., 2009; Deisenroth and Rasmussen, 2011), where the GP models the first-order Markov dynamics of a system. In MCTS, GPs have been used for tree action sampling for the agent (Mern et al., 2021) or for an adversary in a risk-averse setting (Rigter et al., 2021b).

A strand of works have used GPs to model the safety of RL-derived policies (Berkenkamp et al., 2017; Wachi et al., 2018) or the safety of individual states and actions in an MDP (Turchetta et al., 2016; Wachi and Sui, 2020). With assumptions such as Lipschitz continuity of the underlying safety function, this allows for probabilistically guaranteed safe exploration of the MDP. In Chapter 4, we use GPs to develop our own safe exploration algorithm that improves upon these methods. In Chapter 5 we use GPs to model environment dynamics in a similar manner, but treat it explicitly as a belief model for Bayesian RL.

2.3 Metareasoning

Metareasoning is a diverse field of study, with many different problem settings and approaches. Our focus is on metalevel control of *anytime* planning algorithms,

which are algorithms that can be interrupted at any time to produce a solution.

2.3.1 Metalevel Control

In the metalevel control setting, a metalevel agent observes the reasoning process of an object-level agent. The metalevel agent's actions are to control, change, or halt the object-level agent's reasoning process (Cox and Raja, 2011). Most commonly, the objective of the metalevel agent is to optimise the total rewards achieved by the object-level agent, minus the costs incurred through its reasoning. Costs may be time or resource costs, or some finite budget that must be split between competing reasoning strategies (Callaway et al., 2018).

Metalevel MDPs

To aid in understanding the metalevel control problem, we define a simple metalevel MDP in a similar manner as Callaway et al. (2018). The simple metalevel SSP MDP $\mathcal{M}^M$ is defined as a tuple $\langle K^M, \overline{k}^M, A^M, T^M, R^M, G^M \rangle$, where:

- $K^M = K \cup \{\text{DONE}\}$ is the set of states, corresponding to the quality of the solution $\kappa(k)$ produced by the object-level agent and a reserved final state DONE,
- $\overline{k}^M$ is the initial state, corresponding to the initial quality of the solution,
- $A^M = C \cup \{\text{EXEC}\}$, where C is the set of computations that can be run using the object-level agent, and EXEC is the action of executing the current solution,
- $T^M(k, a, k')$ is the transition function, where $k' \in K$ if $a \in C$ and $k' = \text{DONE}$ if $a = \text{EXEC}$. I.e., if a computation action is chosen, the state transitions to the new quality of the solution k', and if the action is to execute the current solution, the state transitions to the final state DONE,
- $R^M(k, a) = -\lambda$ if $a \in C$, where λ is a fixed cost of reasoning for one timestep, and $R^M(k, \text{EXEC}) = \kappa(k)$, and

- $G^M = \{\text{DONE}\}$ is a single goal state.

This defines a metalevel MDP for a single object-level problem instance, where the object-level agent has produced a solution of quality $\kappa(k)$ when it is in state k. The metalevel agent has the choice between paying a fixed cost λ to run a computation $c \in C$, which will change the solution quality to k', or executing the current solution, which will give the reward $\kappa(k)$ of executing the solution. One way that this model is simple is that it assumes that the evolution of quality depends only on the previous quality of the solution and the computation action taken.

If only one computation action is available, i.e. $C = \{\text{COMPUTE}\}$, then the expected evolution of quality over repeated computation actions can be represented as a function $f(t) = \mathbb{E}[\kappa(k) \mid t]$, where t is the number of computation actions taken. This is commonly referred to as a *performance profile*: the expected quality of solution given time t spent reasoning. Rather than defining a computation cost λ, some metalevel control methods instead define a utility function $U(\kappa(k), t)$, which is the expected utility of the solution quality $\kappa(k)$ at time t (Svegliato et al., 2018, 2020; Bhatia et al., 2022).

The optimal policy for the metalevel MDP is clearly the optimal policy π^{M*} in the simple metalevel SSP MDP we have defined. However, in any realistic setting, the transition function T^M is not known *a priori*. It may be possible to calculate it for a given problem instance, but this is impractical if the aim is to create an agent that can efficiently metareason across many different problem instances. It can be shown that constructing T^M for a given problem instance is at least as hard as solving the original problem (Lin et al., 2015).

Due to the complexity of determining the optimal metalevel policy π^{M*}, a common approximation is the *meta-greedy* or *meta-myopic* policy (Hansen and Zilberstein, 2001). This is a one-step lookahead policy, where the metalevel agent decides only between immediately executing the current solution or taking one more computation action before executing the solution. For computation actions, the

myopic value of computation VoC_1 is defined as the expected increase in the quality of the solution after the computation action, minus the cost of the computation action:

$$\mathrm{VoC}_{\mathrm{myopic}}(k) = \mathbb{E}_{s' \sim T^M(k,c,\cdot)} \left[\kappa(s') - \lambda\right] - \kappa(k). \tag{2.9}$$

The meta-myopic policy is then to choose the computation action that maximises the myopic value of computation, or to execute the current solution if the myopic value of computation is negative:

$$\pi^{\mathrm{myopic}}(k) = \begin{cases} \mathrm{EXEC}, & \text{if } \max_{c \in C} \mathrm{VoC}_{\mathrm{myopic}}(k) < 0, \\ \arg\max_{c \in C} \mathrm{VoC}_{\mathrm{myopic}}(k), & \text{otherwise.} \end{cases} \tag{2.10}$$

Although Equation (2.9) still contains the unknown transition function T^M, it is possible to estimate the myopic value of computation using, for example, an expected performance profile where the expectation is also taken over the distribution of problem instances. The meta-myopic policy can be shown to be optimal in the case of diminishing returns (Svegliato et al., 2018), where the expected rate of increase in quality of the solution is a decreasing function of the time spent reasoning and thinking costs are non-decreasing.

2.3.2 Metalevel Control Algorithms for Deterministic Decision-Making

Hansen and Zilberstein (2001) use dynamic programming to solve a metalevel MDP, described as a "dynamic performance profile". They estimate T^M using a dataset of previously observed performance profiles. This requires discretising the space of solution qualities, and their method's complexity scales quadratically with the number of discretisation points. It also requires significant precomputation to build T^M, conceptually similarly to the requirement to pre-train deep RL policies.

Svegliato et al. (2018) use online nonlinear regression to predict performance profiles. This avoids the need to precompute T^M, but means the method cannot learn common structure between different problem instances and incurs significant metareasoning overhead. Taking a different approach, Svegliato et al. (2020) use online model-free RL to learn a metalevel policy.

Bhatia et al. (2022) also use model-free RL to learn metalevel policies. Because they know the value of the solution quality in each state in the metalevel MDP, they are able to define a metalevel reward which is the difference between the cost of computation and the improvement in solution quality. This is a denser reward than the metalevel reward we use in the simple metalevel MDP above and in Chapter 6, which is only available at the end of the reasoning process. We use this work's approach to defining computation actions in terms of object-level algorithm hyperparameters in our algorithm in Chapter 6.

In a motion planning setting, Sung et al. (2021) take a different approach to metalevel control of the optimal execution time for a computed motion plan. They take an imitation learning perspective, calculating the optimal stopping point of full reasoning traces and using this as an expert demonstration. They also pass information about the object-level problem to the metalevel agent, in a similar manner as we do in Chapter 6.

Situated Temporal Planning

In temporal planning, actions have durations and deadlines and the aim is generally to find a valid plan to reach a goal state. In *situated temporal planning* (Cashmore et al., 2018), the agent's reasoning process takes time. Situated temporal planning works use metareasoning to generate satisficing plans (Shperberg et al., 2019), or cost-effective plans (Shperberg et al., 2020), given that deadlines for starting actions may be missed if too much time is spent planning.

In this setting it is key to identify when partial plans are unlikely or impossible to execute in time, even if elaborated into full valid plans. These methods therefore integrate metareasoning capabilities into existing temporal planners, rather than separating the metalevel agent from the object-level agent.

Guiding Search with Metareasoning

In some cases, metareasoning abilities are integrated into object-level decision-making algorithms in order to improve their performance. Several methods use value of computation predictions to focus search effort on areas of the search space

with high estimated value of computation. Hay et al. (2012), Sezener and Dayan (2020) and Cope et al. (2023) do this for MCTS, and Gu et al. (2022) do this for an online heuristic search algorithm.

2.3.3 Metalevel Control Algorithms for Probabilistic Decision-Making

Probabilistic object-level problems are more difficult because the quality of the current solution is not necessarily known, and is a distribution rather than a single value. This is discussed in more depth in Chapter 6.

One RL-based method, Bayesian metalevel policy search (BMPS) (Callaway et al., 2018), has demonstrated that metacognitive RL can learn near-optimal metareasoning behaviour for relatively simple problems. Their method calculates multiple approximations, including $\text{VoC}_{\text{myopic}}$, and uses Bayesian optimisation to learn a weighted combination of these features that gives an estimate of the true value of computation.

One practical metareasoning method for probabilistic planning exists (Lin et al., 2015), but makes a meta-myopic simplification for tractability. The method is also specific to the object-level algorithm Bounded RTDP (BRTDP) (McMahan et al., 2005). Changes in the BRTDP value function bounds over the last timestep are used to predict the increase in solution quality after a single additional period of reasoning. This work also includes a complexity analysis to demonstrate why exact optimal metareasoning is impractical.

Metalevel control can be carried out using simpler methods, such as linear interpolation of planning time between similar problems (Kunze et al., 2017). This method aims to control *contract* (Zilberstein, 1995) algorithms, which cannot be interrupted once they start reasoning, rather than anytime algorithms. This setting provides more limited scope for metalevel control, because the metalevel agent cannot monitor and react to the object-level algorithm's reasoning process during reasoning.

3 Offline Pre-Planning by Aleatoric Approximation

	Chapter 3	Chapter 4	Chapter 5	Chapter 6	Chapter 7
Epistemic uncertainty	Environment dynamics	Environment dynamics	Environment dynamics	Planner dynamics	Planner dynamics
Mode	Offline	Replanning	Online	Offline	Online
Model	Timed MDP	EstMDP	BAMDP	Metalevel MDP	Metalevel MDP
Solution method	VI (max reward)	VI (reachability)	MCTS	Deep RL	Deep RL
Data regime	Small/ medium	Small	Small	Large	Large

Table 1.1: Settings, methods and features for each chapter.

The objective of the HUDSON (Harvesting of Underwater Data from SensOr Networks) project was to develop and demonstrate a proof-of-concept for retrieving data from underwater sensor nodes using autonomous underwater vehicles. The core of the project was the probabilistic planning method developed by the author of this thesis and presented in this chapter.

The underwater environment contains high levels of epistemic uncertainty and some aspects of partial state observability. The dynamics of vehicle navigation and acoustic communication are uncertain, and the vehicle's location and the data contents of sensor nodes in the environment are both partially observable.

The system design was highly driven by the agent's embodiment, its sensing abilities, and the high level of uncertainty in the environment. A key aim of the HUDSON project was to achieve autonomous operation in the highly uncertain underwater setting, *without* relying on costly sensors and infrastructure commonly

used by underwater vehicles to mitigate uncertainty. The chosen low-cost underwater vehicle has very limited computing capability, meaning we were restricted to an offline plan-then-act operation mode. To support this, the policy generated offline must be small enough to fit on the vehicle's hard drive, but large enough to cover any state the vehicle may enter during the mission. We therefore developed a method that approximately models epistemic uncertainty and partial observability using only a standard MDP model, keeping the state space size small and ensuring the model can be solved offline to give a complete policy. Epistemic uncertainty is accounted for by using parameterised generative (sample-based) simulators, which are run with a range of parameters to generate a distribution of possible environment behaviours. These are modelled as aleatoric stochastic outcomes in the MDP model. Partial observability and scalability are addressed by novel design of the MDP state space and transition function. The implications and trade-offs of these design choices are discussed in depth in the conclusion to this chapter, in Section 3.1. In summary, representing uncertainty that is epistemic in nature as aleatoric uncertainty leads to implicitly making a rectangularity assumption, where the values of potentially correlated epistemic parameters are considered independent at each state. This means that potentially exploitable online information gathered during the mission cannot be used to improve the policy. Latter chapters improve on this approach by introducing increasingly adaptive behaviour.

An additional contribution is a novel method for modelling stores of reward, incorporating stochastic reward retrieval and uncertain contents initialisation. This approach represents the process of the vehicle retrieving data from underwater sensor nodes, which contain finite but uncertain quantities of data of varying value. The model is formulated as a continuous-time Markov decision process (CTMDP), and enables variable-quality approximation of the data retrieval process in order to control scalability. It can be readily integrated into the MDP-based mission model by transforming the CTMDP into a discrete-time MDP. Overall, the system is the first practical demonstration of autonomous underwater data retrieval by

underwater vehicles, and is shown to outperform existing human-designed strategies in extensive simulation and real-world field trials.

The flow of a single mission, from problem specification to planning to execution, is shown in Figure 6 of the paper. As summarised in Figure 6, the environment beliefs given to the system may be derived from real previous mission data, or may be uninformative priors. For the purposes of Table 1.1, we therefore place this method in a "small/medium" data regime: the method can benefit from real-world data to improve the accuracy of its models, but this is not essential.

The following paper is multidisciplinary, covering decision-making under uncertainty, marine robotics, and underwater communication. It consists of contributions from researchers from several institutions. To aid clarity, sections highlighted in light grey are non-essential for understanding the core planning method developed by the author of this thesis. A high-level description of the method, and the results of narrower scope experiments, was published in (Budd et al., 2022). That paper is not included in this thesis.

Please note that this paper is currently under review and has not yet been accepted for publication in its current form.

Citation:

Matthew Budd, Georgios Salavasidis, Izzat Kamarudzaman, Benjamin Sherlock, Jeffrey Neasham, Bruno Lacerda, Nick Hawes, Alexander B. Phillips, and Catherine Harris. AUV-based data harvesting from underwater sensor networks using probabilistic planning. *Under review, Journal of Field Robotics*, 2025b

AUV-Based Data Harvesting from Underwater Sensor Networks Using Probabilistic Planning

Matthew Budd
Oxford Robotics Institute
University of Oxford
mbudd@robots.ox.ac.uk

Georgios Salavasidis
National Oceanography Centre
Southampton, UK
geosal@noc.ac.uk

Izzat Kamarudzaman
National Oceanography Centre
Southampton, UK
Izzat.Kamarudzaman@noc.ac.uk

Benjamin Sherlock
School of Engineering
Newcastle University
Newcastle-upon-Tyne, UK
benjamin.sherlock@ncl.ac.uk

Jeffrey Neasham
School of Engineering
Newcastle University
Newcastle-upon-Tyne, UK
jeff.neasham@ncl.ac.uk

Bruno Lacerda
Oxford Robotics Institute
University of Oxford
bruno@robots.ox.ac.uk

Nick Hawes
Oxford Robotics Institute
University of Oxford
nickh@robots.ox.ac.uk

Alexander B. Phillips
National Oceanography Centre
Southampton, UK
abp@noc.ac.uk

Catherine Harris
National Oceanography Centre
Southampton, UK
catherine.harris@noc.ac.uk

Abstract

Underwater acoustic and wireless sensor networks enable transformative spatio-temporal ocean data collection for applications such as climate monitoring, marine life analysis, and industrial diagnostics. However, retrieving data from submerged sensor nodes in near real-time remains challenging due to environmental constraints and the high costs of traditional methods, such as deploying research vessels. This work presents an integrated adaptive framework that combines modelling approaches and planning under uncertainty to enable an Autonomous Underwater Vehicle (AUV) to serve as a mobile sink for sparse underwater sensor networks. The framework dynamically prioritises nodes based on data value, and utilises acoustic communication links for simultaneous data harvesting and time-of-flight localisation. Key contributions include the systems integration effort that addresses real-world operational challenges, a novel modelling approach that underpins the planning methodology, and extensive field trials validating the framework's performance. The results demonstrate significant improvements over conventional hand-designed behaviours for underwater data retrieval, advancing sustainable and autonomous ocean monitoring solutions through robust integration, innovative planning, and real-world validation.

Keywords: Underwater Sensor Networks, Autonomous Underwater Vehicles, Probabilistic Planning, Data Harvesting, AUV navigation, Underwater communication, Robot decision-making

(a) ecoSUBm5 AUV alongside 8 smart sensor nodes (foreground); preliminary tests of the system in July 2021.

(b) Recovery of the ecoSUBm5 AUV post mission in November 2021.

Figure 1: Images from project field trials at the trial site, Loch Ness, Scotland.

1 Introduction

Ocean science has traditionally relied on large research vessels for both *in-situ* data collection and retrieving information from fixed underwater sensor nodes. Static devices, such as oceanographic moorings (McCarthy et al., 2020) and fixed-point observatories (Cristini et al., 2016) have been integral to observing the marine environment, supporting applications like long-term climate monitoring (Henson et al., 2016) and industrial asset management (Jones et al., 2019). These systems provide temporal coverage spanning months or even years, which mobile platforms cannot achieve. However, retrieving data from these networks often requires deploying research vessels, a process that is both logistically demanding and carbon-intensive, thereby impeding efforts to transition to more sustainable practices.

The urgency to modernise ocean monitoring was underscored by the G7 Ocean Decade Navigation Plan in May 2021, which emphasised the need to expand observing capacities while transitioning to low-carbon models. Achieving this vision requires integrating smart underwater sensor networks, autonomous marine robots, and efficient data pipelines to create persistent, sustainable monitoring systems. The integration of these technologies will enable generation of vast amounts of oceanic observation data, leading to the rise of Big Marine Data (BMD). BMD presents unique challenges to designing a capable sub-sea data infrastructure to meet the demand. Addressing these challenges involves advances in low-power underwater sensing, reliable acoustic communication, intelligent robotics, and sophisticated Machine Learning (ML) techniques to enable efficient knowledge extraction and decision support (Jahanbakht et al., 2021).

Static Underwater Acoustic Sensor Networks (UASNs) (Heidemann et al., 2006) represent a pivotal component of this transformation, providing the scalable infrastructure necessary to support the vision of persistent monitoring. These networks, composed of seabed moorings, floats, and fixed sensor nodes communicating via acoustic links, offer an effective solution to the growing demand for data collection and monitoring in remote, hard-to-reach ocean areas. One of the notable strengths of UASNs is their potential to provide near real-time data telemetry (Kilfoyle and Baggeroer, 2000), achieved through strategic network topologies and multi-hop routing (Li et al., 2016; Blanc, 2020). By leveraging Internet-of-Things (IoT) technologies in underwater environments (Mohsan et al., 2023), UASNs can offer broad applications, including water quality monitoring, seismic tracking, marine animal observation, and offshore asset monitoring (Felemban et al., 2015; Mohapatra et al., 2012).

However, UASNs face several challenges, primarily stemming from the complexity of underwater environ-

ments (Awan et al., 2019). Acoustic communication in these settings is hindered by factors such as low bandwidth, high propagation delay, multi-path interference, fading, and environmental variability, all of which degrade the quality of acoustic signals (Zia et al., 2021). Additionally, the degradation of acoustic communication reliability, due to the intricate underwater physics, is further compounded by practical constraints that hinder direct communication. For instance, oceanographic moorings may lack surface expression, making the network partially or completely inaccessible, particularly in challenging environments such as ice-covered or deep waters, unless supported by specialized mobile technology. Additional challenges include the limited battery capacity of underwater devices, sensor node failures, and uneven power consumption across the network as central nodes may be subject to high-power demands for data relay, shortening network lifetimes. In dense networks, traditional acoustic methods may no longer be sufficient, prompting the need for alternative communication techniques, such as optical models, to facilitate rapid data offloading by mobile platforms visiting network nodes (Pontbriand et al., 2015). These challenges are further amplified in large-scale deployments, where nodes may be sparsely distributed or disconnected, increasing the communication-related issues (Akyildiz et al., 2004).

Addressing the challenges of timely data retrieval often requires the use of mobile platforms. Both Autonomous Underwater Vehicles (AUVs) and Uncrewed Surface Vehicles (USVs) have been employed as mobile sinks to harvest data from underwater sensor nodes (Favaro et al., 2012; Zhuo et al., 2020; Nam, 2018; Lv et al., 2018). Additionally, these platforms can enhance network routing and extend network lifetime by acting as mobile sensor nodes, performing tasks such as data relay and cluster management (Wei et al., 2021). In some cases, AUVs and USVs are deployed in tandem, leveraging their complementary capabilities to efficiently retrieve data from sensor nodes (Cheng et al., 2023). While both platforms offer distinct advantages, the ability of AUVs to operate beneath the surface makes them indispensable for data harvesting in environments where surface access is limited by factors such as sea ice or extreme water depth.

AUVs are increasingly utilised in ocean science (Wynn et al., 2014), but their deployment is often constrained by limited onboard computational power, autonomy, and energy resources. Ensuring mission success and safeguarding AUVs in complex, remote environments is challenging, particularly given the limited contingencies currently available, such as weight dropping or mission aborts in response to environmental factors. Current operational paradigms often depend on predefined mission plans, restricting the ability to make real-time decisions, such as determining the optimal order for visiting sensor nodes, which can lead to delays in retrieving time-sensitive data. Furthermore, navigation remains problematic in GPS-denied environments, and communication constraints further hinder adaptability (González-García et al., 2020). Navigation uncertainty arises from factors such as water currents, vehicle dynamics, and sensor biases (Salavasidis et al., 2021). While advanced navigation sensors like Inertial Navigation Systems (INS) combined with Doppler Velocity Logs (DVL) can significantly mitigate these errors, their high cost and energy demands often limit their practicality in cost-sensitive platforms or long-duration missions. Acoustic position feedback systems offer an alternative, but their reliance on extensive and expensive infrastructure makes them unsuitable for large-scale or remote deployments (Paull et al., 2013). Enhancing AUV autonomy is therefore essential for the efficient harvesting of data from a UASN. This involves improving adaptive navigation systems, real-time lightweight decision-making, and resilience to communication disruptions. These advancements would allow AUVs to better adapt to environmental variability, evolving mission priorities, and communication disruptions, ultimately reducing reliance on costly infrastructure and external interventions while improving the efficiency and reliability of data retrieval.

In response to this demand, we present an integrated adaptive framework developed and tested as part of the Harvesting of Underwater Data from Sensor Networks (HUDSON) project. This framework incorporates modelling approaches and uncertainty-aware planning to transform the ecoSUB low-power AUV (Phillips et al., 2017) into an effective mobile sink for sparse UASN, addressing key challenges in underwater data harvesting with precision and adaptability. By incorporating vehicle dynamics, communication constraints, and environmental uncertainties into its decision-making, the framework allows the AUV to dynamically prioritise sensor nodes based on data value as well as utilising acoustic communication links for simultaneous data harvesting and time-of-flight localisation. The planning framework produces a *policy* that accounts for these uncertainties, enabling the AUV to achieve adaptive behaviour during execution without the need for

computationally intensive online planning.

The key contributions of our work are (i) a comprehensive system development and integration effort that addresses real-world operational challenges; (ii) a novel modelling approach for planning under uncertainty with this system, and (iii) extensive field trials (Fig. 1) validating the framework's performance. Both simulated experiments and real-world trials demonstrate significant improvements over conventional hand-designed behaviours for underwater data retrieval, paving the way for more sustainable and autonomous ocean monitoring systems.

2 Related Work

Several existing works have investigated data retrieval or underwater navigation decision-making. One existing work (Cashmore et al., 2014) is conceptually similar to ours in its usage of waypoints and time-aware planning. This work generates abstracted underwater waypoints using probabilistic roadmaps (PRMs), and executes underwater inspection missions between these using a temporal planner. The temporal planner accounts for action execution time uncertainty, but more general action outcome uncertainty is handled by replanning whenever actions fail. This requires on-board computation, which requires more powerful on-board computers. As well as adding to the size and expense of the AUV, this also increases the power consumption, which is a significant concern for underwater vehicles. By comparison, our method reasons about higher levels of uncertainty in navigation, communication and data contents and does not require online replanning. Existing AUV data retrieval works generally do not consider navigation and communication uncertainty, and simply plan shortest touring paths between sensor nodes (Duan et al., 2020). Some do consider heterogeneous data types or values (Duan et al., 2020; Yan et al., 2018), but assume full prior knowledge of the data contents of the sensor nodes. Our method does not make this assumption, and must reason about uncertainty over the data contents of sensor nodes. A similar data muling problem has been analysed in the context of autonomous aerial vehicles (Gong et al., 2018). Their method divides a touring path into segments, where an aerial vehicle retrieves data from one sensor node during each segment. They use a radio channel model, rather than an acoustic channel model in our work, and alter the vehicle's speed to ensure all data is retrieved from each sensor node. However, this work assumes deterministic navigation and full knowledge of the data contents of the sensor nodes, which is not the case in our problem setting.

In this work we make use of the Markov decision process (MDP) family of models. These models are commonly used for autonomous decision-making in environments with high stochasticity (Lacerda et al., 2019), including some previous works which use kinematic AUV navigation simulators in a similar manner as we do (Yan et al., 2018; Hollinger et al., 2012; Hollinger et al., 2016). By formulating components of the AUV mission planning problem as MDPs, we assume that the current state is known, and stochastic action outcomes are represented by fixed, known probabilities. Due to the limited computational resources of the AUV, we are restricted to pre-planning methods that do not require significant online computation.

The Partially Observable MDP (POMDP) is an MDP generalisation where the system's current state is unknown and must be inferred from observations. This formulation would be well-suited to representing the AUV's localisation uncertainty and the uncertainty in the data contents of sensor nodes. However, due to their computational complexity, POMDP methods generally rely on online computation to scale to large problems (Kurniawati and Yadav, 2016; Sunberg and Kochenderfer, 2018). We therefore develop a method of abstracting the localisation and node data contents state space while retaining the MDP framework. This approach encapsulates key aspects of state uncertainty while maintaining computational tractability.

Our problem includes parameters with uncertain values, such as the effect of water currents on the AUV's navigation. This type of uncertainty is known as *epistemic uncertainty*, and MDP generalisations exist which explicitly consider this type of uncertainty. For instance, Bayesian methods maintain and iteratively update a belief distribution over environment parameters (Duff, 2003; Budd et al., 2023), enabling an agent to use observations to reduce epistemic uncertainty. However, ocean current models are generally computationally

intensive and complex, making them infeasible to be represented and updated in real-time by our AUV. Additionally, Bayesian methods are beneficial only if the information that can be acquired during one mission meaningfully improves the AUV's performance in the rest of the mission. Due to the very high stochasticity of underwater navigation, the AUV's ability to learn about the environment during a single mission is very limited. Instead, our system uses probability distributions over uncertain parameters and samples from these distributions when running the communication model (Section 4.3.1) and navigation simulator (Section 4.3.2). This is a similar concept to domain randomisation in robotics (Peng et al., 2018), where a controller is trained on a distribution of environments to improve its generalisation to unseen environments. The generated policy is therefore more robust to the real dynamics of the environment than a policy generated using a single set of parameters. This is particularly useful as the behaviour of the real environment will never perfectly match the simulator due to unmodelled effects. Real environment interaction data gathered from each mission can be used to update the distribution over uncertain parameters, improving the accuracy of the simulators for future missions.

3 A System for Underwater Data Harvesting

HUDSON introduces a novel framework to enable an AUV to autonomously harvest measurements from stationary smart sensor nodes within a sparse UASN via acoustic communication. While retrieving data, the AUV integrates its internal state awareness with considerations of communication and navigation uncertainties, allowing it to dynamically prioritise mission objectives and react to changing environmental conditions. In this section, we provide an overview of the HUDSON system, considering both the design concepts and the hardware implementation. We describe four main modules (i) the underwater sensor network; (ii) the AUV hardware and software; (iii) the communication between the nodes of the UASN and the AUV; and (iv) the software architecture for the data retrieval planner.

3.1 Underwater Acoustic Sensor Network

3.1.1 Overview

The UASN in our framework consists of "smart" sensor nodes equipped with acoustic communication capabilities to interface with the AUV. These nodes record environmental measurements of interest, such as temperature, salinity, and currents, and can adapt their sampling frequency based on observed environmental dynamics (Yan et al., 2018; Chou et al., 2009). For instance, a node may increase its sampling rate when detecting rapid environmental changes, or it may focus on capturing infrequent but high-value events such as passing marine life.

Each data type stored by the sensor nodes has an associated *utility* per byte, reflecting its scientific or operational importance. Nodes prioritise offloading high-utility data first during interactions with the AUV. However, the adaptive behaviour of the nodes means the precise data utility distribution within the UASN is unknown to the AUV at the start of its mission. Consequently, the AUV's objective becomes maximizing the *expected total utility* of the data it retrieves, requiring decision-making that is reactive to the data contents of sensor nodes throughout the mission.

To mitigate localisation errors inherent in underwater navigation, the AUV can acoustically ping sensor nodes in a round-robin fashion to obtain range measurements. These measurements serve as feedback to correct navigation drift, ensuring reliable operation within the acoustic range of the UASN.

3.1.2 NMv3 acoustic modems

The AUV and the underwater sensor nodes are all integrated with a low-cost, low-power, miniature acoustic modem, NMv3, used for ranging & navigation, and communications during operations (Sherlock et al.,

Figure 2: USMART Sensor Node showing the electronics and battery pack within the canister.

2022). The NMv3 acoustic modem operates in the 24 kHz to 32 kHz band with a transmit power source level of 168 dB re 1 µPa @ 1 m and receive power consumption of 12.5 mW. An effective data rate of 470 bit/s with variable length payloads of 2 – 64 bytes. Operating range can be up to 2.3 km depending on acoustic channel conditions. The acoustic modem firmware was updated to provide diagnostics features including ambient noise measurement, and link quality indicators on received packets.
Available modem transmissions include: unicast message to a specific address; unicast message with ack whereby the receiving modem will send an acknowledgement; broadcast message whereby all modems within range will receive the message; ping to a specific address where a receiving modem will respond with an acknowledgement that can then be used to determine the time of flight.

3.1.3 Sensor network nodes

Experimental prototype sensor nodes designed and developed for the USMART project, as described in (Morozs et al., 2022), were utilised to form the HUDSON sensor network. These sensor nodes contained electronics within diver canisters connected to NMv3 acoustic modems. Each sensor node contains a bespoke PCB with power regulators, switches, and sensors (e-compass, temperature, pressure, humidity). Attached to the PCB is a MicroPython Pyboard D-series (PYBD) with STM32F767 processor. The node is powered from four alkaline C cells. An open sensor node is shown in Fig. 2. When the node and modem is in listening mode awaiting incoming acoustic messages the power draw is 19.8 mW. This allows the nodes to be left deployed between mission runs with minimal power load on the batteries. The NMv3 modem when powered also acts as a transponder which is used as part of the navigation system in this work.
In real world applications it is likely that a sensor node would incorporate more environmental sensing capability for example to measure salinity, dissolved oxygen levels, temperature etc. The sensor nodes used in these experiments are simplified units, with more trivial internal sensors, with a focus on testing the communication algorithms and the concept in general.

In our setup, nodes are capable of transmitting three types of acoustic packets: a *data transfer* packet, which typically has a variable size, but is fixed in size for the purposes of our experiments, a *localisation* packet of minimal size that responds to a ping request from the AUV to estimate the range to the node, and a *status update* packet, which conveys information about the node's internal state as well as the data statistics information used to plan data retrieval (Section 4.2.1).

Figure 3: The ecoSUBm5, a micro AUV used in the HUDSON system for underwater data harvesting and navigation tasks.

3.2 AUV

3.2.1 The ecoSUB vehicle

The HUDSON concept is independent of the AUV being used. The concept is adaptable to other types or sizes of AUVs as long as they can navigate to the sensor nodes and harvest data. This study, however, utilises the small and low-cost ecoSUB microAUV (Phillips et al., 2017). Specifically, the ecoSUBm5 unit deployed has a mass of <12 kg, a depth rating of 500 m, and an overall length of 1 m. The hull shape of all ecoSUB variants is based on Myring's equations (Myring, 1976), giving it an axis-symmetric form with a ducted propeller and a vertical rudder at the rear.

The control system is under-actuated, using an aft propeller for forward thrust, a moving-mass mechanism to control pitch, and a rudder to control the yaw angle. The vehicle is ballasted to be slightly positively buoyant, ensuring that the antenna stays clear of the water when stationary on the surface and increasing the likelihood of the AUV returning to the surface in the event of a failure while submerged. This combination of positive buoyancy and under-actuation requires the AUV to maintain a constant forward speed to sustain its depth. This has implications for acoustic communication, as the continuously running propeller generates both acoustic and electrical noise in the communications channel.

The ecoSUBm5 is equipped with a limited set of low-cost, low-powered navigation sensors that allow the vehicle to navigate via GPS on the surface and dead reckon while submerged. The vehicle's orientation is measured using a Bosch BNO55 9-axis orientation sensor, which integrates a tri-axial 14-bit accelerometer, a tri-axial 16-bit gyroscope (with a range of ±2000° per second), a tri-axial geomagnetic sensor, and a 32-bit microcontroller running Bosch BSX3.0 FusionLib software. Depth is inferred from a Keller PAA-11LX absolute pressure piezoresistive pressure transducer, which has an error margin of less than <0.05 % of full scale. For GPS positioning when on the surface, the ecoSUBm5 is equipped with a Venus638FLPx GPS unit.

For underwater acoustic communication, an NMv3 transducer is mounted on the AUV's nose, facing upward (see Fig. 3). Additionally, the vehicle is equipped with a Valeport Hyperion Fluorometer and a Valeport Fast Response Temperature sensor for environmental data collection.

3.2.2 Software architecture

The ecoSUB unit used in this research is powered by an Intel Edison module, which features a dual-core, dual-threaded Intel Atom Central Processing Unit (CPU) operating at 500 MHz, coupled with a 32-bit Intel Quark microcontroller at 100 MHz. The software architecture follows the front-seat/back-seat paradigm (Eickstedt and Sideleau, 2009), enhancing software portability by decoupling the high-level autonomy system from the lower-level Guidance, Navigation, and Control (GNC) system. This architecture is implemented using the Robot Operating System (ROS) middleware (Quigley et al., 2009).

The front-seat serves as the core of the system, providing critical functionalities such as guidance, control, navigation, and health and safety monitoring. Conversely, the back-seat extends the system's capabilities by

enabling high-level autonomy and decision-making algorithms tailored to specific application requirements. The back-seat system is capable of generating demand requests for the front-seat controllers, including parameters such as heading, depth, speed, and other operational settings. It can also request satellite communication for message transmission when the vehicle is at the surface. In addition, the back-seat may request vehicle status updates from the front-seat, including localisation data, sensor measurements, and battery levels, to support its operations and ensure system efficiency. To maintain vehicle safety, the front-seat has the authority to grant, revoke, or reclaim control from the back-seat if any commands from the latter are deemed to compromise safety (Fenucci et al., 2022).
A key functionality of the back-seat system relevant to this work is the implementation of underwater communication and acoustic-aided localisation. Initially developed for multi-vehicle cooperation (Fenucci et al., 2022), this system has been refined to meet the specific demands of this research. The communication algorithm facilitates the transmission and reception of messages via the acoustic channel, while the acoustic-aided localisation system leverages this communication to obtain range observations. By dynamically scheduling messages and adapting transmission modalities, the system acquires additional localisation data without disrupting network functionality. Equipped with acoustic modems, the underwater network nodes allow the AUV to estimate distances through a point-to-point interrogation mechanism. The AUV periodically sends unicast requests in designated time slots, receiving immediate acknowledgements from the nodes to minimize turn-around time. Using an estimated sound speed, the measured two-way travel time is converted into range measurements to each node. These range measurements, combined with the known positions of the nodes, are integrated using an Extended Kalman Filter (EKF) running on the back-seat, ultimately refining the front-seat's dead-reckoning navigation estimates (Fenucci et al., 2022).

3.3 Data retrieval

3.3.1 Overview

The AUV retrieves data from sensor nodes using a polling mechanism, across one or more *interrogation cycles*. Each interrogation cycle is started by the AUV, and the underwater nodes respond by sending data transfer packets to the AUV. A time gap T_{gap} is left between each interrogation cycle to allow localisation and status update packets to be exchanged between the AUV and the nodes.
A single interrogation cycle is visualized in Fig. 4, beginning with a *Poll* message requesting M_{train} data transfer packets of a specified size. While the number of packets and their size can vary depending on factors such as link quality, they are fixed for the purpose of our experiments. Upon reception of the request, the target underwater node starts the data offloading process by sending initially a short status update packet, followed by a train of M_{train} data transfer packets, and concludes with an End of Data (EOD) packet to signal the completion of the transmission. At the reception side (AUV), all successfully received and decoded packets are acknowledged after a pre-determined transmission threshold time T_{timeout}. If all M_{train} packets are successfully received earlier, then the acknowledgement is sent instantly without any further delays. Then, acknowledged data transfer packets can now be removed from the node's buffer. However, packets that were lost during transmission, or those that the modem was unable to decode, will be maintained in the buffer to be re-transmitted in the future. Table 1 presents the parameter values used in the implementation of the data retrieval process. It includes the specific configurations and settings that define how data is transferred, such as packet size, the number of packets, and other relevant parameters that influence the retrieval process during experimentation.
As well as data retrieval, the AUV periodically requests underwater nodes to provide status update messages. These messages contain information about the amount of different types of data that the sensor node has recorded and stored (with the associated *utility* per byte) and an estimate of the rates at which the node generates new data. The AUV can use these status updates to maintain an estimate of the data distribution in the network, which is used by the data retrieval planer to determine the next optimal AUV action.

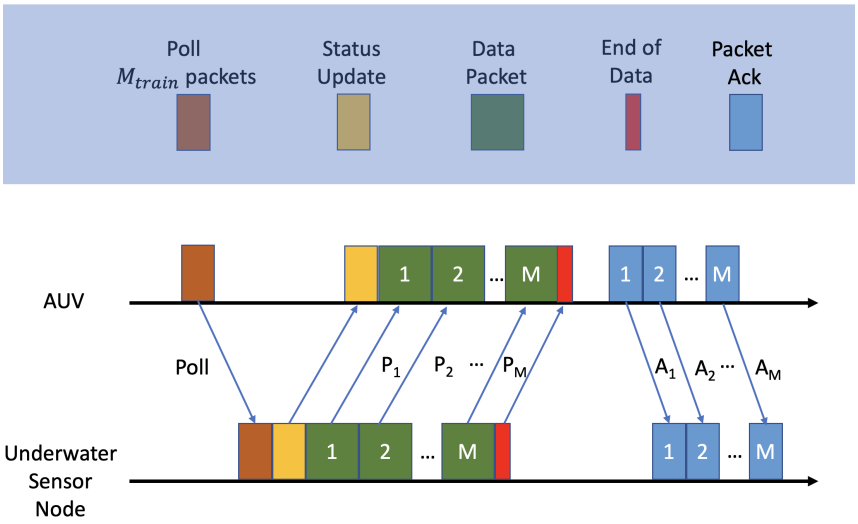


Figure 4: Visualisation of the data offloading (polling) process.

3.3.2 Acoustic communications model

Simulation results for the acoustic communication signals in Additive White Gaussian Noise (AWGN) and multipath channels (Sherlock et al., 2022) were used to produce lookup tables for packet success probabilities in a communications performance model. For given parameters such as straight-line range, receiver depth, payload length, ambient noise conditions, alpha attenuation, and a discrete level (none, low, high) of estimated multipath interference, a probability of successful packet delivery is produced. This model is used by the planner based on predicted environmental conditions as well as during the live mission using real-time noise measurements from the modem on the AUV. Incoming packets received at the AUV also provide link quality indicators (LQI) to provide additional metrics that relate to the Signal-to-Noise Ratio (SNR) from noise and multipath arrivals. The expected low speeds of the AUV fall well within the Doppler tolerance of the acoustic waveforms and receiver structures in the modem and will have negligible effect on the packet delivery probabilities.

Packet delivery probability. The lookup tables for a given SNR value will provide a probability of successful packet delivery. To obtain the estimated SNR value at the receiver, the source level (SL) of 168 dB re 1 µPa @ 1 m, bandwidth of 8 kHz at centre frequency of 28 kHz, and range from transmitter to receiver is used. The sonar equations are used to determine transmission losses (TL) in dB due to free field spreading and attenuation (dB/km) combined with the ambient noise losses (NL) and then subtracted from the initial source level (SL). The azimuth angle from sensor node to the modem on the ecoSUB also has to be taken into account due to occlusion caused by the tail end of the vehicle. As the vehicle travels with the nose angled downwards this causes a reduction in signal strength for acoustic communications arriving from behind. A 10 dB reduction to SNR is applied for signals arriving from beyond 160°.

$$\mathrm{TL} = 20 \cdot \log_{10}(\text{range}) + \text{attenuation} \cdot \text{range} \cdot 0.001 \tag{1}$$

$$\mathrm{NL} = \text{spectral density} + 10 \cdot \log_{10}(\text{bandwidth}) \tag{2}$$

$$\mathrm{SNR} = \mathrm{SL} - \mathrm{TL} - \mathrm{NL} \tag{3}$$

Expected data throughput. The expected data throughput in bytes per second is calculated by considering different cases of i.e., (i) the length of the interrogation cycle in seconds, and (ii) the number of data transfer packets successfully transferred in the interrogation cycle. The interrogation cycle is visualized in Fig. 4.

Parameter	Value	Description
b^{req}	7 B	Payload size of the data request packet.
$b^{payload}$	16 B	Data retrieval content of a data transfer packet.
$b^{overhead}$	16 B	Overhead size of a data transfer packet.
b^{data}	32 B	Payload size of a data transfer packet, consisting of $b^{payload} + b^{overhead}$.
b^{loc}	0 B*	Payload size of a localisation packet. * localisation packets are control messages with no payload.
b^{ack}	6 B	Payload size of acknowledgement packet.
b^{stat}	13 B	Payload size of status update packet.
b^{eod}	5 B	Payload size of end of data (EOD) packet.
M_{train}	4	Number of data transfer packets requested per interrogation cycle.
T_{timeout}	15 s	Time threshold for AUV to wait for all data to arrive.

Table 1: Parameter values for data retrieval process implementation. $b^{payload}$ is separated from b^{data} because only half of the data retrieval packet is used for data content. The rest consists of bookkeeping information which we do not wish to optimise for.

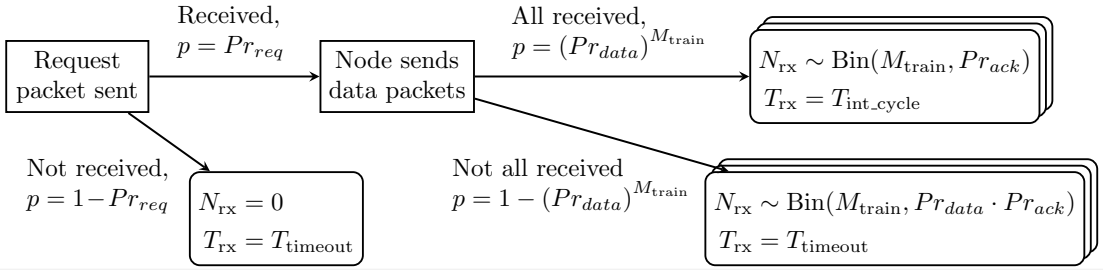


Figure 5: Outcomes probability tree for a single interrogation cycle, giving the number of packets N_{rx} successfully transferred and acknowledged and the time T_{rx} taken to do so. Pr_{req}, Pr_{data} and Pr_{ack} are the respective probabilities of a single request, data, and acknowledgement packet being successfully transferred. $T_{\text{int\_cycle}}$ is the length of a successful interrogation cycle in seconds and Bin() is the Binomial distribution.

Fig. 5 shows the possible outcomes of a single interrogation cycle, defining the distribution of the random variables N_{rx} and T_{rx}. The packet delivery probabilities for each packet type are calculated as $Pr_{req} = Pr(comm \mid b^{req}, \cdot)$, $Pr_{data} = Pr(comm \mid b^{data}, \cdot)$, and $Pr_{ack} = Pr(comm \mid b^{ack}, \cdot)$. Note that the source/destination parameters for $Pr(comm)$, for example the receiver acoustic noise level, will differ depending on the sender/receiver. Additionally, the length of the interrogation cycle $T_{\text{int\_cycle}}$ is distance dependent due to propagation delays.
The expected data throughput in bytes per second, also incorporating the time gap T_{gap} between interrogation cycles, is then calculated as

$$\lambda = \mathbb{E}_{N_{\text{rx}}, T_{\text{rx}}} \left[\frac{N_{\text{rx}} \cdot b^{payload}}{T_{rx} + T_{\text{gap}}} \right]. \tag{4}$$

3.4 AUV Data Retrieval Planning

When an AUV is dispatched on a *data retrieval mission*, its aim is to maximize the total utility retrieved from the sensor network. It does this by navigating between sensor nodes and communicating with them, under some time or battery bound. To achieve efficient data harvesting, a planning algorithm for the AUV

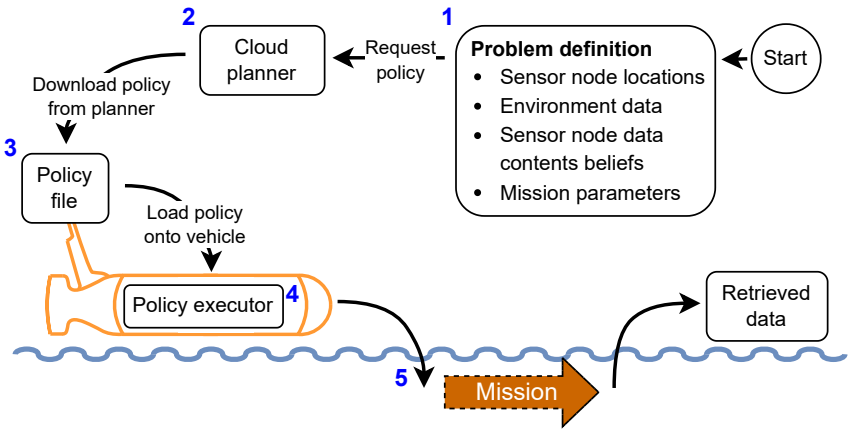


Figure 6: Flow diagram of an AUV data retrieval mission.

has been developed to provide the required onboard autonomy. The workflow for the data retrieval planner is shown in Fig. 6, and is comprised of the following steps:

All inputs to the planner are encapsulated in a *problem definition* (Fig. 6, 1). This contains all information about the vehicle, data network and environment that a user has at the start of the mission. Some aspects of the environment and data network are not known *a priori*, such as water current strength and direction, acoustic channel characteristics, and the data contents of each sensor node. These uncertainties are represented by probability distributions over the relevant parameters (*belief distributions*), rather than fixed values. The planner uses these distributions to reason about the possible outcomes of actions and to generate a policy that is robust to environmental variability and incomplete information. If little prior information is known about the deployment environment, belief distributions can be uninformative (e.g. uniform or wide Gaussian distributions) or can incorporate predictions based on e.g. weather forecasts or a simulation of the sensor network and the data it collects.

The problem definition includes the following components, collectively forming the input to the planner:

- Sensor node locations, which are assumed to be static and known,
- The initial location of the AUV, assumed to be known before the mission,
- Belief distributions over environment parameters (e.g. water current fields), the acoustic channel characteristics, and the data contents of each sensor node,
- Mission parameters such as the mission time bound, the final position the vehicle must reach at the end of the mission, and in-mission reporting requirements,
- Planner parameters: waypoint and action generation parameters (Section 4.4) and the timestep for the navigation model (Section 4.7), and
- Vehicle data: vehicle model, belief distributions over vehicle parameters (e.g. dynamics coefficients, sensor noise), and control system parameters (speed and depth demands, controller gains, timeouts, etc.).

The planning algorithm (2) constructs a model, which is fully described in Section 4, to represent the AUV data retrieval problem. The model accounts for uncertainty in navigation and communication and data contents, and specifies rewards for retrieving data and constraints such as requiring the vehicle to return to the final position before the end of the mission. To construct the model, the planner uses a model of the vehicle and physical environment (Section 4.3.2), and a model of the acoustic communication process (Section 4.3.1). This model is solved to give a reward-maximising policy (3), which is the output of the planner. The processes internal to the planner are illustrated in Fig. 12.

The policy specifies the optimal vehicle behaviour in any situation it may encounter during the mission. By pre-generating a complete policy rather than planning online, the AUV can make decisions quickly and efficiently without the need for additional on-board compute resources. The on-board policy executor (4) reads the policy to determine the most appropriate action for the vehicle at any time based on *in-situ* observations and the mission progress already achieved. To do so, it uses a SQLite database that is loaded onto the AUV prior to the mission. This allows fast lookup of state to action mappings, without having to load the entire large ($\geq 1\,\text{GB}$) policy file from disk into memory.

This workflow allows the AUV to react to new information and events during the course of the mission (5) without further planning. Finally, logged data from previous missions can be used to improve the accuracy of belief distributions for the next mission, forming a feedback loop of increasing performance.

4 Planning under uncertainty for underwater data harvesting

The AUV planning method was originally presented in condensed form in (Budd et al., 2022) and is described in more detail in this section.

4.1 Preliminaries

Definition 1 (Markov Decision Process)**.** A Markov decision process (MDP) is a tuple $\mathcal{M} = \langle S, \overline{s}, A, \delta, R \rangle$, where:

- S is a finite set of states;
- $\overline{s} \in \mathit{Dist}(S)$ is the initial state distribution;
- A is a finite set of actions;
- $\delta : S \times A \times S \to [0, 1]$ is the transition function, specifying the probability of transitioning to state s' after taking action a in state s;
- $R : S \times A \to \mathbb{R}_{\geq 0}$ is the reward function, specifying the reward received after taking action a in state s.

In this work, the AUV's mission-level decisions are specified by a deterministic, stationary *policy*. A policy in an MDP is a mapping $\pi : S \to A$ that specifies the action to take in each state.

Timed MDPs extend MDPs by adding discrete distributions over the duration of actions. They can be seen as a special case of a Semi-MDP (Howard, 1960), where action durations are restricted to a finite set of integer values. We use Timed MDPs to represent the AUV's overall mission, including its stochastic navigation outcomes and its mission time limit.

Definition 2 (Timed Markov Decision Process)**.** A timed Markov decision process (TMDP) is a tuple $\tilde{\mathcal{M}} = \langle S, \overline{s}, A, \delta, T, \Theta, R \rangle$, where:

- S, $\overline{s}$, A, δ and R are as in the MDP definition;
- $T = \{t_1, \ldots, t_{|T|}\} \subset \mathbb{N}_{\geq 0}$ is a finite set of discrete action execution times, which we assume to be in increasing order representing a constant time step; and
- $\Theta = \{\theta_{s,a,s'} \mid \delta(s, a, s') > 0\}$, where each $\theta_{s,a,s'} : T \to [0, 1]$ is a probability distribution over integer durations, representing the time taken to execute action a from s and finish in s'.

We consider TMDP problems with a finite *time-bound* $\beta \in \mathbb{N}_{>0}$. After the time bound, executing actions and receiving additional reward is not possible. Given β, a TMDP can be unfolded into a finite MDP $\mathcal{M}^T_\beta = \langle S^T_\beta, \overline{s}^T_\beta, A, \delta^T_\beta, R^T_\beta \rangle$ via *time augmentation*. This state space of this MDP includes the current timestep: $S^T = \{(s, t) \in S \times \mathbb{N} \mid t \leq \beta + t_{|T|}\}$. The transition function δ^T_β and reward function R^T_β are

defined as follows:

$$\delta_\beta^T\left((s,t),a,(s',t+k)\right) = \begin{cases} \delta(s,a,s')\theta_{s,a,s'}(k) & \text{if } t \leq \beta \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

$$R_\beta^T((s,t),a) = R(s,a) \tag{6}$$

Continuous time Markov Decision Processes (CTMDPs) extend MDPs to represent continuous-time systems. Actions can be taken at any time and rewards are accumulated continuously. We use CTMDPs to model the AUV's data retrieval process from sensor nodes, as, at a high level, the AUV can continuously receive data from a node while acoustically communicating with it.

Definition 3 (Continuous time Markov Decision Process)**.** A continuous time Markov Decision Process (CTMDP) is a tuple $\mathcal{Q} = \langle S, \overline{s}, A, \Delta, \mathcal{R} \rangle$, where:

- S, $\overline{s}$ and A are as in the MDP definition;
- $\Delta : S \times A \times S \to \mathbb{R}_{\geq 0}$ is the rate transition function; and
- $\mathcal{R} : S \times A \to \mathbb{R}_{\geq 0}$ is the reward rate function, specifying the rate that reward is accumulated after taking action a in state s.

If multiple transitions are possible from a state s, a *race condition* occurs between processes corresponding to each possible transition. The duration before the next transition occurs, the *sojourn time*, is exponentially distributed with rate $E(s,a) = \sum_{s' \in S} \Delta(s,a,s')$. The duration of the process that results in a transition to a specific state s' is modelled as an exponential distribution with rate $\Delta(s,a,s')$. The probability of transitioning to state s' given that action a was taken in state s is given by $\Delta(s,a,s')/E(s,a)$.

4.2 Problem formulation

4.2.1 Sensor nodes, acoustic packets and data contents

The UASN consists of a set of sensor nodes $\Phi = \{\phi_1, \ldots, \phi_{|\Phi|}\}$. The known location of each node is $loc(\phi) = (lat, lon, depth) \in \mathbb{R}^3$. These nodes are able to communicate with the AUV via acoustic communication, and potentially with each other if they are within some maximum communications distance. As described in Section 3.1.1, nodes can send 3 types of acoustic packet with constant, known sizes:

- A *data transfer* packet is of size b^{data} and contains $b^{payload}$ bytes of data;
- A *localisation* packet is of size b^{loc} and provides the AUV with a distance estimate to the node;
- A *data statistics* packet is of size b^{stat} and describes the data the node contains.

With some probability the AUV will successfully receive and decode a packet from a node. This probability is dependent on acoustic channel conditions including transmission losses, ambient noise, and multipath arrivals. These can be estimated from the distance between the node and the AUV, the depth of the node, the depth of the AUV, and the environmental conditions (Section 3.3.2).

Sensor nodes contain data of varying types, where each type has a different utility per byte. Data types are denoted by $D = \{d_1, ..., d_{|D|}\}$, and a utility function $U : D \to \mathbb{R}_{>0}$ gives the utility per byte $U(d)$ for data type d. For simplicity, we assume that D is sorted in order of ascending utility. The number of bytes of data type d in node ϕ is denoted by $L(\phi, d)$, where $L : \Phi \times D \to \mathbb{Z}_{\geq 0}$. Given this definition, the total utility of data stored on a node ϕ is $\sum_{d \in D} U(d) \cdot L(\phi, d)$.

Before the mission, the AUV has a prior over the value of L, and only knows the actual value with certainty when it receives a data statistics packet from the node.

4.2.2 Problem definition

Problem 1. Given a set of sensor nodes Φ, a set of data types D, a utility function U, a prior over the data contents of each node $\{Dist(L(\phi, d)) \mid \phi \in \Phi, d \in D\}$, and a time bound β, compute a policy that maximizes the expected total utility of data retrieved from the sensor nodes before reaching the time bound.

Addressing Problem 1 requires solving a number of challenges:

1. **Poor localisation.** Although AUV depth is observable via pressure sensing, the AUV's latitude/-longitude location is unobservable while underwater. The AUV's position is estimated through dead-reckoning, a process prone to cumulative errors caused by unobserved water currents, sensor noise, and uncertainties in the AUV's dynamic model. When the AUV comes within range of sensor nodes, it may receive noisy time-of-flight position feedback. This feedback is then integrated into the navigation system using an EKF to improve positioning accuracy. When on the surface, the AUV's latitude/longitude is available via GPS.
2. **Imperfect communication.** Acoustic communications have a maximum range and are imperfect. Packets, including those for localisation and data transfer, may not be successfully received.
3. **Unknown node data contents.** The AUV has a prior over the data contents of each node, but only knows the actual contents with certainty when it receives a data statistics packet from the node.

For the vehicle localisation and navigation aspect of the system, we address Challenges 1 and 2 by building an uncertain navigation model (Section 4.7) which uses a communication-based abstraction of the AUV's position (Section 4.4.1). For the data retrieval aspect, we address Challenges 2 and 3 by building a stochastic data retrieval model (Section 4.6) that represents the AUV's ability to retrieve data from sensor nodes.

4.3 Planning method inputs

4.3.1 Acoustic communication model

As acoustic communications are used for both AUV localisation and retrieval of data from sensor nodes, the planner uses a model of the acoustic communication channel to reason about these processes. The *comms model* provides two functions:

1. The *packet delivery probability function* $Pr(comm \mid b, dist, depth_{\text{tx}}, depth_{\text{rx}}, \theta_{comm})$ is a function that calculates the probability of successful packet receipt.
2. The *expected data throughput function* $\lambda(dist, depth_{\text{tx}}, depth_{\text{rx}}, \theta_{comm})$ is a function that calculates the expected data transfer rate in bytes per second when retrieving data from a sensor node.

The arguments of the comms model functions are the packet payload size b in bytes (i.e. data bytes only, excluding fixed components such as headers and checksums), the distance between the transmitter and receiver $dist$, the depth of the transmitter $depth_{\text{tx}}$, the depth of the receiver $depth_{\text{rx}}$, and a set of comms model parameters θ_{comm}. Comms model parameters are discussed in detail in Section 3.3.2, and may include the acoustic noise level, multipath interference level and attenuation/sound speed characteristics of the environment.

As mentioned above, when using the comms model, the planner typically operates using a distribution over comms model parameters $Dist(\theta_{comm})$, rather than a single parameter value set, to improve robustness. The implementation of this model and its parameters are described in Section 3.3.2.

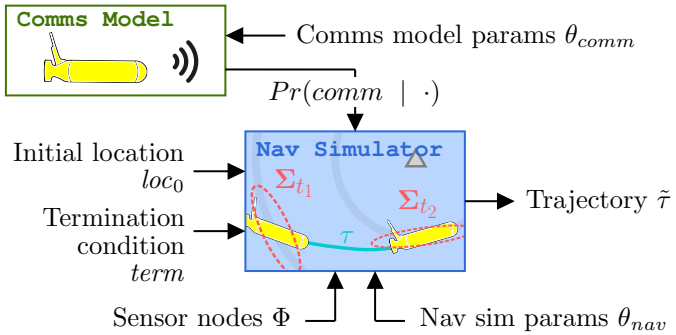


Figure 7: Illustration of inputs to and outputs from the navigation simulator.

Parameter	Description
AUV kinematic parameters	Maximum speed, sensor noise bias values, including noise values for AUV surge, sway and heave, etc.
AUV controller parameters	Navigation parameters, PID controller gains, guidance system parameters (e.g. waypoint radius of acceptance, line-of-sight based heading demand, ...), target depth when travelling underwater, etc.
Water currents field data	Specifies (potentially spatially-varying) water current vectors in operating area.
Bathymetry data	Specifies the depth of the ocean floor in the operating area if available.

Table 2: Key variables in the navigation simulator parameters θ_{nav}.

4.3.2 Navigation simulator

The planner utilises a *navigation simulator* $Nav()$ to model the AUV's navigation process. The simulator accounts for vehicle dynamics, sensor inaccuracies, and environmental disturbances, such as water currents. The simulated vehicle also includes a navigation controller with the same behaviour as the real AUV. We assume this simulator is a stochastic generative function that produces *samples* of navigation trajectories given an initial location loc_0 and some termination condition $term$ on the simulated AUV's state:

$$\tilde{\tau} \sim Nav\left(\tau \mid loc_0, term, Pr(comm \mid b^{loc}, \cdot, \theta_{comm}), \Phi, \theta_{nav}\right). \tag{7}$$

Equation 7 formalises the inputs and outputs of a single call of the navigation simulator Nav. Each simulator run is parameterised by a set of navigation simulator parameters θ_{nav}, which are detailed in Table 2, and makes use of a comms model instance $Pr(comm \mid b^{loc}, \cdot, \theta_{comm})$ to simulate the AUV's ability to receive localisation packets from sensor nodes. The navigation simulator uses the provided comms model function by setting the values of the $dist$, $depth_{\text{tx}}$ and $depth_{\text{rx}}$ parameters, shown as "$\cdot$" in (7), to match the positions and depths of the simulated AUV and sensor node exchanging a localisation packet. A trajectory τ is a sequence of locations $\tau = \{(t_1, loc_1), \ldots, (t_{|\tau|}, loc_{|\tau|})\}$, where t_i is the time at which the AUV is at location $loc_i = (lat_i, lon_i, depth_i)$. These inputs and outputs are illustrated in Fig. 7.

Due to random noise in the simulated AUV's sensors and actuators, and random acoustic packet reception sampling, the simulator produces a distribution of trajectories given the same input parameters. Similarly to the comms model parameters, the planner is provided with a distribution over navigation simulator parameters $Dist(\theta_{nav})$. Practically, sampling navigation simulator parameters entails sampling possible water current fields and vehicle dynamics parameters, as the controller parameters are assumed to be known by the system designer. Bathymetry data need not be given if the vehicle does not closely approach the ocean floor or coastal areas. To sample a single trajectory, the planner samples navigation simulator parameters $\theta_{nav} \sim Dist(\theta_{nav})$ and comms model parameters $\theta_{comm} \sim Dist(\theta_{comm})$, and then generates a trajectory using the navigation simulator.

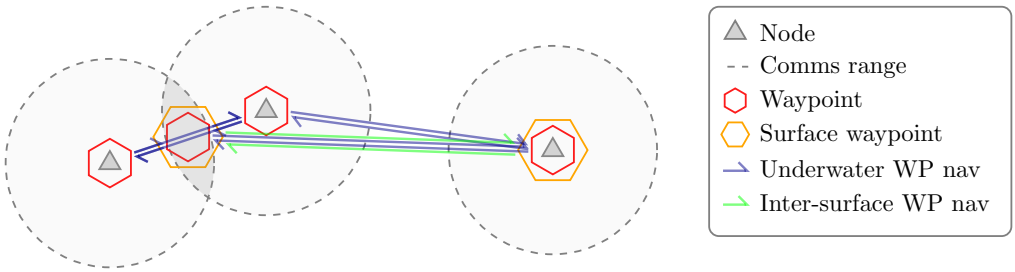


Figure 8: Waypoint placement and navigation actions in an AUV topological map composed of 4 underwater waypoints and 2 surface waypoints. For clarity, only two types of navigation actions are shown: navigation between underwater waypoints and navigation between surface waypoints.

4.4 Navigation model: AUV topological map

The AUV topological map is a tuple $\mathcal{T} = \langle V, E \rangle$ that represents the AUV's operating area as a graph. Topological waypoints V are described in Section 4.4.1, and topological edges E are described in Section 4.4.2.

4.4.1 Topological waypoints: defining locations to visit

We define a topological abstraction of locations in the operating area, rather than specifying e.g. a grid of latitude/longitude coordinates. The topological waypoints V consist of *underwater waypoints* V^u and *surface waypoints* V^s that the AUV can visit.

We start by defining the maximum communication range $d^{comm}_{\max}$ between a node and the AUV using a minimum packet delivery probability threshold on $Pr(comm \mid b^{data}, \cdot, \theta_{comm})$. To account for the distribution over comms model parameters, $d^{comm}_{\max}$ is calculated by taking its expectation over $Dist(\theta_{comm})$. This threshold implicitly defines a minimum data offload rate from a node to the AUV, as described in Section 3.3.2. For simplicity we assume that the max communication range is the same for all nodes, but in practice this assumption could be lifted if necessary (e.g. if nodes' depths or communication abilities vary significantly). This would require more complex optimisation of where to place waypoints than simply taking the centroid of the nodes' locations as we do below.

Underwater waypoints $v^u \in V^u$ are defined by the set of sensor nodes that can be communicated with from that waypoint. That is, waypoints are defined by

$$V^u = \{\Phi' \subseteq \Phi \mid \forall_{\phi \in \Phi'},\ \|loc(\phi), loc(\Phi')\|_2 < d^{comm}_{\max}\}, \tag{8}$$

where $loc(\Phi)$ is defined as the centroid of the locations of each sensor node $\phi' \in \Phi'$. V^u therefore defines a set of underwater waypoints, where each waypoint is a subset of sensor nodes where there exists a location where the AUV can feasibly communicate with all nodes in the subset. Informally and in practice, one underwater waypoint is placed for each sensor node, and additional waypoints are placed in the middle of the overlap area between nodes' communication areas if the overlap area is sufficiently large. Three single-node waypoints and one 2-node underwater waypoint are illustrated in Fig. 8.

Surface waypoints $v^s \in V^s$ have an associated location $loc(v^s) = (lat, lon, 0)$. While on the surface, the AUV's current waypoint is the closest surface waypoint to the AUV's current location. There exist different methods of defining surface waypoints, such as building a grid map, Voronoi diagram, or analysing common locations for AUV surfacing when underwater navigation fails. A simple method, illustrated in Fig. 8, is to place a surface waypoint above sensor node locations while ensuring a minimum separation distance between them.

4.4.2 Topological edges: navigation between waypoints

Target	Behaviour
$v_{\text{tgt}} \in V^u$	1. The AUV sets its target location to $loc(v_{\text{tgt}})$. If the AUV starts on the surface, it starts to dive to the target depth to ensure that it stays underwater for most of its travel. Underwater travel is generally desired for AUVs, both to avoid surface effects such as waves and to remain clear of surface traffic. However this is not a system requirement and the vehicle could travel along the surface if desired. 2. When the distance to the target waypoint is less than a communications threshold, the AUV starts attempting to contact the nodes $\phi \in v_{\text{tgt}}$. 3. Termination of the navigation action occurs when: a. The AUV successfully contacts all nodes in v_{tgt}. The outcome waypoint is v_{tgt}. b. The AUV fails to contact some of the nodes in v_{tgt} by the time its estimated position is within a small threshold distance of $loc(v_{\text{tgt}})$. The outcome waypoint is the waypoint matching the nodes it successfully contacted. c. The AUV has no contact with any nodes by the time its estimated position is within a small threshold distance of $loc(v_{\text{tgt}})$. The AUV is lost so must surface for a location fix. The outcome waypoint is the closest surface waypoint to its surfacing location.
$v_{\text{tgt}} \in V^s$	1. The AUV sets its target location to $loc(v_{\text{tgt}})$. However, it sets its target depth to a travelling depth to ensure that it stays underwater for most of its travel. 2. When the distance from the current estimated position to the target waypoint is less than a threshold, the AUV sets its target depth to 0 so starts to ascend to the surface. 3. When it reaches the surface, the action terminates and the outcome waypoint is the closest surface waypoint to the surfacing location.

Table 3: Description of navigation behaviour between waypoints. v_{start} and v_{tgt} are the navigation action's start and target waypoints respectively.

The AUV navigates between waypoints using a set of navigation actions, some of which are illustrated in Fig. 8. Navigation actions have a start waypoint v_{start} and a target waypoint v_{tgt}, but may not always reach the target waypoint. Although it is possible to generate a navigation action between all pairs of waypoints, in practice we only generate navigation actions between waypoints that are within a certain distance of each other. Formally:

$$E = \{(v_{\text{start}}, v_{\text{tgt}}) \in V \times V \mid \|loc(v_{\text{start}}), loc(v_{\text{tgt}})\|_2 < d^{nav}_{\max}\}. \tag{9}$$

This set provides a formal definition of the navigation actions available to the AUV, to be incorporated into the navigation model.

AUV behaviour during navigation actions is described in Table 3, and differs based on the targeted waypoint. Navigation actions ensure that the vehicle mostly travels underwater, to avoid wave and wind disturbance and reduce the risk of collision with surface vessels. The table also describes the termination conditions for underwater navigation actions, which are based on the AUV's ability to contact sensor nodes when it estimates that it is close enough to the waypoint. For single-node waypoints, the communications threshold distance is $d^{comm}_{\max}$. For multi-node waypoints, the communications threshold distance is based on the overlap area between the communication ranges of the nodes.

4.5 Navigation TMDP

Given an AUV topological map $\mathcal{T} = \langle V, E \rangle$, the Navigation Timed Markov Decision Process (*Navigation TMDP*) is defined as $\mathcal{M}_{\mathcal{T}} = \langle S_{\mathcal{T}}, \overline{s_{\mathcal{T}}}, A_{\mathcal{T}}, \delta_{\mathcal{T}}, T_{\mathcal{T}}, \Theta_{\mathcal{T}} \rangle$ where:

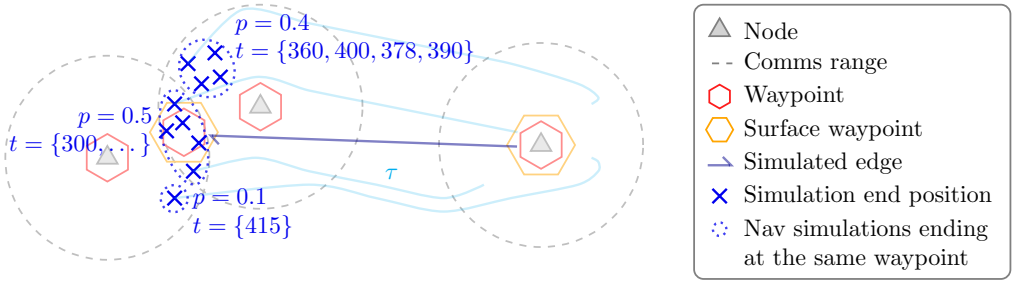


Figure 9: 10 navigation simulations, resulting in three possible outcome waypoints, for an underwater navigation action. The action's target waypoint is underwater, with two associated sensor nodes.

- $S_{\mathcal{T}} = V$, i.e. states are waypoints in the topological map,
- $\overline{s_{\mathcal{T}}} \in V^s$ is the initial state, assumed to be a surface waypoint where the mission commences,
- $A_{\mathcal{T}} = E$ is the set of actions, i.e. navigation actions between waypoints as described in Section 4.4.2,
- The transition function $\delta_{\mathcal{T}}$ and duration function $\Theta_{\mathcal{T}}$ are estimated by sampling action execution attempts using the comms model and navigation simulator as described below.

The navigation simulator, given parameter distributions $Dist(\theta_{nav})$ and $Dist(\theta_{comm})$, is used to estimate the probabilistic outcomes and durations of navigation actions. For each run of the navigation simulator, parameter values are sampled from $Dist(\theta_{nav})$ and $Dist(\theta_{comm})$ and the initial location is sampled from an area around the start waypoint. During the run, the navigation simulator simulates the real system behaviour and action termination conditions as described in Table 3. This includes simulating localisation packet receipt and the final waypoint achieved by the AUV.

For each edge $e = (v_{\text{start}}, v_{\text{tgt}}) \in E$, the navigation simulator is run N_A times to give a dataset $x_{(v_{\text{start}}, v_{\text{tgt}})} = \{(v_i, t_i, \tau_i)\}_{i=0}^{N_A}$ where v_i is the final waypoint achieved by the AUV, t_i is the time taken until termination, and τ_i is the trajectory taken by the simulated AUV. The transition function $\delta_{\mathcal{T}}(v, e, v')$ is calculated by counting the proportion of samples that end in a given waypoint:

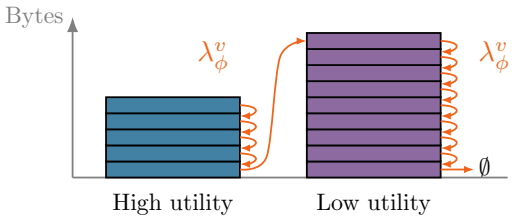
$$\delta_{\mathcal{T}}(v_{\text{start}}, (v_{\text{start}}, v_{\text{tgt}}), v') = \frac{1}{N_A} |\{v_i \mid (v_i, t_i, \tau_i) \in x_{(v_{\text{start}}, v_{\text{tgt}})} \text{ and } v_i = v'\}|. \tag{10}$$

This grouping of navigation simulations by their outcome waypoint is illustrated in Fig. 9 by the three dotted circles, corresponding to three possible outcome waypoints. Five simulations out of $N_A = 10$ end with the AUV communicating with both sensor nodes, so the action is estimated to have a probability $p = 0.5$ of transitioning to the intended target waypoint. Probabilities are similarly shown in the figure for other outcome waypoints. Also shown are the times taken, t_i, for each sample that ends in the corresponding waypoint.

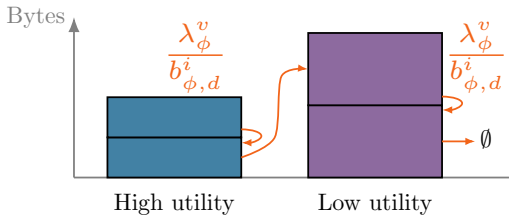
The duration function $\Theta_{\mathcal{T}}$ is estimated by fitting discrete distributions to the times taken for each $(v_{\text{start}}, e, v')$ combination present in the dataset for each edge. We start by separating the times t_i in each dataset $x_{(v_{\text{start}}, v_{\text{tgt}})}$ by their associated outcome waypoint:

$$T^{v'}_{(v_{\text{start}}, v_{\text{tgt}})} = \{t_i \mid (v_i, t_i, \tau_i) \in x_{(v_{\text{start}}, v_{\text{tgt}})} \text{ and } v_i = v'\}. \tag{11}$$

$T^{v'}_{(v_{\text{start}}, v_{\text{tgt}})}$ therefore contains the navigation durations for all samples that ended in waypoint v' after executing action $e = (v_{\text{start}}, v_{\text{tgt}})$ from v_{start}. We then cluster the times in the dataset $T^{v'}_{(v_{\text{start}}, v_{\text{tgt}})}$ into a fixed number of clusters $|\hat{T}|$, to form a cluster set $\hat{T}^{v'}_{(v_{\text{start}}, v_{\text{tgt}})} = \{\hat{t}_1, \ldots, \hat{t}_{|\hat{T}|}\}$. The clustering algorithm we use is Jenks natural breaks optimisation (Jenks, 1967). The duration function can then be defined based on how



(a) Data contents of each node showing individual bytes. The retrieval stochastic process is shown with orange arrows.



(b) Model of node data contents in blocks of bytes. The stochastic retrieval process between blocks has an adjusted rate based on the size of the block.

Figure 10: Illustration of the contents of a node containing two types of data, with differing utility values. After the AUV retrieves all high utility data, it starts retrieving low utility data. After all data is retrieved, the node is empty ($\emptyset$).

many time values fall into each cluster:

$$\theta_{\mathcal{T},v_{\text{start}},e,v'}(\hat{t}) = \begin{cases} \frac{1}{|T^{v'}_{(v_{\text{start}},v_{\text{tgt}})}|}|\{t_i \mid t_i \in T^{v'}_{(v_{\text{start}},v_{\text{tgt}})} \text{ and } t_i \in \hat{t}\}| & \text{if } v' \in x_{(v_{\text{start}},v_{\text{tgt}})} \text{ and } \hat{t} \in \hat{T}^{v'}_{(v_{\text{start}},v_{\text{tgt}})} \\ 0 & \text{otherwise,} \end{cases} \tag{12}$$

where $t_i \in \hat{t}$ means that t_i falls into the cluster $\hat{t}$, i.e. $\hat{t}$ is the element in $\hat{T}^{v'}_{(v_{\text{start}},v_{\text{tgt}})}$ that is closest to t_i. By carrying out clustering, we therefore limit the number of possible durations for each action, aiding scalability given that the number of samples N_A may be large.

In summary, transition probabilities and outcome durations for each navigation action in the navigation TMDP $\mathcal{M}_\mathcal{T}$ are derived using trajectory samples from the navigation simulator.

4.6 Data retrieval model

For each sensor node $\phi \in \Phi$, the AUV may choose to retrieve data from the node while it is at a waypoint where $\phi \in v^u$. The closer the waypoint is to the node, the higher the data transfer rate will be as packet delivery probabilities will likely be higher and acoustic propagation times are lower. Using the comms model, we can enable the AUV to reason about the data retrieval rate from different waypoints. For example in Fig. 8, if the AUV visits the waypoint between the two left-hand nodes it will be able to retrieve data from both nodes without travelling as far as it would by travelling to each node in turn. However, as well as being a harder waypoint to reach navigation-wise, the data transfer rate will be lower for both nodes than it would be if the AUV visited each node individually. If one node contains a large amount of data, it may be more efficient to visit that node's dedicated waypoint and hence move closer to the node. In this section we describe how we generate a per-node CTMDP, which provides a model of the data retrieval process for that node from each waypoint where the node can be contacted.

During the data retrieval process, nodes send packets containing several bytes of data each, with variable timing between packets defined by the network protocol. For high-level decision-making purposes we abstract this into the expected bytes rate over time $\lambda(\cdot)$, as defined in Section 4.3.1. The calculation of this retrieval rate, given packet timing and delivery probablities, is given in Section 3.3.2. We define the data retrieval bytes rate for an AUV at waypoint v retrieving data from node ϕ as λ^v_ϕ. To incorporate differing acoustic packet delivery probabilities across possible comms model parameters, it is calculated by taking the expectation over the comms model parameter distribution:

$$\lambda^v_\phi = \mathbb{E}_{\theta_{comm} \sim Dist(\theta_{comm})} \left[\lambda(\|loc(v), loc(\phi)\|_2, depth(\phi), depth(v), \theta_{comm})\right]. \tag{13}$$

As described in Problem 1, each sensor node contains an amount of data specified by $L(\phi, d)$ for each data type $d \in D$. Recall that D is sorted in order of ascending utility, $U(d_i) \leq U(d_{i+1})$, and is retrieved in order of decreasing utility. As data retrieval is stochastic, it is therefore a stochastic counting process where the number of bytes of data (of decreasing utility) remaining on the node decreases over time. This is illustrated for a simple case in Fig. 10a.

Rather than formalise a CTMDP over individual bytes of data, we discretise the data contents of the node into blocks of bytes $\{b^i_{\phi,d}\}_{i=0}^{|i|_{\phi,d}}$ for each data type d. This is necessary for scalability reasons, as the number of states in the CTMDP would be prohibitively large if we modelled the data contents at the byte level. The smaller the blocks, the more fine-grained the model can reason about the data contents of a node, but the more states the CTMDP will have. In Fig. 10b, we show the data contents of a node discretised into i-indexed blocks of bytes of size $b^i_{\phi,d}$ for data type d.

For notational simplicity and because the number of bytes of data on a node is likely large, we treat the data values in bytes as continuous rather than discrete variables. This allows us to define a CTMDP for each node, where actions correspond to which waypoint the vehicle is at while retrieving the data.

4.6.1 Per-data type CTMDP

For a given node ϕ, the retrieval of data type d, represented as k blocks of data, is modelled as a CTMDP $\mathcal{Q}_{\phi,d} = \langle S_{\phi,d}, \overline{s}_{\phi,d}, A_{\phi,d}, \Delta_{\phi,d}, \mathcal{R}_{\phi,d} \rangle$.

- $S_{\phi,d} = \{s^i_{\phi,d}\}_{i=0}^{k}$ is the set of states, where the CTMDP state is $s^i_{\phi,d}$ if the number of bytes of that data type is within the range covered by that block of data (formally, $L(\phi, d) \in s^i_{\phi,d} \iff \sum_{j=0}^{i} b^j_{\phi,d} < L(\phi, d) \leq \sum_{j=0}^{i+1} b^j_{\phi,d}$) and $s^0_{\phi,d}$ is the state where the node contains no data of type d,
- $\overline{s}_{\phi,d} : S_{\phi,d} \to [0, 1]$ is the initial state distribution, which we define below,
- $A_{\phi,d} = \{a_{\phi,v} \mid v \in V^u \text{ and } \phi \in v\}$ is a set of actions corresponding to carrying out retrieval of data type d from node ϕ while the AUV is at waypoint v,
- $\Delta_{\phi,d} : S_{\phi,d} \times A_{\phi,d} \times S_{\phi,d} \to [0, 1]$ is the transition rate function, which we define below, and
- $\mathcal{R}_{\phi,d} : S_{\phi,d} \times A_{\phi,d} \to \mathbb{R}_{\geq 0}$ is the reward rate function, which we define below.

While the AUV is retrieving data of type d, the reward rate is straightforwardly the expected transfer rate in bytes per second times the utility per byte $U(d)$:

$$\mathcal{R}_{\phi,d}(s^i_{\phi,d}, a_{\phi,v}) = \lambda^v_\phi \cdot U(d). \tag{14}$$

For a fixed retrieval waypoint, the reward rate will be higher earlier in the data retrieval process as the AUV is retrieving higher utility data first.

A transition occurs from $s^i_{\phi,d}$ to $s^{i-1}_{\phi,d}$ when the node is exhausted of $b^i_{\phi,d}$ bytes of data of type d. The transition rate is therefore the data retrieval rate divided by the size of the block of data:

$$\Delta_{\phi,d}(s^i_{\phi,d}, a_{\phi,v}, s^j_{\phi,d}) = \begin{cases} \frac{\lambda^v_\phi}{b^i_{\phi,d}} & \text{if } j = i - 1 \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

These adjusted transition rates between blocks of data are visualized in a simple example in Fig. 10b. In the final state $s^0_{\phi,d}$, no transitions are enabled and the reward rate is zero.

The initial distribution $\overline{s}_{\phi,d}$ is defined by the prior distribution over the data contents of the node, i.e. it depends on the level of knowledge and confidence the AUV has about the data contents of the node before it starts the mission. The initial distribution for each data type d can be calculated by integration over the prior distribution of the number of bytes of data of type d on the node:

$$\overline{s}_{\phi,d}(s^i_{\phi,d}) = \int_{L(\phi,d) \in s^i_{\phi,d}} \mathit{Dist}(L(\phi, d))\, dL(\phi, d). \tag{16}$$

The retrieval process duration and reward are random variables whose behaviour is a combination of the prior distribution over the data contents of the node and the stochastic retrieval process. By building a chain CTMDP, we are constructing a phase-type distribution over these random variables. A phase-type distribution can arbitrarily well approximate any continuous distribution, and methods are available to carry out this fitting (Bolch et al., 2006). We must also consider the model fidelity vs size tradeoff when introducing more blocks of data, as each node in the system must be represented by its own CTMDP.

4.6.2 All data types CTMDP

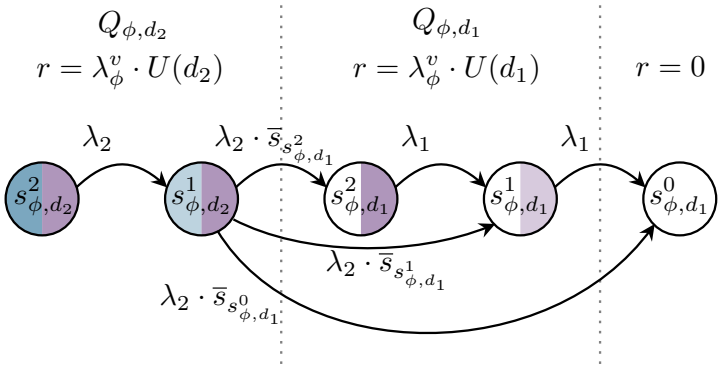


Figure 11: Illustration of the chain CTMDP for the simple case shown in Fig. 10b, where the different data type contents are discretised into two blocks each. For data type d_1 the block size is b^1, and for data type d_2 the block size is b^2. The colour intensity of the LHS of each node shows the remaining high-utility data d_2, and the RHS shows the remaining low-utility data d_1. $\lambda_2 = \lambda^v_\phi / b^2_{\phi,d_2}$ and $\lambda_1 = \lambda^v_\phi / b^1_{\phi,d_1}$.

To form a full joint model of the data retrieval process, one could perform parallel composition for the CTMDPS for each data type $d \in D$. However, this would result in an state space size exponential in the number of data types. However, we can take advantage of the fact that the AUV will always retrieve the highest utility data first, and only move on to lower utility data when the higher utility data is exhausted, to simplify the model.

The all data types CTMDP is a *chain* of per-data type CTMDPs, where the AUV retrieves data in order of decreasing utility. This is illustrated in Fig. 11 for a simple case with two data types, matching the case shown in Fig. 10b. This diagram omits action labels as it only shows retrieval from one waypoint, and omits $\overline{s}_\phi$ as this is defined across all states in the CTMDP chain. If multiple waypoints could be used to contact this node, the choice of waypoint would be an action in the CTMDP and would result in different values of λ^v_ϕ. As shown in the figure, the transition to the empty state $s^0_{\phi,d}$ for each data type that is not the lowest utility data type is replaced with a transition to the CTMDP of the next highest utility data type, weighted according to the initial distribution of for that next data type. The CTMDP for sensor node ϕ is then defined as $\mathcal{Q}_\phi = \langle S_\phi, \overline{s}_\phi, A_\phi, \Delta_\phi, \mathcal{R}_\phi \rangle$ where:

- $S_\phi = (S_{\phi,d_{|D|}} \setminus \{s^0_{\phi,d_{|D|}}\}) \cup (S_{\phi,d_{|D|-1}} \setminus \{s^0_{\phi,d_{|D|-1}}\}) \cdots \cup S_{\phi,d_1}$,
- The initial state distribution $\overline{s}_\phi$ is defined as:

$$\overline{s}_\phi(s) = \begin{cases} \overline{s}_{\phi,d_{|D|}}(s) & \text{if } s \in S_{\phi,d_{|D|}} \\ \left(\prod_{d_i \in \{d_{|D|},\ldots,d_{x+1}\}} \overline{s}_{\phi,d_i}(s^0_{\phi,d_i})\right) \cdot \overline{s}_{\phi,d_x}(s) & \text{if } s \in S_{\phi,d_x} \text{ for some } x \in \{1,\ldots,|D|-1\}, \end{cases} \tag{17}$$

- $A_\phi = \{a_{\phi,v} \mid v \in V^u \text{ and } \phi \in v\}$ is the set of actions corresponding to carrying out retrieval of data from node ϕ while the AUV is at waypoint v, i.e. the same action space as the per-data type CTMDPs,

- The transition rate function Δ_ϕ is defined as:

$$\Delta_\phi(s, a_{\phi,v}, s') = \begin{cases} \Delta_{\phi,d_x}(s, a_{\phi,v}, s') & \text{if } s, s' \in S_{\phi,d_x} \text{ for some } x \in \{1, \dots, |D|\}, \\ \Delta_{\phi,d_x}(s, a_{\phi,v}, s^0_{\phi,d_x}) \cdot \overline{s}_\phi(s^0_{\phi,d_{x-1}}) & \text{if } s \in S_{\phi,d_x} \text{ and } s' \in S_{\phi,d_{x-1}} \\ & \text{for some } x \in \{2, \dots, |D|\}, \\ 0 & \text{otherwise.} \end{cases} \tag{18}$$

The chain CTMDP simplification will introduce some approximation error. Namely, even if the AUV receives a data statistics packet during the mission that informs it of a node's contents of a data type that is *not* the highest utility data type that that node currently contains, the AUV will not be able to act on this information until it has exhausted the higher utility data. This is because the CTMDP is only able to transition to the next highest utility data contents according to $\overline{s}_\phi$, which is based on the data contents prior $Dist(L(\phi, d))$ rather than on updated information in the statistics packet. In practice, this approximation still allows for efficient behaviour since it most accurately reasons about the highest utility (and therefore most valuable) data types present on a node.

Finally, note that this formulation assumes that the data contents of each node is known upon initialisation, i.e. immediately upon starting the mission. In reality, the AUV will only know the data contents of a node when it receives a data statistics packet from the node during the mission. We address this in Section 4.7.2 when combining the navigation TMDP with the data retrieval CTMDPs.

4.7 Mission TMDP construction

We must now combine the discrete navigation TMDP $\mathcal{M}_\mathcal{T}$ with the continuous CTMDPs for data retrieval $\{\mathcal{Q}_\phi\}_{\phi \in \Phi}$, to form a single TMDP that models the AUV's mission. The full process of constructing the mission TMDP, including earlier steps discussed in previous sections, is shown in Fig. 12. To carry out a product composition of these models, we must discretise the CTMDPs into MDPs where one timestep in the MDP corresponds to a timestep in the TMDP. This means that data retrieval actions in the mission TMDP will take one timestep, giving the finest grained control possible over data retrieval durations. These steps are described in this following section. Let the length of one timestep in the navigation TMDP be Δt.

4.7.1 Timestep-conditioned uniformisation

As shown at (f) in Fig. 12, the CTMDP $\mathcal{Q}_\phi$ for each sensor node is converted to an MDP Q^D_ϕ. The standard method of converting a CTMDP to an MDP is to use uniformisation (Baier et al., 2005), which identifies a fixed timestep value for that CTMDP given the fastest transition rate in the CTMDP and then introduces self-loop transitions to match the CTMDP sojourn times and transition probabilities. However, we need a common timestep for all CTMDPs, equal to the timestep of the navigation TMDP, to combine them into a single mission TMDP. We therefore describe our own timestep-conditioned uniformisation process below. This process is based on adjusting the reward values of the MDP to ensure they remain identical in expectation to the CTMDP rewards. Transition probabilities are calculated by standard uniformisation.

Transitions in a CTMDP are a Poisson process, where the time until the next transition is exponentially distributed with rate λ. Let $P_\lambda(t \leq T)$ be the CDF of the first arrival time in the Poisson process. For our chain CTMDP, there are two cases for the transition outcome after one timestep:

i. The data in the current block is not exhausted, and the MDP remains in the same state (self-loop). Therefore data was retreived throughout the timestep, and the reward is $\lambda^v_\phi \cdot U(d) \cdot \Delta t$.
ii. The data in the current block was exhausted, and the MDP transitions to the next state. The reward is $\lambda^v_\phi \cdot U(d) \cdot \mathbb{E}[t_{\text{ex}}]$ where $t_{\text{ex}} < \Delta t$ is the time at which the data was exhausted, which could have been at any point during the timestep.

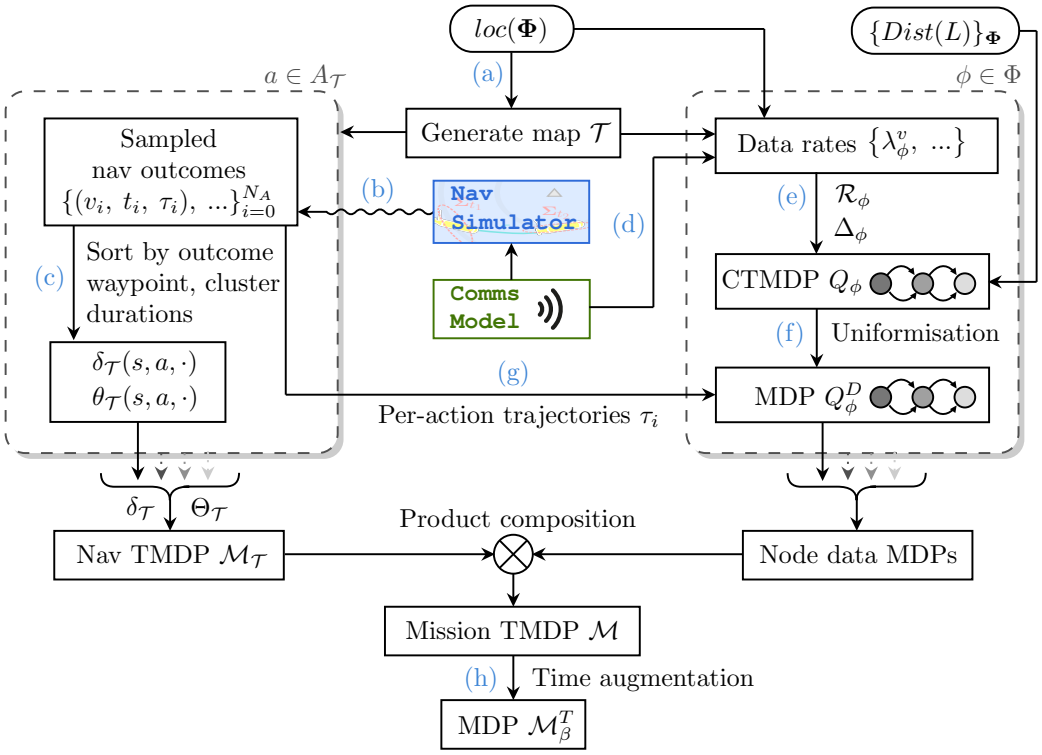


Figure 12: Data flow diagram showing components and steps in the construction of the TMDP model for the AUV mission. Blocks with dotted line edges and shadows represent multiple copies of the same type of component, iterating over the actions (i.e. topological map edges, on LHS) or nodes (RHS). Labels correspond to the sections describing each process: (a): Topological map generation (Sections 4.4.1 and 4.4.2), (b) and (c): Trajectory sampling and clustering (Section 4.5), (d): Calculate data rates, using comms model (Section 4.3.1), (e): Calculate transition and reward rates (Section 4.6.1), (f): Uniformisation with common timestep (Section 4.7.1), (g): Evaluate statistics packet reception probabilities (Section 4.7.2), (h): Time augmentation (Section 4.1).

We can calculate $\mathbb{E}[t_{\text{ex}}]$ in this way: from Bayes' rule, the probability that the first Poisson process arrival took place within T_1 time units given that it took place within $T_2 > T_1$ time units is:

$$P_\lambda(t \le T_1 \mid t \le T_2) = \frac{P_\lambda(t \le T_2 \mid t \le T_1) P_\lambda(t \le T_1)}{P_\lambda(t \le T_2)} \tag{19}$$

$$= \frac{P_\lambda(t \le T_1)}{P_\lambda(t \le T_2)} = \frac{1 - e^{-\lambda \cdot T_1}}{1 - e^{-\lambda \cdot T_2}}. \tag{20}$$

The PDF can then be calculated by differentiating the CDF with respect to T_1:

$$p_\lambda(t = T_1 \mid t \le T_2) = \frac{\lambda \cdot e^{-\lambda \cdot T_1}}{1 - e^{-\lambda \cdot T_2}}. \tag{21}$$

Finally, $\mathbb{E}[t_{\text{ex}}]$ can be calculated:

$$\mathbb{E}[t_{\text{ex}}] = \int_{t'=0}^{\Delta t} t' \cdot p_\lambda(t \le t' \mid t \le \Delta t)\ dt' \tag{22}$$

$$= \frac{1 - e^{-\lambda \cdot \Delta t} \cdot (\lambda \cdot \Delta t + 1)}{1 - e^{-\lambda \cdot \Delta t} \cdot \lambda}. \tag{23}$$

4.7.2 Node data MDP initialisation

The initial state distribution of each node data CTMDP is defined by the prior distribution over the data contents of the node, as described in Section 4.6.2. In reality, the AUV will only know the data contents of a node when it receives a data statistics packet from the node. Statistics packets can be received at any time, during any AUV action such as navigation or data retrieval actions for other nodes. We address this by altering the MDP Q_ϕ^D generated from uniformisation in two ways:

i. Introduce a new, deterministic initial state for each node MDP: the *unknown contents state* $s_\phi^?$. The state space of Q_ϕ^D is then $S_\phi^D = S_\phi \cup \{s_\phi^?\}$.
ii. Extend the action space of the node MDP Q_ϕ^D to include *all actions* in the navigation TMDP and the data retrieval CTMDPs of other nodes: $A_\phi^D = A_\phi \cup A_\mathcal{T} \cup \bigcup_{\phi' \in \Phi \setminus \{\phi\}} A_{\phi'}$. These additional actions have effect only in the initial state, and have zero reward. They exist only to model the dynamics of the AUV receiving a data statistics packet during the course of that action.

Let $p_{\text{stat}}(a)$ be the probability that the AUV receives a data statistics packet during while carrying out action $a \in A_\phi^D \setminus A_\phi$. The transition function for the initial state is then $\Delta_\phi^D(s_\phi^?, a, s') = p_{\text{stat}}(a) \cdot \overline{s}_\phi(s)$, i.e. initialisation to the prior distribution of the node's data contents occurs with probability $p_{\text{stat}}(a)$.

$p_{\text{stat}}(a)$ is calculated given the statistics packet behaviour of the network. For simplicity, we assume here that a statistics packet is transmitted with a fixed probability at each timestep, i.e. a Poisson process with rate λ_{stat}. With this assumption we can calculate the probability of a statistics packet being received during a trajectory τ:

$$p_{\text{stat}}(\tau) = 1 - \prod_{t=0}^{t=|\tau|-1} e^{-\lambda_{\text{stat}} \cdot (t_{n+1} - t_n) \cdot Pr(comm | b^{stat}, \|loc_n, loc(\phi)\|_2, \cdot, \theta_{comm})}. \tag{24}$$

For a navigation action, $p_{\text{stat}}(a \mid \theta_{comm})$ is the mean probability across sampled trajectories for that action, given a specific set of comms model parameters θ_{comm}. For a data retrieval action for another node while the AUV is at waypoint v, $p_{\text{stat}}(a \mid \theta_{comm})$ is calculated based on the distance between the waypoint and node and the length of a data retrieval action. Formally,

$$p_{\text{stat}}(a \mid \theta_{comm}) = \begin{cases} \frac{1}{|\{\tau_i | (v_i, t_i, \tau_i) \in x_{(v_{\text{start}}, v_{\text{tgt}})}\}|} \sum_{\{\tau_i | (v_i, t_i, \tau_i) \in x_{(v_{\text{start}}, v_{\text{tgt}})}\}} p_{\text{stat}}(\tau_i) & \text{if } a = (v_{\text{start}}, v_{\text{tgt}}) \in A_\mathcal{T}, \\ 1 - e^{-\lambda_{\text{stat}} \cdot \Delta t \cdot Pr(comm | b^{stat}, \|loc(v), loc(\phi)\|_2, \cdot, \theta_{comm})} & \text{if } a \in \bigcup_{\phi' \in \Phi \setminus \{\phi\}} A_{\phi'}, \\ 0 & \text{otherwise.} \end{cases} \tag{25}$$

The final value $p_{\text{stat}}(a)$ can then be calculated by taking the expectation of $p_{\text{stat}}(a \mid \theta_{comm})$ over the comms model parameter distribution $Dist(\theta_{comm})$. More complex behaviour, such as statistics messages being shared throughout the network via gossiping between nodes, can also be modelled to derive a different form for $p_{\text{stat}}(a)$ that encapsulates that behaviour.

4.7.3 Full mission TMDP

We can now define the mission TMDP as $\tilde{\mathcal{M}} = \langle S, \overline{s}, A, \delta, T, \Theta, R \rangle$, where:

- $S = V \times_{\phi \in \Phi} S_\phi^D$,
- $\overline{s} = (\overline{s_\mathcal{T}}, s_{\phi_i}^?, \ldots, s_{\phi_{|\Phi|}}^?)$ is the (deterministic) initial state,
- $A = A_\mathcal{T} \cup \bigcup_{\phi' \in \Phi} A_{\phi'}$, i.e. nav actions and data retrieval actions for each node/associated waypoints,
- δ is the transition function, which is the product of the transition functions of the navigation TMDP and the MDPs for each sensor node. State features keep their current value by default unless explicitly specified by the transition function of that component (T)MDP,
- T is as defined in the TMDP definition (Definition 2),

- Θ incorporates the navigation TMDP durations and a fixed duration of 1 for data retrieval actions:

$$\theta_{s,a,s'}(t) = \begin{cases} \theta_{\mathcal{T},v_{\text{start}},e,v'}(t) & \text{if } a = (v_{\text{start}}, v_{\text{tgt}}) \in A_{\mathcal{T}} \text{ and } v' \in s' \\ \mathbb{1}(t=1) & \text{if } a \in A_{\phi'} \text{ for some } \phi' \in \Phi. \end{cases} \tag{26}$$

- R is the reward function, which is defined by the reward rates of the MDPs for each sensor node:

$$R(s,a) = \begin{cases} R_\phi(s_\phi, a_{\phi,v}) & \text{if } a = a_{\phi,v} \in A_\phi \text{ for some } \phi \in \Phi \text{ and } v \in V^u, \\ 0 & \text{otherwise.} \end{cases} \tag{27}$$

We can then convert the mission TMDP to an MDP by time augmentation as described in Section 4.1. This final MDP can be solved using standard techniques. Since we aim to compute a policy offline, we use the PRISM model checker to solve the model (Kwiatkowska et al., 2011), using topological value iteration to exploit the acyclic structure of the MDP obtained from the mission TMDP: as all mission TMDP actions have a duration of at least one timestep, the MDP has no self-loops and can be solved with a single backward pass of value iteration.

4.8 Model extensions

The model described so far is the simplest version of the planning model: sensor nodes have a fixed amount of data, and the acoustic communication environment is static. In this subsection we describe several extensions to the model that can be made to increase its flexibility and applicability to the real world.

4.8.1 Node data generation during mission

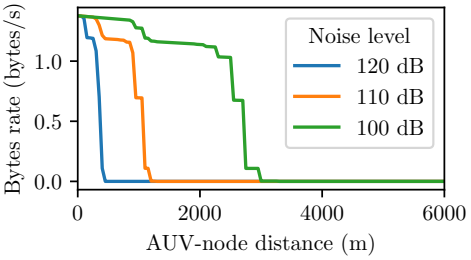
In a setting where the rate of data collection by the sensor nodes is low, during the mission the data contents of a node will only significantly change when data is retrieved by the AUV. However, in some settings sensor nodes may collect data at a sufficiently high rate that their data contents will increase significantly during the course of the mission. We refer to data collection by sensor nodes as data "generation" from the perspective of the AUV, as it occurs independently of the AUV's actions.

We model data generation as a Poisson process with rate $\lambda^{\text{gen}}_{\phi,d}$ for each sensor node ϕ and data type d. Assuming that the generation rate $\lambda^{\text{gen}}_{\phi,d} \ll \lambda^{v}_{\phi}\ \forall\ (\phi, d, v)$, we can model data generation by extending the action space of the node MDP Q^D_ϕ. Similarly to the statistics packet reception process modelling described in Section 4.7.2, we add all actions present in the navigation TMDP and the data retrieval CTMDPs of other nodes to the action space of the node MDP. If the current state of the sensor node MDP is $s^i_{\phi,d}$, the probability of enough data being generated to increment the state by one block of data of type d within time t is:

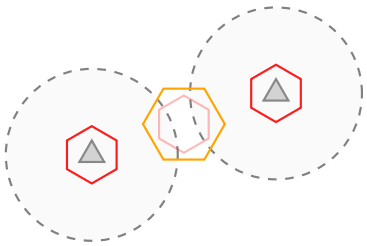
$$\delta(s^i_{\phi,d}, a, s^j_{\phi,d}) = \begin{cases} 1 - e^{-\frac{\lambda^{\text{gen}}_{\phi,d_m}}{b^i_{\phi,d_n}} \cdot \theta(a)} & \text{if } j = i+1 \text{ and } \ m = n, \text{ or } i = |i|_{\phi,d_m} \text{ and } j = 0, n = m+1, \\ 0 & \text{otherwise,} \end{cases} \tag{28}$$

where $\theta(a)$ is the mean duration of action a. $\theta(a) = \Delta t$ for data retrieval actions or is equal to the mean duration of the sampled trajectories for navigation actions.

Implementation of this extension benefits from a hysteresis value for each block of data, acting as a minimum amount of data that must be present in a block before the data status is incremented. This prevents e.g. a block of data being incremented by only a few bytes being generated, which would be interpreted as a full block of data by the AUV. The hysteresis effect is applied only to nodes not in contact with the AUV, so that the AUV can fully empty a block of data if it chooses.



(a) Comms model predicted effective communication range given different levels of environmental acoustic noise. All other model parameters held constant.



(b) An excerpt from Fig. 8, where reduced comms range causes the central, 2-node underwater waypoint to be infeasible.

Figure 13: Illustrations of the justification and approach for planning for varying acoustic performance. The left-hand figure shows how effective communication range can vary with environmental conditions which may be measurable by the AUV, leading to some waypoints potentially being infeasible in the right-hand figure.

4.8.2 Planning for varying acoustic performance

Recall that the comms model has parameters θ_{comm} that define its behaviour, and the planner inputs a distribution over these parameters $Dist(\theta_{comm})$. Different parameter values lead to different levels of acoustic communication performance: the effective communication range, the data transfer rate, and the localisation effectiveness. This is illustrated in Fig. 13a, where different values of the environmental noise parameter result in different maximum ranges for effective data transfer. Equivalently, different distributions over comms model parameters result in different distributions over acoustic communication performance.

Some parameters, in particular the ambient noise level, can be measured by the AUV during the mission. This information can be used to adapt the AUV's behaviour to the current acoustic communication performance. To enable this, the planner inputs a *set* of comms model parameter distributions $\Theta_{comm} = \{Dist(\theta_{comm})_i\}_{i=0}^{|\Theta_{comm}|}$. Policies are then generated that cover each case in Θ_{comm}. During execution, the policy executor uses the onboard comms model (Section 3.3.2) to determine the closest-matching comms model parameter distribution to the current environmental conditions.

To generate comms-dependent policies we introduce a new state variable $Dist(\theta_{comm})_i$ which indexes the set Θ_{comm}. We have no model for evolution of the acoustic environment over time, and more importantly we can assume that AUV actions do not affect the acoustic environment. Therefore $Dist(\theta_{comm})_i$ is set by the mission TMDP initial state distribution, making our $Dist(\theta_{comm})_i$-conditioned policy equivalent to $|\Theta_{comm}|$ separate policies. The transition, reward and duration functions are then conditioned on the value of $Dist(\theta_{comm})_i$. Each component step for TMDP model construction detailed above is repeated for each value $Dist(\theta_{comm})_i \in \Theta_{comm}$. Waypoint autogeneration (Section 4.4.1) is also conditioned on the value of $Dist(\theta_{comm})_i$. Some waypoints may not be feasible with low communiation ranges: in Fig. 8, if the comms range circles were smaller, there may be no overlap between them. The underwater waypoint between the two nodes in the figure would then be infeasible: this is illustrated in Fig. 13b. Actions are only enabled in a state s (and only simulated by the navigation simulator) if the value of $Dist(\theta_{comm})_i$ in s is such that the target waypoint is feasible.

4.8.3 Incentivising mission-level behaviours

As well as data retrieval, there are other mission-level behaviours that may be desirable. The first behaviour we incorporate is to ensure that the AUV finishes its mission at a specified recovery waypoint. This can be modelled by adding a mission termination action to this waypoint in the mission TMDP. As with other TMDP actions, this action is only enabled before the time bound β.

The second mission-level behaviour we incorporate is AUV *mission progress reporting*. When on the surface the AUV can report its mission status via satellite modem. Progress reporting actions are enabled only at surface waypoints and during *reporting windows*, specified as a set of time intervals. A state variable must be added to the mission TMDP state to reflect that a progress report can only be made once during each window.

There are different approaches to incentivising the planner to incorporate these actions into the policy. One principled manner is to specify the reporting and recovery requirements in a specification language such as Linear Temporal Logic (LTL), and carry out probabilistic model checking to trade off the reward of the mission with the probability of satisfying the specification (Lacerda et al., 2019). We do not use this type of method in our experiments due to the increase in solution complexity incurred. Instead we simply modify the mission TMDP reward function by adding large rewards R_{recovery} and R_{report} to the relevant actions.

5 Simulation experiments

For the simulation and field experiments, we consider a setting with two classes of data with different utilities per byte:

- *Measurement* data corresponds to standard logged environmental data, such as temperature or salinity. This data has utility $U(d_m) = 1$ per byte.
- *Event* data corresponds to more interesting and important observations, and therefore has a higher utility $U(d_e) = 5$ per byte. This could be data from a more power-intensive sensor that is only activated when an important environmental event is detected, or data about an infrequent event such as passing marine wildlife or human activity.

Numerical results in this section and Section 6 are based on conservative implementation parameters chosen for the project, for example network parameters in Table 1. By fixing these parameters we can ensure that the results are comparable across experiments. Results therefore do not represent the maximum performance of the system hardware. Byte rates reported are "valuable" bytes i.e. measurement or event data, not including protocol overheads. Additional planner and experiment configuration details are given in the supplementary material.

5.1 Simulator

A dedicated simulator has been developed as a ROS-based platform replicating the backseat system used in the ecoSUB AUV, ensuring consistency with the operational setup. It models key elements such as the vehicle dynamics, tuned specifically for the ecoSUBm5, and environmental factors such as bathymetry. To simulate real-world navigation challenges, the simulator incorporates motion sensor inaccuracies like compass biases, velocity errors, and noise. Additionally, it simulates communication constraints using the acoustic models (discussed in Section 5.1.2), effectively capturing the complexities of underwater data transmission. By integrating these elements, the simulator supports policy generation and also serves as a platform for evaluating the performance of acoustic-aided localisation and the planning system across varied underwater scenarios. When used for evaluation, the simulator's random components are reproducibly seeded in order to ensure that methods are evaluated in exactly the same environment.

5.1.1 Simulated physical environment and planner representation

Simulation experiments take place in a physical area designed to match the field trial layout (Section 6). Five sensor nodes N2–N6 are placed as shown in Fig. 14, with approximately 800 m between N2 and N5. Sensor nodes are at a depth of 10 m and the AUV travels at a depth of 10 m.

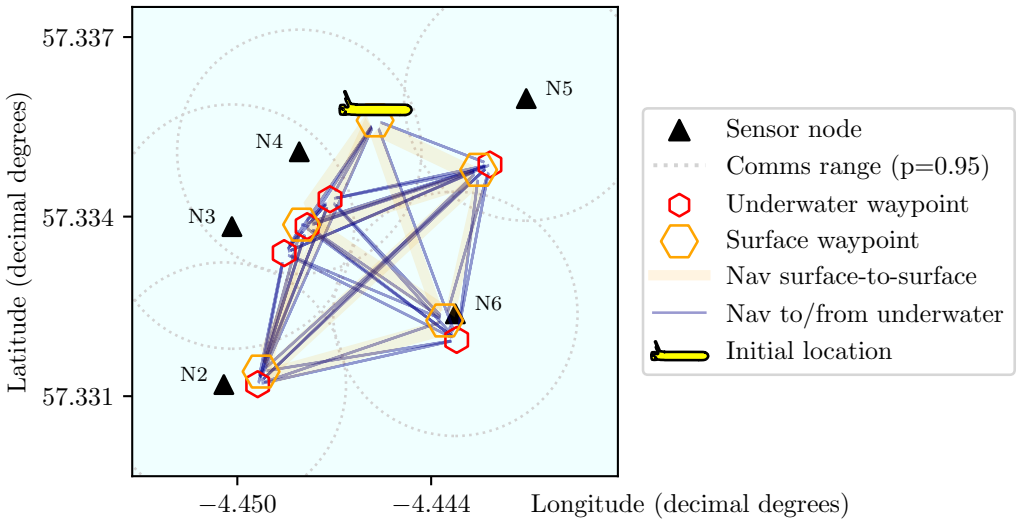


Figure 14: Planner representation of physical layout. This includes problem-defined parameters (sensor node locations and initial location) and planner-defined concepts (waypoints, navigation actions, and comms range estimates). Bathymetry data is used by the evaluation simulator but is not shown here for clarity.

The planner-generated layout includes 5 surface waypoints and 6 underwater waypoints, one of which can communicate with both N3 and N4. As in the field trial, waypoints are moved from their autogenerated positions to place them in deeper water, to ensure the safety of the AUV. 108 navigation actions, shown as arrows in Fig. 14, are generated by the planner.

5.1.2 Evaluation communications simulator

For simulated experiments we require a simulator to model the acoustic channel between the simulated AUV and simulated sensor nodes. Although the planner's acoustic communications model (Section 3.3.2) could be used for this task, this would produce unrealistically optimistic results. As illustrated by our field trial acoustic data (Section 6.1), the real-world acoustic channel will never match the planning model due to unmodelled environmental factors. We therefore define a more complex communication simulator that behaves differently to the planner's model, and should provide a closer representation of the real-world channel. It does this by incorporating the effects of refraction and reflection based on the bottom depth, the bottom acoustic reflectivity as a simple boolean, and a simplified sound speed profile defined as a constant gradient of velocity change.

For each simulated packet transmission, the evaluation communications simulator evaluates whether a path exists between the AUV and the sensor node using the radius of refraction and the bathymetry data depth at the simulated AUV's location. If a path exists, the simulator calculates the level of multipath based on the reflectivity of the bottom and the sound speed profile. If a valid path exists, the SNR is calculated according to Section 3.3.2, given the multipath level, to give the packet success probability. In simulation, localisation and data transfer packets are dropped probabilistically based on this success probability.

Node status update packets may be transmitted at the same time as localisation pings, but with a minimum period between transmissions such that not every localisation packet will be accompanied by a status update.

The acoustic environment parameters for simulated experiments are set to match those of the challenging July trial communication environment (Section 6.1). This is in order to limit the communication ranges of nodes, given the limited spatial scale of the trials layout. In better acoustic conditions, the NMv3 acoustic modems (Section 3.1.2) would have a range exceeding the trials area, leading to all nodes being contactable

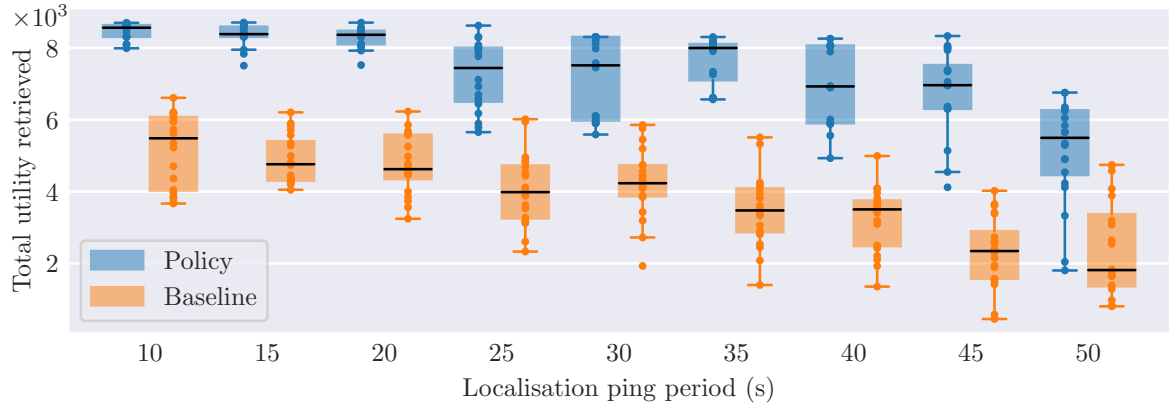


Figure 15: Total utility of data retrieved by the policy-based and baseline methods with an unknown data distribution in the network. 20 random seeds per plot, shown as scatter points with box plots summarizing the distribution.

from the start location.

5.1.3 Baseline comparison method

The baseline comparison method is intended to represent the type of approach typically used for AUV mission planning (Pebody, 2007; Phillips et al., 2023). It is a scripted, rule-based method that does not consider uncertainty in navigation and communication. The baseline finds a shortest touring path between single-node waypoints, and determines the expected duration of each segment based on the AUV's nominal speed. The mission time bound minus the total travel time is then the total allocated time for data retrieval. This data retrieval time is allocated according to the expected total utility of the data on each node according to the prior. If the priors are equal for all nodes, the data retrieval time is split equally between all nodes. The AUV then follows the touring path, stopping at each node for the allocated retrieval time before moving on to the next node. If the current node is fully exhausted of data, and is not currently generating new data, the baseline will move on to the next node earlier than specified in order to minimize wasted time.

Note that the baseline may not successfully spend the intended amount of time at a node, because navigation actions may take longer than expected. The baseline specifies time-of-departure rules (the time at which the AUV should leave the node to reach the next node on time) to prevent the AUV falling further and further behind schedule as the mission progresses. In order to avoid unfairly penalizing the baseline, we keep the same waypoint navigation behaviour as used in our method, including surfacing when lost. This prevents the baseline becoming permanently lost underwater and unable to complete the mission.

5.2 Results

5.2.1 Varying localisation uncertainty

Localisation period is defined as the time between sequential nodes producing localisation packets, so the total time for all nodes to produce one localisation packet is $|\Phi| \times$ localisation period. Greater localisation periods lead to greater uncertainty in the AUV's localisation, leading to more navigation failures and surfacing to re-localise. Additionally, as the AUV must receive at least one localisation ping from a node to reach a waypoint which include that node, successfully reaching a waypoint will likely take longer with longer localisation periods. Finally, status update frequency also decreases with longer localisation period. For the

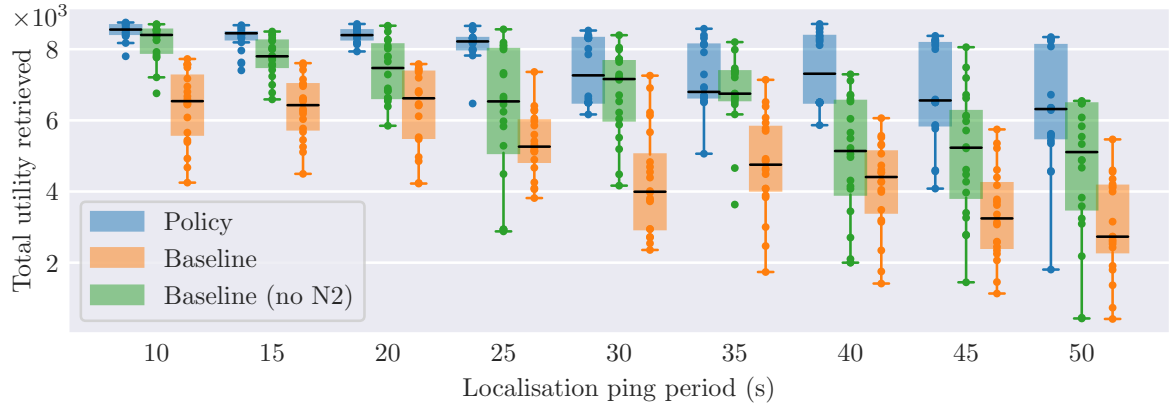


Figure 16: Total utility of data retrieved by the policy-based and baseline methods with an *a priori* known data distribution in the network. 20 random seeds per plot, shown as scatter points with box plots summarizing the distribution.

policy method, this leads to uncertainty about the network's data distribution persisting for a longer time into the mission.

The mission time bound was set to 75 minutes. Simulated spatially varying water currents were drawn from a 2D Gaussian process with a standard deviation of $0.0167\,\mathrm{m\,s^{-1}}$, leading to a 97.5% confidence interval of $0.05\,\mathrm{m\,s^{-1}}$ for east and west current magnitude, and a lengthscale of 1000 m.

Fig. 15 compares the performance of the policy-based method and the baseline method with varying localisation period. For both methods, the total utility of data retrieved decreases with longer localisation periods, and the standard deviation also increases. However, the baseline performance decreases more rapidly than the policy performance, as the baseline method is less able to adapt to the increased uncertainty in localisation. For frequent localisation (localisation period of 10 s), the policy's expected performance is 60% higher than the baseline's, while for localisation periods over 45 s the policy's expected performance is over 100% higher than the baseline's.

Fig. 16 compares the performance of the policy-based method and the baseline when the exact data distribution in the network is known before the mission. Although this is an unrealistic assumption in most cases, this case demonstrates the contribution of the policy's ability to adapting to data knowledge vs its general ability to adapt to uncertain navigation outcomes. Compared to the unknown data distribution in Fig. 15, the policy's performance for a known data distribution is similar or slightly higher. The baseline performs significantly better with a known data distribution, as it can pre-allocate data retrieval time more effectively. *Baseline (no N2)* refers to a variant of the baseline that skips node N2, which does not have enough valuable data to justify the time spent travelling to it. This variant is not possible for an unknown data distribution.

The results again show the increasing performance gap of the policy-based method over the baseline with longer localisation periods, for both baseline variants. In every case the baselines are outperformed by the policy-based method.

As localisation messages become less frequent, the vehicle's position uncertainty grows more quickly while it is underwater. For example, a localisation period of 50 s leads to approximately 2× greater average localisation error than with a localisation period of 10 s in this experiment. As a result, navigation actions take longer to complete and exhibit a higher standard deviation in navigation time. Navigation actions are more likely to result in either the vehicle reaching a different waypoint than the target waypoint, or becoming lost and having to surface for a GPS fix. On average, the vehicle spends approximately 4× as long at the surface

with a localisation period of 50 s than with 10 s, due to the vehicle becoming lost more frequently and having to surface.

As the localisation period increases, the policy-based method retrieves data from fewer nodes on average, prioritising those nodes that it has discovered contain the most valuable data. At the longest localisation periods, most executions of the policy-based method only manage to retrieve data from three nodes. With increasing localisation period, the policy-based method is more likely to fully drain all data from sensor nodes it manages to communicate with, as it will likely not be able to communicate with all sensor nodes. Importantly, the policy-based method also adapts better than the baseline to reaching unexpected waypoints. If the AUV reaches a waypoint that was not the target of the navigation action, it can still take advantage of this and retrieve data from the associated sensor nodes. The baseline cannot do this, as it is focused on communicating with a specific node.

5.2.2 Data generation

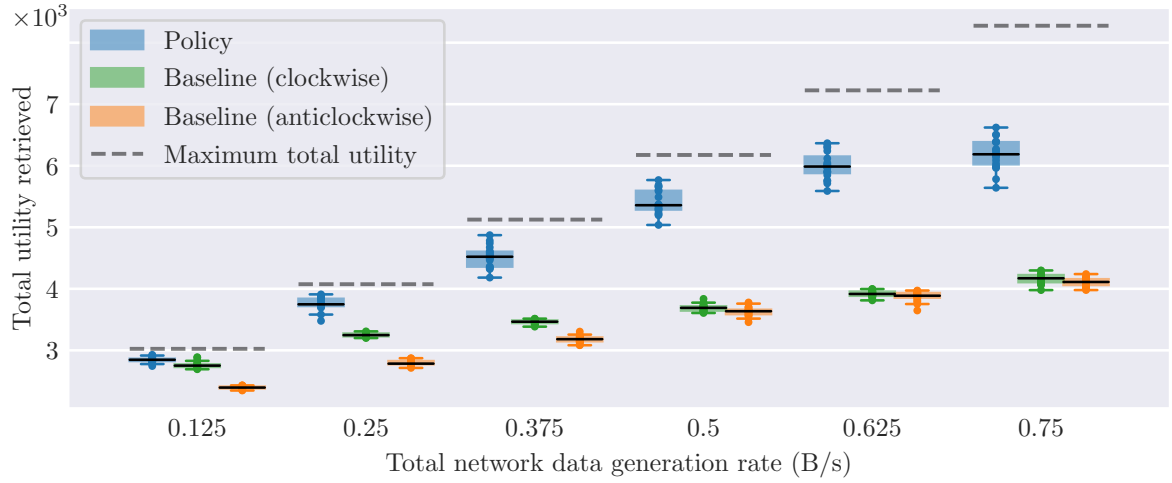


Figure 17: Total utility of data retrieved by the policy-based and two baseline methods in a scenario where data is generated at different rates by each node throughout the mission. The grey dotted line shows the expected maximum utility of data that could be retrieved during the mission, given the data generation rate.

The previous section investigated retrieval of data from nodes with a fixed data distribution. This is an appropriate assumption for networks which generate data sufficiently slowly that the data distribution does not meaningfully change during the mission, for example a use-case where an AUV infrequently visits a network of sensors to retrieve their collected data since the last visit. However, networks with faster data generation rates will likely require more frequent visits, and the data distribution will change during the mission. This experiment evaluates the in-mission data generation model extension (Section 4.8.1) on a scenario with varying rates of data generation by each node.

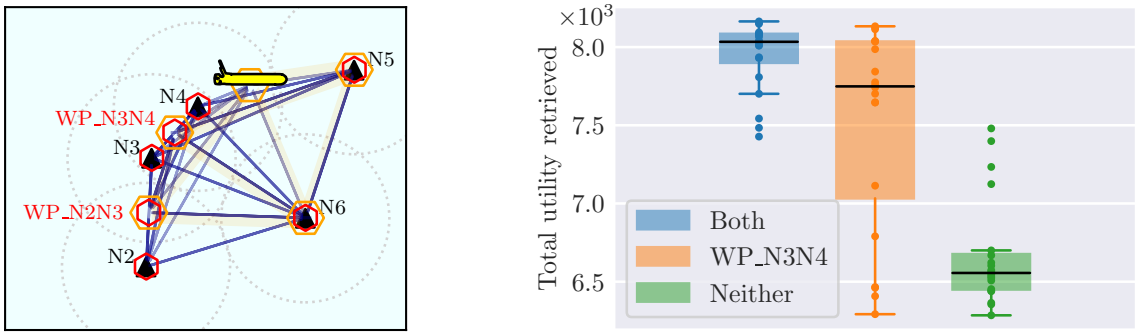
In this scenario, the mission length was set to 2 hours 20 minutes, and the localisation period was set to 20 s. Each sensor node has a maximum capacity of 800 bytes of measurement data, and has 400 bytes contents at the start of the mission. Event data is not used in this experiment. Before the mission the system has a belief over nodes' data generation rate, corresponding to relative generation rates of 1, 5, 4, 0.5 and 2 for nodes N2–N6 respectively. When simulating each experiment, the total network generation rate is split between the nodes according to these relative rates. The believed data generation rate of each node is then multiplied by a Gaussian with mean 1 and standard deviation 0.1 to give the ground-truth data generation rate for that node for that experiment. The on-board network state monitor on the vehicle uses the reported generation rates from nodes to estimate the data contents of nodes in the time between receiving status

update packets.

Fig. 17 shows the total utility of data retrieved by the policy-based method and two baseline methods, with increasing total network data generation rates. Grey dotted lines show the expected total utility of the data contents of the network over the whole mission, i.e. the utility of the initial contents plus the utility of the data generated throughout the mission for all nodes. The line is therefore an upper bound on expected performance. The results show the policy-based method clearly outperforming two baselines with differing direction of travel around the network. With low data generation rates, e.g. $0.125\,\mathrm{B\,s^{-1}}$, the policy-based method only slightly outperforms the baselines and all results have a low variance. This is because, at such low data generation rates, no complex behaviour is required to retrieve most of the data in the network within the time bound. In this case the performance is dominated by the initial data contents of the nodes, and the policy-based method shows similar behaviour to the baselines. It is, however, better able to adapt to communication failures and navigation uncertainties, leading to a reasonable performance improvement over the average performance of the baselines.

At higher data generation rates, the policy-based method increasingly outperforms the baselines by planning frequent sub-tours to nodes generating data quickly, while visiting nodes generating data slowly less frequently. The policy-based method tends to organises its time such that fast data-generating nodes have time to regenerate most of their data contents before the AUV returns to them. Some lower-rate nodes are visited only once or not at all. At around $0.625\,\mathrm{B\,s^{-1}}$ total network generation, the performance of all methods starts to plateau, as data is being generated faster than it can be retrieved by the AUV. With the comms simulator parameters used in this experiment, the AUV is able to retrieve data from nodes at an average rate of around $1.4\,\mathrm{B\,s^{-1}}$.

5.2.3 Multi-node waypoints



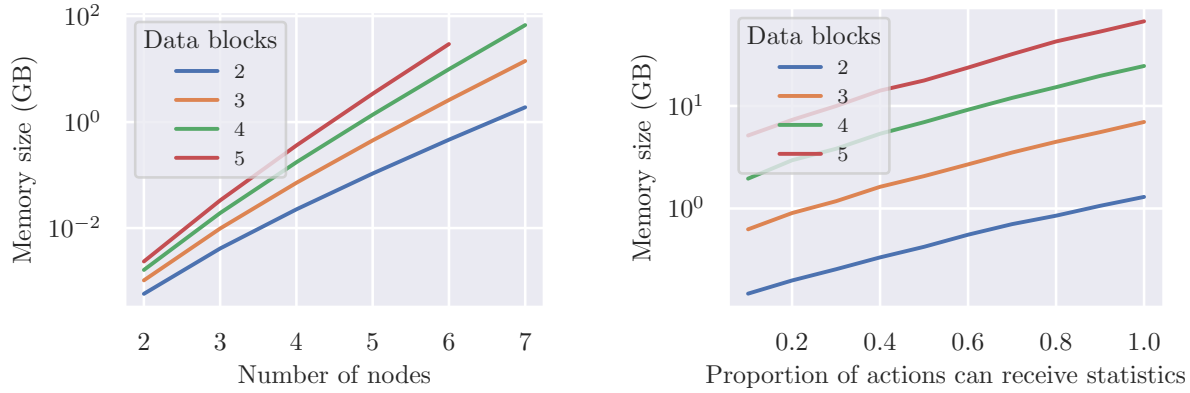
(a) Autogenerated waypoints for field trial node layout. WP_N2N3 and WP_N3N4 waypoints are multi-node waypoints. Legend shown in Fig. 14.

(b) Total utility of data retrieved by the policy-based method with different waypoint configurations. 20 random seeds per plot.

Figure 18: Illustration of the benefit of planning to communicate with multiple sensor nodes.

This experiment investigates the utility of defining waypoints that can communicate with multiple sensor nodes, rather than relying on travelling to and communicating with each node individually. To support this, waypoint autogeneration is applied to the field trial node layout to give the layout in Fig. 18a, with two multi-node waypoints WP_N2N3 and WP_N3N4. To avoid bias due to varying node data contents, the same data distribution is used for all nodes in this experiment: 400 bytes of event data and 400 bytes of measurement data. The data contents of nodes is not known *a priori*, and the localisation ping period is set to 10 s.

Three different policy-based methods are compared in Fig. 18b. One policy was generated with both multi-



(a) Memory utilisation with known data contents.

(b) Memory utilisation with unknown data contents.

Figure 19: Memory utilisation scalability analysis for two cases of the planning method.

node waypoints ("Both"), one only with WP_N3N4 ("WP_N3N4"), and one with no multi-node waypoints ("Neither"). The multi-node waypoint policies outperform the single-node waypoint policy, with the policy using both multi-node waypoints performing best. The large deviation in returns from WP_N3N4 and Neither is due to the fact that these policies must commit to travelling all the way to WP_N2 in order to retrieve data from N2. In some simulated missions, the AUV is able to quickly retrieve data from N3 and N4 so has time to commit to travelling to N2, while in others it is not. The WP_N3N4 policy generally saves time by using waypoint WP_N3N4, so is more likely to be able to commit to travelling to N2.

Policies perform better when given the option to use multi-node waypoints, as it gives them more flexibility in their behaviour. Multi-node waypoints allow the AUV to retrieve data from multiple nodes, but typically at a lower data rate. In many situations this is an acceptable trade-off as it reduces travel time. This is particularly true when nodes do not have enough data to justify the time spent travelling to them individually. Qualitatively, across all simulated experiments, policy-based methods seem to prefer to target multi-node waypoints when they are available.

5.2.4 Planning method scalability

We solve planning models using PRISM's Sparse engine (Kwiatkowska et al., 2011), meaning that the memory requirements of the model are proportional to the number of states in the model and the total number of transition outcomes defined between states. Our planning model is the parallel composition of one mission Timed MDP with $|\Phi|$ sensor node MDPs, meaning that its size scales linearly with TMDP complexity and exponentially with the number of sensor nodes and sensor node model complexity.

Fig. 19 shows the memory utilisation of the planning method for two cases: one with known data contents before the mission (Fig. 19a) and one with unknown data contents (Fig. 19b). Fig. 19a illustrates the exponential growth of model size with increasing number of nodes in the problem, and increasing complexity of the sensor node models. "Data blocks" refers to the number of data blocks used to model the data contents of each node.

Fig. 19b shows the memory utilisation for a problem with unknown data contents. These models therefore incorporate the dynamics of statistics packet reception, as described in Section 4.7.2. The x-axis defines the proportion of actions in the TMDP that have a positive probability of the AUV receiving a statistics packet for a given node, while carrying out that action. With increasing proportion, the model size grows exponentially, as the model must consider receiving any combination of statistics packets from nodes at each

action. With larger number of data blocks, the size of the possible transition outcome set upon receiving a statistics packet grows, leading to a larger model size.

Compared to memory requirements, solution time is generally not a limiting factor due to the timed MDP structure enabling efficient solution (Section 4.7.3). Problems with a memory usage of $\sim 25\,\mathrm{GB}$ can be solved in under 20 minutes (3.4 GHz 40-core CPU, 128 GB RAM), and solution time is proportional to memory usage. Some time is also required to run navigation simulations to build the TMDP. For 100 repeats of 100 navigation actions, the navigation simulator takes around 20 minutes to run. This simulation output is cached for each previously computed combination of physical layout and model parameter belief distributions.

6 Field experiments

The system described in this study was experimented during the HUDSON project trials conducted in July 2021 and November 2021 in Loch Ness. The Loch is 36.2 km long with a maximum width of 2.7 km and a maximum depth of approximately 230 m. The water depth in the working area varied from 5 m to in excess of 200 m. The shore site used was the Old Life Boat station, Temple Pier, Drumnadrochit. This site has been used by the National Oceanography Centre (UK) very successfully in the past due to its proximity to the Loch edge, and in particular the sheltered area of Urquhart Bay. At the trial site in July, the aim was to integrate the main building blocks of the system (see Fig. 1a) and run some pre-trial wet tests to assess the readiness of the system (performance of the underwater communication, controlling the vehicle through back-seat driver, etc.). Following, the goal for the November trial was to deploy the network of sensor nodes in the Loch and run full HUDSON missions with the ecoSUBm5.

6.1 Acoustic environment

The acoustic environment in the July 2021 pre-trials was particularly challenging. An extended period of dry and sunny weather had produced a steep thermocline down to below 40 m. This caused strong downward refraction of the acoustic signals, which when combined with the soft sediment bottom in the bay area, resulted in effective communication ranges being limited to around 600 m with additional shadow zones at closer ranges. An example Sound Velocity Profile (SVP) for July 2021 is shown in Fig. 20, which for a transmitter positioned at 10 m in a depth of 40 m produced similar acoustic paths as shown. With sensor nodes positioned several hundred meters apart, and outside of the effective acoustic range on account of the downward refraction, this resulted in lost communication and navigation packets.
November 2021 trials benefited from considerably more benign channel conditions. The SVP and example raytraces for comparison are shown in Fig. 21. The stark contrast between the acoustic environments is clear from Fig. 20 and Fig. 21.

6.2 Experimental setup

In our experimental scenario in November, the UASN was comprised of 5 nodes, with each one generating event and measurement data as described in Section 5. Fig. 22 shows a side profile diagram of a mooring, and the physical layout of the network is shown in Fig. 23, with the geographical detailed in Table 4.

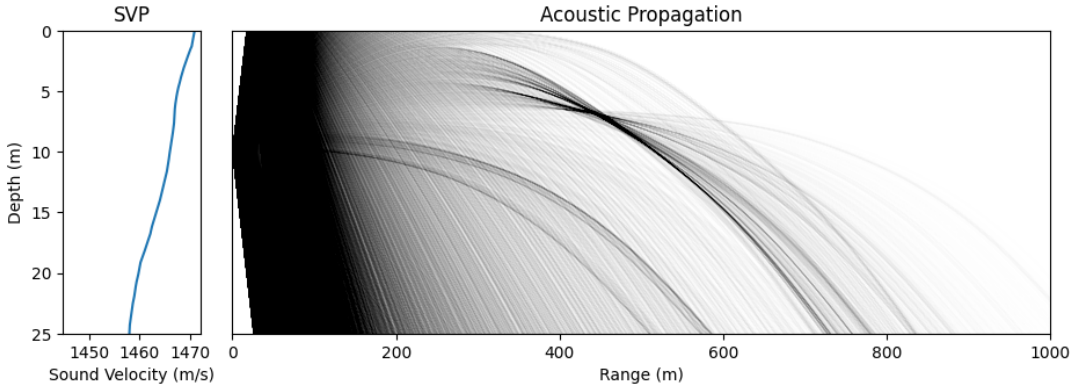


Figure 20: July 2021 Depth vs Sound Velocity. Raytrace showing refraction of acoustic signals. Transmitter set at depth of 10 m, bottom set at depth of 40 m. The steep thermocline led to strong downward refraction of the acoustic signal. The soft silt bed led to minimal bottom reflection resulting in the reduced range and shadow zones.

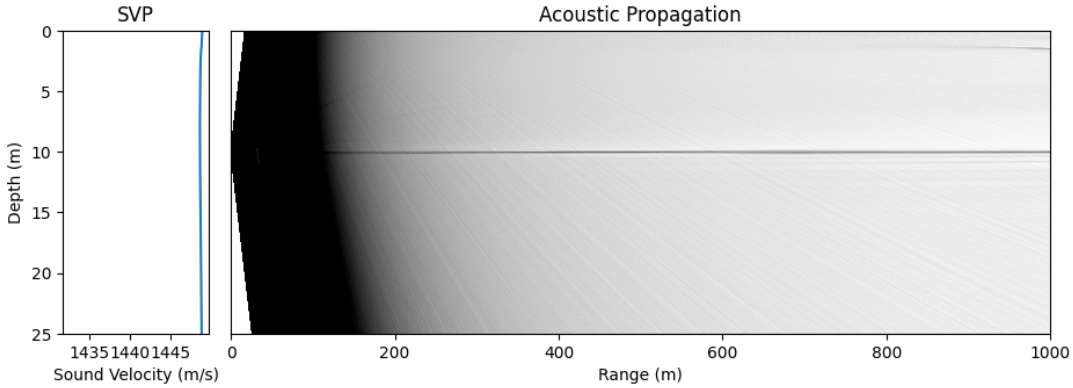


Figure 21: November 2021 Depth vs Sound Velocity. Raytrace showing refraction of acoustic signals. Transmitter set at depth of 10 m, bottom set at depth of 40 m. Here the propagation provides much improved acoustic channels over the conditions seen in July 2021.

Node	Network address	Latitude	Longitude	Water Depth	Modem Depth
N2	4	57.331194°	-4.450417°	31 m	10 m
N3	8	57.333833°	-4.450167°	19 m	10 m
N4	2	57.335083°	-4.448083°	19 m	10 m
N5	5	57.335972°	-4.441056°	33 m	10 m
N6	6	57.332376°	-4.443250°	75 m	10 m

Table 4: Sensor node locations and network information.

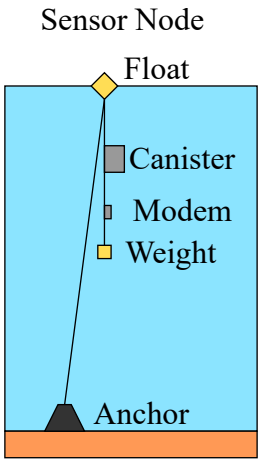


Figure 22: The sensor nodes were moored with the modem hanging below the canister. The modems were also attached to the rope with a weight on the end in order to keep the modem orientation pointed downwards.

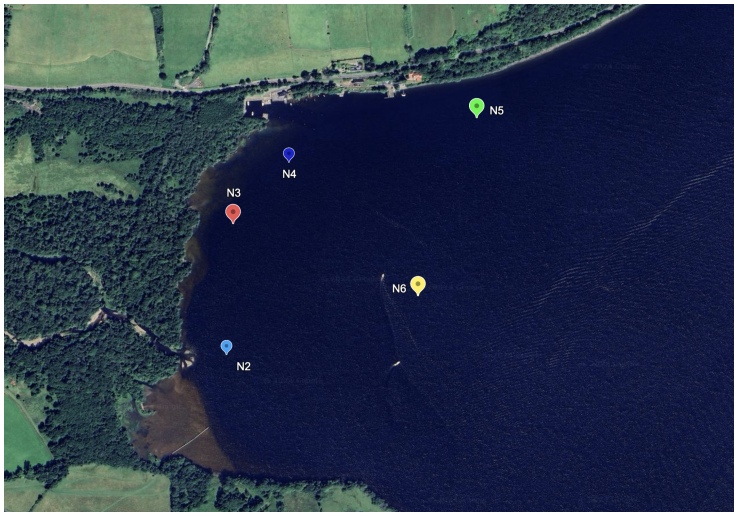


Figure 23: Underwater network layout.

Additional planner and experiment configuration detail is given in the supplementary material.

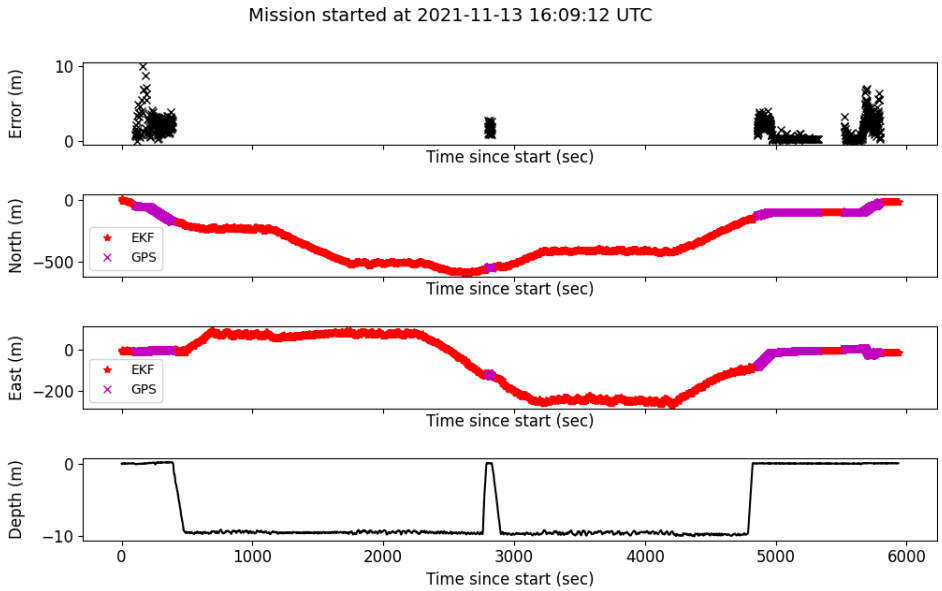


Figure 24: Estimates from the acoustic-aided localisation algorithm versus GPS position (when available) over time during the policy-based mission (13 November). Top: Euclidean norm of the error over time. Centre: estimated position in the North and East directions over time. Bottom: vehicle depth over time.

6.3 Localisation performance

The localisation ping period was set to 3 seconds, meaning the localisation algorithm on the AUV pings the nodes in a round-robin fashion, with a 3-second interval between consecutive pings. However, in practice, this schedule is not strictly adhered to. The process is interrupted when the acoustic channel is prioritised for node communication, which takes precedence over the localisation pinging. The left-hand plot of Fig. 25 shows the AUV's path during a mission conducted in November, where a policy was being executed onboard the vehicle. Due to the lack of ground truth position information, Fig. 24 illustrates the localisation performance during surfacing localisations mid-mission (approximately 3000 seconds from the mission start) and at the mission's conclusion. The error represents the deviation between the onboard localisation estimates and the GPS data obtained at the water's surface. The localisation system provided estimates with an error of less than 5 meters, both mid-mission and at the end, which is comparable to the GPS error.
As can be seen in Fig. 25, localisation error generally increases during data retrieval as fewer localisation pings are received and the AUV does small loops to retrieve data.

6.4 System performance

The full HUDSON system was tested against the baseline (Section 5.1.3) in two data retrieval scenarios. In the unknown data contents scenario (Section 6.4.1), the network data contents were unknown to the AUV *a priori* and must be discovered by receiving status packets from sensor nodes. The data distribution was identical to that of the unknown data contents simulated experiments (Section 5.2.1). The known data contents scenario (Section 6.4.2) had a different data distribution, with the AUV aware of the data contents at each node.

Method	Date	Retrieved			Mean data rate	Figure
		Utility	Event	Measurement	($\mathrm{B\,s^{-1}}$)	
Policy A	13/11	5560	1112 B	0 B	1.1	-
Policy B	13/11	8860	1660 B	560 B	1.4	Fig. 25
Policy C	14/11	9284	1764 B	464 B	1.6	-
Baseline	11/11	5256	952 B	496 B	1.4	Fig. 26

Table 5: Runs summary for the unknown network data contents scenario. The mean data rate in bytes per second is calculated across the time spent attempting to communicate with sensor nodes.

Method	Date	Retrieval Packet Delivery Rate (PDR) per Node				
		N2	N3	N4	N5	N6
Policy A	13/11	N/A	(05/12) 0.42	(18/24) 0.75	(21/32) 0.66	(28/28) 1.00
Policy B	13/11	N/A	(30/36) 0.83	(32/40) 0.80	(28/40) 0.70	(61/64) 0.95
Policy C	14/11	(08/08) 1.00	(30/36) 0.83	(23/28) 0.82	(25/28) 0.89	(61/76) 0.80
Baseline	11/11	(31/36) 0.86	(13/16) 0.81	(13/16) 0.81	(16/24) 0.67	(18/20) 0.90

Table 6: Packet delivery rate summary for the unknown network data contents scenario. For each node a total received packets over total requested packets is calculated to produce a packet delivery rate for retrieved data and event packets.

6.4.1 Unknown network data contents

Table 5 summarizes the runs for the unknown network data contents scenario, totalling 3 policy-based and 1 baseline runs. One illustration of the stochasticity of the real-world environment is the large difference in acoustic data rate experienced between Policy A and Policy B, despite the two missions being run on the same day in immediate succession.

Table 6 breaks down the packet delivery rate for each node for the different runs. This is calculated based on the acoustic packets associated with the data/event retrieval transaction as described in Section 3.3. Comparing the relative PDR values in Baseline 11/11 for each node with the map of the AUV motion in Fig. 26, the high PDR (0.90) for N6 corresponds to a relatively close range as the AUV transited past the node. In contrast, the PDR for node N5 (0.67) aligns with the greater range from the AUV during the data retrieval operation.

Fig. 25 shows one of the policy-based mission runs, scoring 8860 utility points. The AUV took a tour of 4 nodes of the network in a clockwise direction, starting with node N5. After visiting N6, it surfaced to perform a mission progress report then continued to a location where it could communicate with both N3 and N4. In the policy execution trace in the top right of the figure, it can be seen that the AUV switches between retrieving data from N3 and N4: it fully retrieves all valuable event data from both before starting to retrieve measurement data from both in the remaining mission time before it must return to the recovery location. As localisation was run at a high frequency (localisation period 3s), the AUV has low localisation uncertainty at all times and does not fail any navigation actions.

Policy A acted qualitatively similar to Policy B, but encountered a much more difficult acoustic environment, leading to a 30% lower data rate. Policy C encountered favourable water current and acoustic conditions (with a high byte rate of 1.6 bytes per second) and was able to visit all nodes in a single clockwise tour, starting with N5. It therefore scored the highest of all runs in the unknown network data contents scenario.

The baseline run for the unknown network data contents scenario is shown in Fig. 26. The AUV visited all nodes in a single anticlockwise tour, starting with N4. The baseline scored lowest of all runs, close only to Policy A which had a 30% lower data rate. The baseline run did experience an unexplained and uncommanded surfacing event mid-mission, visible in the depth plot and by the purple GPS fix markers near

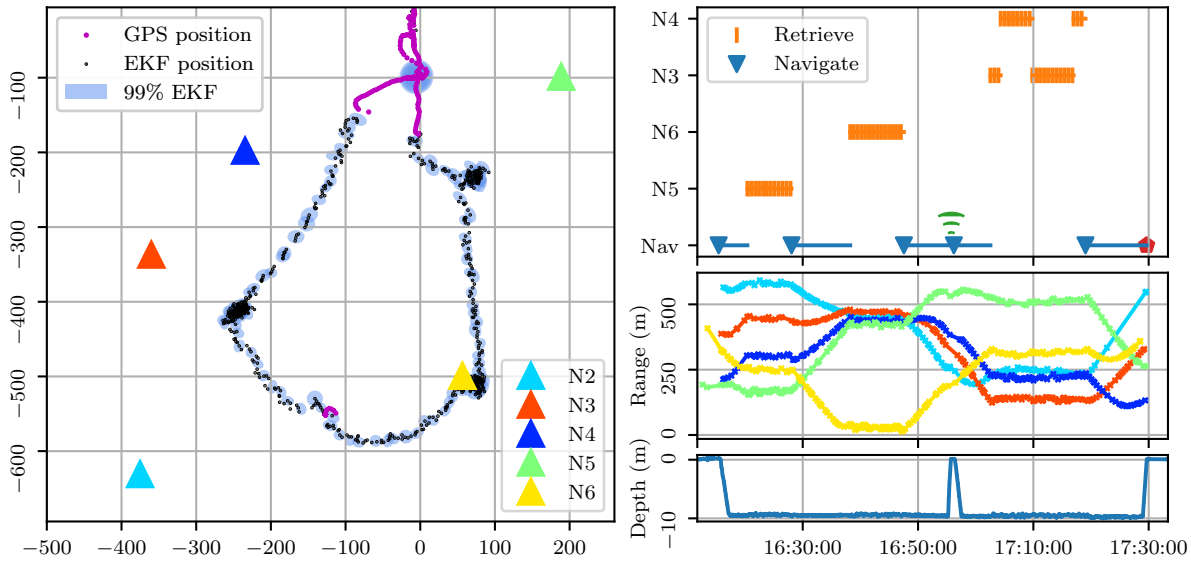


Figure 25: A policy-based mission (13/11) with the unknown network data contents scenario. LHS: physical layout and localisation estimates throughout the mission, with downsampled estimated positions and 99% EKF confidence ellipses. RHS top: log of executed actions, with line segments showing action durations. The green signal marker is a mission progress report (Section 4.8.3). RHS middle: localisation packets received, one marker per packet receipt. RHS bottom: vehicle depth, with mid-mission surfacing for a progress report.

N2 in the navigation plot. However, this surfacing event took place while the vehicle was retrieving data from N2, and execution data and the average byte rate in Table 5 show that the vehicle was able to continue communicating with the node while surfacing and re-diving, such that the event did not meaningfully impact the data rate or utility score.

6.4.2 Known network data contents

Method	Date	Retrieved			Mean data rate	Figure
		Utility	Event	Measurement	($\mathrm{B\,s^{-1}}$)	
Policy	13/11	7140	1136 B	1460 B	1.4	Fig. 27
Baseline	10/11	7828	1220 B	1728 B	1.3	Fig. 28

Table 7: Runs summary for the known network data contents scenario. The mean data rate in bytes per second is calculated across the time spent attempting to communicate with sensor nodes.

Table 7 summarizes the runs for the known network data contents scenario, totalling 1 policy-based and 1 baseline run. With this distribution of known data contents, and high-frequency localisation, the policy's behaviour in this scenario was always to visit all nodes in a single tour, meaning that multiple policy runs would not differ in their qualitative behaviour and would only differ in score due to the stochasticity of the environment.

Table **??** breaks down the packet delivery rate for each node for the different runs. Comparing the PDR for the Baseline (10/11) run with the motion of the AUV shown in Fig. 28, again it is clear to see the difference in PDR for N5 (0.82) and N6 (0.91) and the relative range between AUV and each node during the data retrieval process.

Fig. 27 shows the policy-based run for the known network data contents scenario. Both methods successfully

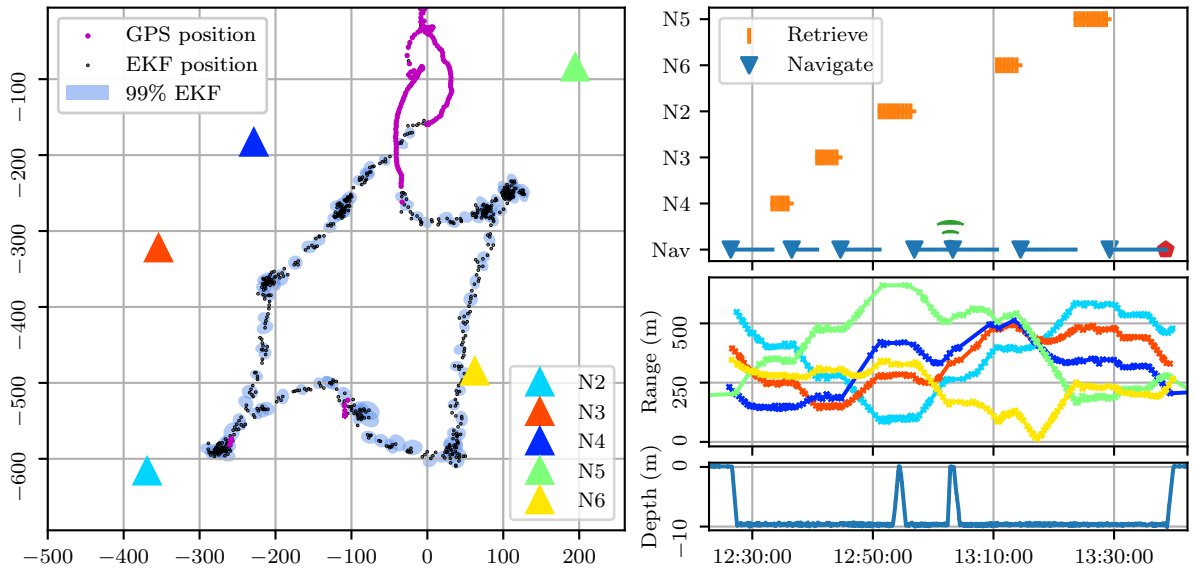


Figure 26: A baseline mission (11/11) with the unknown network data contents scenario. A description of the plots is provided in the caption of Fig. 25. This baseline run experienced an uncommanded surfacing event, which is visible in the bottom right plot alongside the intended surfacing for the mission progress report. The surfacing location can be seen in purple near N2 in the navigation plot.

Method	Date	Retrieval Packet Delivery Rate (PDR) per Node				
		N2	N3	N4	N5	N6
Policy	13/11	(10/12) 0.83	(32/36) 0.89	(24/28) 0.86	(45/52) 0.87	(52/60) 0.87
Baseline	10/11	(33/36) 0.92	(41/44) 0.93	(31/32) 0.97	(46/56) 0.82	(40/44) 0.91

Table 8: Packet delivery rate summary for the known network data contents scenario. For each node a total received packets over total requested packets is calculated to produce a packet delivery rate for retrieved data and event packets.

retrieved all event data from the network, meaning that the policy spent an adequate amount of time at each node to maximize the valuable data retrieved. Other than going anticlockwise instead of clockwise, the policy-based run behaved qualitatively similar to the baseline run shown in Fig. 28 in that it visited all nodes in a single tour. Despite carrying out the same actions in reverse order to the baseline, the policy-based run spent 10% longer time navigating due to environmental conditions. This is shown by the longer physical path length in Fig. 27 compared to Fig. 28. Overall, the baseline method spent 25% more time (2263 s vs 1804 s) communicating with nodes so was able to retrieve more data. We therefore class the planner-level performance of the policy-based method as comparable to the baseline method in this scenario.

7 Method Assumptions and Limitations

As discussed in Section 3.4, our method relies on the following assumptions. Firstly, we assume that the locations of sensor nodes are known and static. This is not a strong assumption, as in practice sensor nodes are most often deployed in fixed locations and their precise positions do not need to be known with high accuracy. In our field trial, node deployment positions were recorded with standard GPS accuracy, and the mooring lines (Fig. 22) would have allowed for some movement of the nodes. Despite this, the localisation error was $\leq 10\,\text{m}$ in practice (Fig. 24), which is small relative to the communication range of the sensor

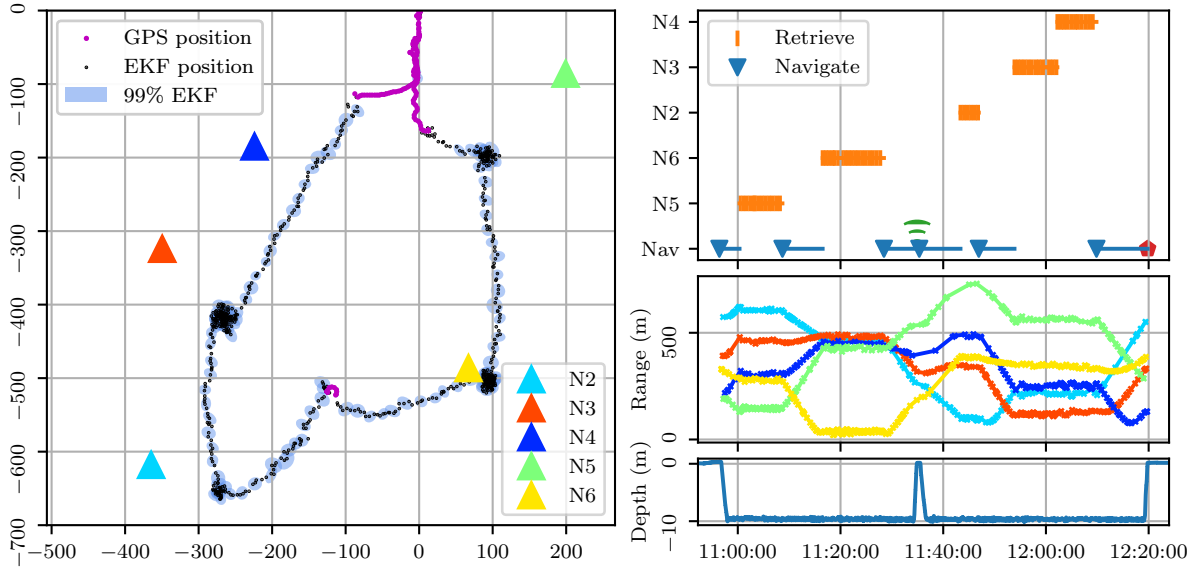


Figure 27: A policy-based mission (13/11) with the known network data contents scenario. A description of the plots is provided in the caption of Fig. 25. With a known and roughly even data distribution, the policy determined visiting every node in a single tour, similar to a baseline run, was the optimal strategy.

nodes. Moreover, methods exist to automatically determine acoustic beacon locations using only range measurements and no prior information (Olson et al., 2007; Blanco et al., 2008). The problem formulation also assumes that the AUV's initial position is known, but this assumption can easily be relaxed by defining an initial state distribution over the AUV's initial waypoint in the navigation model.

Secondly, we assume access to belief distributions over environment, acoustic channel, data contents, and vehicle dynamics parameters. At the start of system deployment, these belief distributions can be initialised as uninformative distributions to reflect the lack of prior knowledge. While inaccurate distributions may reduce performance, more accurate distributions can be learned by analysing data from completed missions. Both the field trial and simulation results were achieved with a uniform belief distribution over water current strength, representing little prior knowledge other than the maximum expected current strength. Maintaining belief distributions over unknown quantities is a standard approach in probabilistic robotics to address uncertainty and partial observability (Thrun et al., 2005).

Thirdly, in order to plan for AUV navigation, the method requires a reasonably accurate model of the vehicle's navigation behaviour. While we maintain belief distributions over the vehicle's dynamics coefficients, we assume that the *structure* of the dynamics model, including the control system behaviour, is known. This is a reasonable assumption, as we would expect the user of the system to have access to their vehicle's design parameters, such as its control system parameters and dynamics coefficients. In our field trial, the AUV's navigation model was comparitively simple and was not extensively tuned to the specific vehicle used. A similar requirement applies for the acoustic communication model to reasonably accurately capture the communication behaviour of the acoustic modems. Similarly, we assume that the user has access to the acoustic modem's design parameters, and we achieve good results in simulation and in the field trial with a relatively simple acoustic model.

Finally, we do not model sensor node failure due to e.g. hardware failure or battery depletion, and we assume that sensor nodes always have their acoustic modems powered in receive mode. The planning method effectively assumes that given enough attempts, the AUV will always be able to communicate with a sensor node if it is within range. The planning model could be extended to account for sensor node

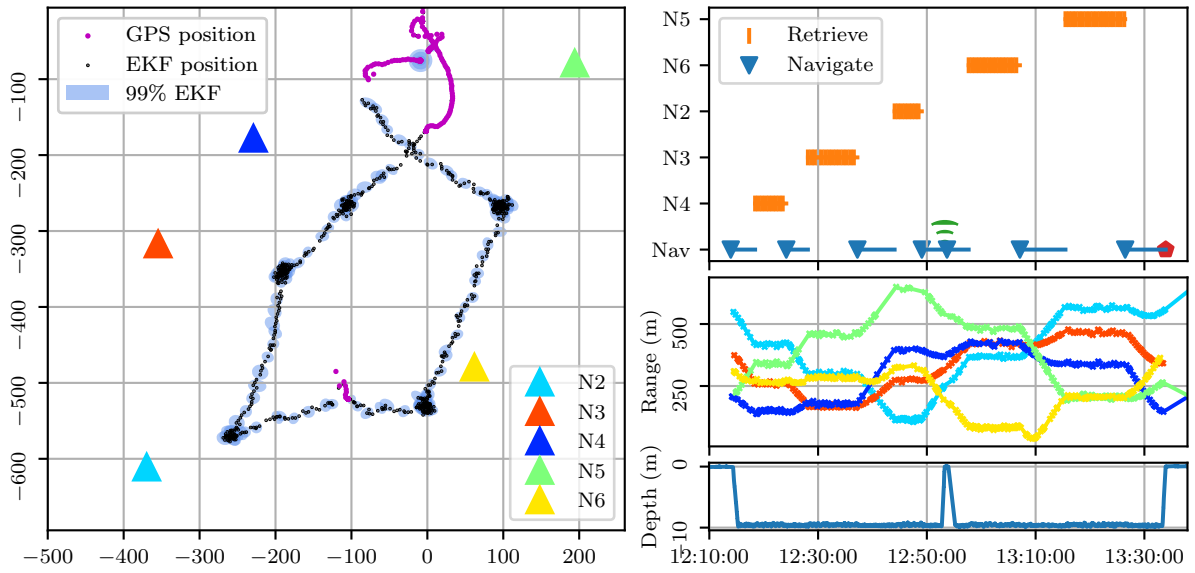


Figure 28: A baseline mission (10/11) with the known network data contents scenario. A description of the plots is provided in the caption of Fig. 25.

failure by introducing an additional state in the sensor node CTMDPs to represent the node having failed. Note that node failure may reduce the mission data retrieval performance, but does not prevent the AUV from retrieving data from other nodes and returning to the mission end position. To reduce node power consumption, future work could explore duty cycling and scheduling of modem power-up between sensor nodes and the vehicle.

8 Conclusion and future work

Underwater acoustic and wireless sensor networks offer transformative capabilities for spatio-temporal ocean data collection, supporting applications such as climate monitoring, marine life analysis, and industrial diagnostics. However, retrieving data from submerged sensor nodes in near real-time remains a significant challenge due to environmental constraints and the high costs of traditional methods like research vessel deployments.

This work introduces an integrated adaptive framework that leverages modelling approaches and planning under uncertainty to enable an Autonomous Underwater Vehicle (AUV) to function as a mobile sink for sparse underwater sensor networks. By dynamically prioritizing nodes based on data value and utilising acoustic communication for simultaneous data harvesting and time-of-flight localisation, the framework addresses critical operational challenges. Field trials validated the framework's performance, demonstrating substantial improvements over conventional hand-designed behaviours for underwater data retrieval. These findings underscore the transformative potential of AUV autonomy in advancing sustainable ocean monitoring. With integration into long-range AUV platforms, like those from the Autosub Long Range (ALR) family of vehicles (Roper et al., 2017; Roper et al., 2021), this approach paves the way for scalable, cost-effective solutions to the growing demands of ocean science, unlocking new possibilities for understanding and managing our oceans.

For the planning method, future work includes improving the scalability of the method to larger numbers of nodes and longer missions. Methods to improve scalability include hierarchical planning, heuristic search

or learning-based methods. Additionally, the system's performance would likely be improved by extending the MDP formulation to a robust MDP formulation (Nilim and El Ghaoui, 2005), which would allow the planner to better consider the impacts of environment uncertainty.

Further work could explore formulating the system as a decision-making process spanning multiple missions, rather than solving each mission individually as in this work. This would also require exploring effective methods for updating planner belief distributions and models based on the data and experience gathered during each mission.

Acknowledgements

The Harvesting of Underwater Data from SensOr Networks (HUDSON) project was funded by the EPSRC ORCA Hub Partnership Resource Fund (EP/R026173/1). The initial development of the ecoSUB AUVs was funded by the Innovate UK project 'Launch & Recovery of Multiple AUVs from an USV' (Project Number: 102302). Development of the ad-hoc acoustic network was funded by the Innovate UK project 'LCAT: Low Cost AUV Technology: development of smart networks and AI based navigation for dynamic underwater environments' (Project Number: 104058). Development of the V3 Nano modem was funded by the EPSRC project 'USMART - smart dust for large scale underwater wireless sensing' (EP/P017975/1). Other work which contributed to this experiment has been funded by the UK's Natural Environment Research Council's Oceanids capital programme and the EPSRC "From Sensing to Collaboration" programme grant (EP/V000748/1). Matthew Budd is supported by an Amazon Web Services Lighthouse scholarship.

The authors would like to express their gratitude to all those involved in undertaking the field trials, including, but not limited to, James Burris, Ed Chaney, Len McLean, and Gordon Menzies. Additionally, the authors would like to thank Jeremy Sitbon for his invaluable assistance in preparing the ecoSUB vehicle for the field trials and Mauro Dragone for his contributions to the HUDSON project, including his support during the trials and his work on developing the data polling mechanism.

Conflict of interest

The authors declare that they have no conflict of interest.

References

Akyildiz, I. F., Pompili, D., and Melodia, T. (2004). Challenges for efficient communication in underwater acoustic sensor networks. *ACM Sigbed Review*, 1(2):3–8.

Awan, K. M., Shah, P. A., Iqbal, K., Gillani, S., Ahmad, W., and Nam, Y. (2019). Underwater wireless sensor networks: A review of recent issues and challenges. *Wireless Communications and Mobile Computing*, 2019(1):6470359.

Baier, C., Hermanns, H., Katoen, J.-P., and Haverkort, B. R. (2005). Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. *Theoretical Computer Science*, 345(1):2–26.

Blanc, S. (2020). Event-driven data gathering in pure asynchronous multi-hop underwater acoustic sensor networks. *Sensors*, 20(5):1407.

Blanco, J.-L., Gonzalez, J., and Fernández-Madrigal, J.-A. (2008). A pure probabilistic approach to range-only SLAM. In *2008 IEEE international conference on robotics and automation (ICRA)*, pages 1436–1441. IEEE.

Bolch, G., Greiner, S., De Meer, H., and Trivedi, K. S. (2006). *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons.

Budd, M., Duckworth, P., Hawes, N., and Lacerda, B. (2023). Bayesian reinforcement learning for single-episode missions in partially unknown environments. In *Conference on Robot Learning*, pages 1189–1198. PMLR.

Budd, M., Salavasidis, G., Karnarudzaman, I., Harris, C. A., Phillips, A. B., Duckworth, P., Hawes, N., and Lacerda, B. (2022). Probabilistic planning for AUV data harvesting from smart underwater sensor networks. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12051–12057. IEEE.

Cashmore, M., Fox, M., Larkworthy, T., Long, D., and Magazzeni, D. (2014). AUV mission control via temporal planning. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 6535–6541. IEEE.

Cheng, M., Guan, Q., Ji, F., Huang, Y., and Quek, T. Q. (2023). Mobile relaying between USV and AUV under fer constraints for underwater data transmission. *IEEE Communications Letters*.

Chou, C. T., Rana, R., and Hu, W. (2009). Energy efficient information collection in wireless sensor networks using adaptive compressive sensing. In *IEEE Conference on Local Computer Networks*.

Cristini, L., Lampitt, R. S., Cardin, V., Delory, E., Haugan, P., O'Neill, N., Petihakis, G., and Ruhl, H. A. (2016). Cost and value of multidisciplinary fixed-point ocean observatories. *Marine policy*, 71:138–146.

Duan, R., Du, J., Jiang, C., and Ren, Y. (2020). Value-based hierarchical information collection for AUV-enabled internet of underwater things. *IEEE Internet of Things Journal*, 7(10):9870–9883.

Duff, M. O. (2003). *Optimal learning: Computational procedures for Bayes-adaptive Markov decision processes.* PhD thesis, Dept. of Computer Science, University of Massachusetts Amherst.

Eickstedt, D. P. and Sideleau, S. R. (2009). The backseat control architecture for autonomous robotic vehicles: A case study with the Iver2 AUV. In *OCEANS 2009*, pages 1–8. IEEE.

Favaro, F., Casari, P., Guerra, F., and Zorzi, M. (2012). Data upload from a static underwater network to an AUV: Polling or random access? In *2012 Oceans-Yeosu*, pages 1–6. IEEE.

Felemban, E., Shaikh, F. K., Qureshi, U. M., Sheikh, A. A., and Qaisar, S. B. (2015). Underwater sensor network applications: A comprehensive survey. *International Journal of Distributed Sensor Networks*, 11(11):896832.

Fenucci, D., Sitbon, J., Neasham, J., Phillips, A., and Munafò, A. (2022). Ad hoc acoustic network aided localization for micro-AUVs. *Field Robotics*, 2(1):1888–1919.

Gong, J., Chang, T.-H., Shen, C., and Chen, X. (2018). Flight time minimization of UAV for data collection over wireless sensor networks. *IEEE Journal on Selected Areas in Communications*, 36(9):1942–1954.

González-García, J., Gómez-Espinosa, A., Cuan-Urquizo, E., García-Valdovinos, L. G., Salgado-Jiménez, T., and Escobedo Cabello, J. A. (2020). Autonomous underwater vehicles: Localization, navigation, and communication for collaborative missions. *Applied sciences*, 10(4):1256.

Heidemann, J., Ye, W., Wills, J., Syed, A., and Li, Y. (2006). Research challenges and applications for underwater sensor networking. In *IEEE Wireless Communications and Networking Conference, 2006. WCNC 2006.*, volume 1, pages 228–235. IEEE.

Henson, S. A., Beaulieu, C., and Lampitt, R. (2016). Observing climate change trends in ocean biogeochemistry: when and where. *Global change biology*, 22(4):1561–1571.

Hollinger, G. A., Choudhary, S., Qarabaqi, P., Murphy, C., Mitra, U., Sukhatme, G. S., Stojanovic, M., Singh, H., and Hover, F. (2012). Underwater data collection using robotic sensor networks. *IEEE Journal on Selected Areas in Communications*, 30(5):899–911.

Hollinger, G. A., Pereira, A. A., Binney, J., Somers, T., and Sukhatme, G. S. (2016). Learning uncertainty in ocean current predictions for safe and reliable navigation of underwater vehicles. *Journal of Field Robotics*, 33(1):47–66.

Howard, R. A. (1960). *Dynamic programming and Markov processes.* John Wiley.

Jahanbakht, M., Xiang, W., Hanzo, L., and Azghadi, M. R. (2021). Internet of underwater things and big marine data analytics—a comprehensive survey. *IEEE Communications Surveys & Tutorials*, 23(2):904–956.

Jenks, G. F. (1967). The data model concept in statistical mapping. *International yearbook of cartography*, 7:186–190.

Jones, D. O., Gates, A. R., Huvenne, V. A., Phillips, A. B., and Bett, B. J. (2019). Autonomous marine environmental monitoring: Application in decommissioned oil fields. *Science of the total environment*, 668:835–853.

Kilfoyle, D. B. and Baggeroer, A. B. (2000). The state of the art in underwater acoustic telemetry. *IEEE Journal of oceanic engineering*, 25(1):4–27.

Kurniawati, H. and Yadav, V. (2016). An online POMDP solver for uncertainty planning in dynamic environment. In *Robotics Research*, pages 611–629. Springer.

Kwiatkowska, M., Norman, G., and Parker, D. (2011). PRISM 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings 23*, pages 585–591. Springer.

Lacerda, B., Faruq, F., Parker, D., and Hawes, N. (2019). Probabilistic planning with formal performance guarantees for mobile service robots. *The International Journal of Robotics Research*, 38(9):1098–1123.

Li, N., Martínez, J.-F., Meneses Chaus, J. M., and Eckert, M. (2016). A survey on underwater acoustic sensor network routing protocols. *Sensors*, 16(3):414.

Lv, Z., Zhang, J., Jin, J., Li, Q., and Gao, B. (2018). Energy consumption research of mobile data collection protocol for underwater nodes using an USV. *Sensors*, 18(4):1211.

McCarthy, G. D., Brown, P. J., Flagg, C. N., Goni, G., Houpert, L., Hughes, C. W., Hummels, R., Inall, M., Jochumsen, K., Larsen, K., et al. (2020). Sustainable observations of the AMOC: Methodology and technology. *Reviews of Geophysics*, 58(1):e2019RG000654.

Mohapatra, A. K., Gautam, N., and Gibson, R. L. (2012). Combined routing and node replacement in energy-efficient underwater sensor networks for seismic monitoring. *IEEE Journal of Oceanic Engineering*, 38(1):80–90.

Mohsan, S. A. H., Li, Y., Sadiq, M., Liang, J., and Khan, M. A. (2023). Recent advances, future trends, applications and challenges of internet of underwater things (iout): A comprehensive review. *Journal of Marine Science and Engineering*, 11(1):124.

Morozs, N., Sherlock, B., Henson, B. T., Neasham, J. A., Mitchell, P. D., and Zakharov, Y. (2022). Data gathering in UWA sensor networks: Practical considerations and lessons from sea trials. *Journal of Marine Science and Engineering*, 10(9):1268.

Myring, D. (1976). A theoretical study of body drag in subcritical axisymmetric flow. *The Aeronautical Quarterly*, 27(3):186–194.

Nam, H. (2018). Data-gathering protocol-based AUV path-planning for long-duration cooperation in underwater acoustic sensor networks. *IEEE sensors journal*, 18(21):8902–8912.

Nilim, A. and El Ghaoui, L. (2005). Robust control of Markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798.

Olson, E., Leonard, J. J., and Teller, S. (2007). Robust range-only beacon localization. *IEEE Journal of Oceanic Engineering*, 31(4):949–958.

Paull, L., Saeedi, S., Seto, M., and Li, H. (2013). AUV navigation and localization: A review. *IEEE Journal of oceanic engineering*, 39(1):131–149.

Pebody, M. (2007). The contribution of scripted command sequences and low level control behaviours to autonomous underwater vehicle control systems and their impact on reliability and mission success. In *OCEANS 2007-Europe*, pages 1–5. IEEE.

Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2018). Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE.

Phillips, A. B., Gold, N., Linton, N., Harris, C. A., Richards, E., Templeton, R., Thuné, S., Sitbon, J., Muller, M., Vincent, I., et al. (2017). Agile design of low-cost autonomous underwater vehicles. In *OCEANS 2017-Aberdeen*, pages 1–7. IEEE.

Phillips, A. B., Templeton, R., Roper, D., Morrison, R., Pebody, M., Bagley, P. M., Marlow, R., Chaney, E., Burris, J., Consensi, A., et al. (2023). Autosub Long Range 1500: A continuous 2000 km field trial. *Ocean Engineering*, 280:114626.

Pontbriand, C., Farr, N., Hansen, J., Kinsey, J. C., Pelletier, L.-P., Ware, J., and Fourie, D. (2015). Wireless data harvesting using the AUV sentry and WHOI optical modem. In *OCEANS 2015-MTS/IEEE Washington*, pages 1–6. IEEE.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A. Y., et al. (2009). ROS: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan.

Roper, D., Harris, C. A., Salavasidis, G., Pebody, M., Templeton, R., Prampart, T., Kingsland, M., Morrison, R., Furlong, M., Phillips, A. B., et al. (2021). Autosub Long Range 6000: A multiple-month endurance AUV for deep-ocean monitoring and survey. *IEEE Journal of Oceanic Engineering*, 46(4):1179–1191.

Roper, D. T., Phillips, A. B., Harris, C. A., Salavasidis, G., Pebody, M., Templeton, R., Amma, S. V. S., Smart, M., and McPhail, S. (2017). Autosub long range 1500: An ultra-endurance AUV with 6000 km range. In *OCEANS 2017-Aberdeen*, pages 1–5. IEEE.

Salavasidis, G., Munafò, A., Fenucci, D., Harris, C. A., Prampart, T., Templeton, R., Smart, M., Roper, D. T., Pebody, M., Abrahamsen, E. P., et al. (2021). Terrain-aided navigation for long-range auvs in dynamic under-mapped environments. *Journal of Field Robotics*, 38(3):402–428.

Sherlock, B., Morozs, N., Neasham, J., and Mitchell, P. (2022). Ultra-low-cost and ultra-low-power, miniature acoustic modems using multipath tolerant spread-spectrum techniques. *Electronics*, 11(9):1446.

Sunberg, Z. and Kochenderfer, M. (2018). Online algorithms for POMDPs with continuous state, action, and observation spaces. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, pages 259–263.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic robotics*. MIT Press.

Wei, X., Guo, H., Wang, X., Wang, X., and Qiu, M. (2021). Reliable data collection techniques in underwater wireless sensor networks: A survey. *IEEE Communications Surveys & Tutorials*, 24(1):404–431.

Wynn, R. B., Huvenne, V. A., Le Bas, T. P., Murton, B. J., Connelly, D. P., Bett, B. J., Ruhl, H. A., Morris, K. J., Peakall, J., Parsons, D. R., et al. (2014). Autonomous underwater vehicles (auvs): Their past, present and future contributions to the advancement of marine geoscience. *Marine geology*, 352:451–468.

Yan, J., Yang, X., Luo, X., and Chen, C. (2018). Energy-efficient data collection over auv-assisted underwater acoustic sensor network. *IEEE Systems Journal*, 12(4):3519–3530.

Zhuo, X., Liu, M., Wei, Y., Yu, G., Qu, F., and Sun, R. (2020). AUV-aided energy-efficient data collection in underwater acoustic sensor networks. *IEEE Internet of Things Journal*, 7(10):10010–10022.

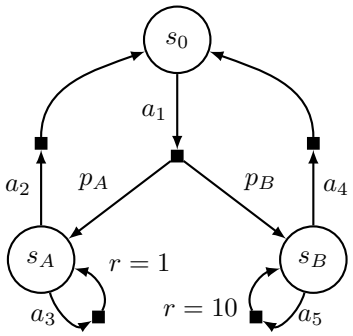
Zia, M. Y. I., Poncela, J., and Otero, P. (2021). State-of-the-art underwater acoustic communication modems: Classifications, analyses and design challenges. *Wireless personal communications*, 116:1325–1360.

3.1 Additional Discussion

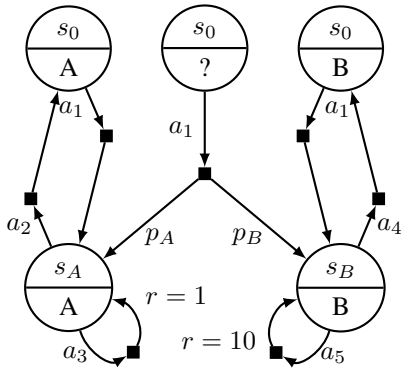
Approximately modelling uncertainty that originates from epistemic sources as an MDP with only aleatoric uncertainty is common in robotics, such as in sim2real domain randomisation (Peng et al., 2018; Ramos et al., 2019). Although some success was achieved with Markovian policies (Antonova et al., 2017), deep RL domain randomisation methods typically learn history-dependent recurrent policies. This allows these methods to achieve some adaptivity without explicitly modelling belief dynamics. We are limited to Markovian policies due to the offline planning nature of our method, which precludes learning of practical history-dependent policies.

We now discuss potential issues with the aleatoric approximation method, and discuss why these do not significantly impact the system's performance. The standard MDP framework assumes stationary transition probabilities that do not change based on the agent's accumulated experience. However, when epistemic uncertainty is converted to aleatoric uncertainty, this stationarity assumption is violated: the agent's experience should, in principle, provide information about the true underlying parameters. This should reduce the level of epistemic uncertainty and change the effective transition probabilities, but the aleatoric approximation MDP treats all uncertainty as irreducible noise. Even if epistemic parameters are not correlated between different states and transitions, multiple executions of the same action in the same state should yield information that reduces uncertainty about future executions of that same state-action pair. Figure 3.1a shows a simple MDP, with two possible transition functions. The only parameters that vary between transition functions are p_A and p_B, which must sum to 1.

Consider the case when the possible transition functions are deterministic: $p_A = 1$ and $p_B = 0$ for transition function A, and $p_A = 0$ and $p_B = 1$ for transition function B. The initial belief over transition functions is uniform. The aleatoric simplification is then to model the MDP as shown in Figure 3.1a, where $p_A = p_B = 0.5$. This case results in the worst case approximation error, because an optimal infinite horizon (or long enough finite horizon) policy for the approximate MDP will be to attempt a_1



(a) A simple MDP with an uncertain transition function. p_A and p_B vary based on the transition function.



(b) A belief state MDP for Figure 3.1a, if the possible transition functions are deterministic. The bottom of each state shows the transition function belief.

Figure 3.1: Simple examples of an MDP with epistemic uncertainty and a possible belief state MDP.

until reaching s_B. In the case that the true transition function is A, this will result in behaviour that never incurs any reward. Optimal behaviour in this epistemically uncertain MDP is achieved by a policy for a simple belief state MDP as shown in Figure 3.1b. The lower half of each state shows the transition function belief for that state, starting with "?" representing the prior over transition functions. This belief state MDP is finite due to the deterministic nature of the possible transition functions, where the true transition function is known immediately after a_1 is taken. If the possible transition functions were not deterministic, the belief state MDP would be infinite. Note that the more similar the possible transition functions are, the more attempts of action a_1 it would take to determine the true transition function with high confidence.

Higher levels of epistemic uncertainty in some states will be modelled as higher levels of aleatoric uncertainty. Depending on the problem setting, even the same level of epistemic uncertainty may lead to different levels of aleatoric uncertainty in different states. State-dependent aleatoric uncertainty can be straightforwardly represented by the stochastic outcomes of an MDP, but this fundamentally obscures the true behaviour of the system: in some states this uncertainty could be reduced, in principle, by gathering more information, but in other states it cannot be.

In practice, our method described in this chapter avoids the worst case described above. This is largely because the transition function uncertainty set encompasses a continuum of possible transition functions, with significant overlap in their outcome probabilities, rather than being limited to two mutually exclusive deterministic functions. Additionally, as the model is a timed MDP with a mission time bound, the agent is disincentivised from repeatedly retrying durative actions to get slightly better outcomes. In the next chapter, the problem setting and method make this issue more apparent, and we discuss how it is addressed by online replanning.

Another consequence of many possible transition functions explaining the agent's experience during a mission is that an adaptive method offers limited benefit, even if it were feasible given the vehicle's compute constraints. The small amount of data gathered during a single mission would not meaningfully change the prior over underwater current dynamics, leading to no change in behaviour.

One alternative to adaptive behaviour would be a robust MDP method (Nilim and El Ghaoui, 2005), which assumes an adversarial environment and aims to maximise the worst case performance. The most common approaches to robust MDPs make the rectangularity assumption for computational tractability (Iyengar, 2005). This is the assumption of independence between action outcomes. In our domain, one result of this would be the outcome that experienced water currents are always directly against the vehicle's direction of travel, which would be a very conservative and unrealistic assumption.

3.2 Limitations and Future Work

Our method is able to achieve some aspects of partial observability-aware behaviour, using the transition from "unknown" data state to an initial data state distribution (paper Section 4.7.2). This simplification works well for the fixed data distribution collection case, but could be improved for longer missions with in-mission data generation. Namely, there is no principled mechanism for the vehicle's belief of sensor node data contents to become more uncertain if it travels far from a node after first contacting it. Integrating an offline POMDP planning method with

the data contents models, for example point-based value iteration (Pineau et al., 2003), would be a more principled (and for some more complex experiment settings, better performing) approach. Also relating to the solution method, the value iteration algorithm used to solve the TMDP provides a complete policy. A heuristic search dynamic programming algorithm, such as LRTDP (Bonet and Geffner, 2003), would likely provide faster solution times and smaller policy sizes. However, one advantage of the complete policy is that, even if the real environment dynamics differ significantly to the extent that the AUV enters a state that should be unreachable under the policy and the model, there will still be a valid action to take. The AUV can then attempt to continue the mission from this state.

Beyond a full POMDP treatment of the data retrieval process, an adaptive method that accounts for parameter uncertainty could improve reasoning about data retrieval process in some circumstances. Although the data CTMDPs are constructed using distributions of possible data retrieval rates, these are aggregated into an expectation in order to calculate the transition probabilities (paper, Eq. 13 and Section 4.6.1). The data retrieval process (and data regeneration process, in the in-mission data generation case) is therefore represented with fixed probabilities in the corresponding CTMDP. The method therefore only considers the expected communication performance for the data retrieval process, no matter the other aspects of the distribution of acoustic communication-related parameters. When the ground-truth generation/regeneration rates differ during a mission execution, these are effectively considered to be "unlikely" stochastic outcomes by the model. Maintaining epistemic parameters for data retrieval rates for each node would be possible, but would require belief planning and therefore a larger pre-computed policy or more onboard computation.

One potentially useful addition to the modelling approach, not currently implemented, would be to account for completely incommunicable sensor nodes. In the case where a sensor node is inoperable or completely unreachable due to unmodelled acoustic effects, the vehicle would ideally be able to conclude this and avoid spending time attempting to communicate with it. This would be an example of simple

binary epistemic uncertainty (reachable or unreachable sensor node), but would still increase the state space size at least polynomially in the number of sensor nodes.

Finally, though we define and use belief distributions over model parameters, we do not prescribe how these parameters and models are updated between missions using experience collected by the robot. This is largely due to the complexity of ocean current and acoustic communication models, and the fact that this is not the focus of the paper. However, the system concept is designed to incorporate such updates between missions (paper, Section 3.4).

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	AUV-Based Data Harvesting from Underwater Sensor Networks Using Probabilistic Planning
Publication Status	□ Published □ Accepted for Publication ☑ Submitted for Publication □ Unpublished and unsubmitted work written in a manuscript style
Publication Details	**Matthew Budd**, Georgios Salavasidis, Izzat Kamarudzaman, Benjamin Sherlock, Jeffrey Neasham, Bruno Lacerda, Nick Hawes, Alexander B. Phillips, and Catherine Harris. AUV-based data harvesting from underwater sensor networks using probabilistic planning. *Under Review.*

Student Confirmation

Student Name:	Matthew Budd		
Contribution to the Paper	• I developed the planning approach proposed in the paper. • I implemented the planner and plan executor, and tested the system in field trial and simulation. • I wrote all planner-related sections of the paper (Sections 2, 3.4, 4, 5.1.1, 5.1.3, 5.2, 6.4, 7), with review and input from Bruno Lacerda and Nick Hawes.		
Signature		Date	April 24, 2025

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title:	Professor Nick Hawes		
Supervisor comments	I confirm that Matthew made a substantial contribution to the publication, and that the description above is accurate.		
Signature		Date	April 25, 2025

This completed form should be included in the thesis, at the end of the relevant chapter.

4 Replanning with New Experience

	Chapter 3	Chapter 4	Chapter 5	Chapter 6	Chapter 7
Epistemic uncertainty	Environment dynamics	Environment dynamics	Environment dynamics	Planner dynamics	Planner dynamics
Mode	Offline	Replanning	Online	Offline	Online
Model	Timed MDP	EstMDP	BAMDP	Metalevel MDP	Metalevel MDP
Solution method	VI (max reward)	VI (reachability)	MCTS	Deep RL	Deep RL
Data regime	Small/ medium	Small	Small	Large	Large

Table 1.1: Settings, methods and features for each chapter.

In this chapter, we study the MDP safe exploration problem (Moldovan and Abbeel, 2012). In the simpler Bayesian optimisation (BO) setting, the safe exploration task (as most commonly posed) is to find the global maximum of an unknown scalar objective function, while avoiding sampling function values at locations $\mathbf{x}$ where a safety function $f(\mathbf{x})$ is below a certain threshold (Sui et al., 2015; Bottero et al., 2022). The safety and objective functions may be the same function, and the goal may be to maximise the objective function or to model it with high accuracy within the safe region. In MDP safe exploration, the function input space $\mathbf{x}$ is the state space of an MDP. This introduces additional complexity: as well as determining *where* to safely and informatively sample the function, the agent must determine how to safely *reach* and *return from* the sample locations.

We assume a more general problem setting than previous MDP safe exploration methods. Previous methods apply only to deterministic MDPs (Turchetta et al.,

2019; Wachi and Sui, 2020; Prajapat et al., 2022) or deterministic non-linear systems (Prajapat et al., 2025), whereas ours applies to stochastic MDPs. Furthermore, in our setting, the unknown function may be multidimensional, and the MDP transition and reward functions may depend on the unknown function. This greatly increases the class of problems that can be solved, as demonstrated by the complex underwater current mapping problem in the experiments.

The method in this chapter is based on two formalisms developed by the thesis author: the PKSMDP and the EstMDP.

The Partially Known State MDP (**PKSMDP**) describes the interaction between the agent and the environment, by separating the *known* agent dynamics from the *unknown* environment dynamics. The transition function of the agent is fully defined, but is *conditioned on* the environment dynamics at the agent's current state. For example, if the agent is an autonomous underwater vehicle, the navigation dynamics of the vehicle can be predicted given a model of the vehicle and specific water current vectors, even if the whole current field is not known. However, the PKSMDP cannot be straightforwardly solved as a standard MDP, as the environment dynamics in the transition function are not known.

The Estimated MDP (**EstMDP**) is the combination of the PKSMDP and a Gaussian process (GP) (Rasmussen and Williams, 2006) model of the environment dynamics, incorporating the environment measurements made by the agent so far. We use Gaussian processes because they suit the noisy, local sensing capabilities of the agent, and are considered "the gold standard for many real-world modeling problems" (Wilson et al., 2020). This is particularly true for the natural phenomena we are interested in modelling, such as radiation and ocean currents. The EstMDP captures the agent's belief about the environment dynamics, including stochastic outcomes due to possible environment dynamics. It is therefore similar to the TMDP model in the previous chapter, in that it approximates epistemic uncertainty as aleatoric uncertainty.

The method explores the environment by selecting informative goal states to reach, balancing the informativeness of the goal with the safety and cost of the

path to the goal. Example safety limits include maximum values for radiation level, terrain steepness, or water current velocity in a specific direction. While the goal-reaching policy is executed, action outcomes and new measurements of the environment are used to update the EstMDP. Rather than a fully online method that solves the new EstMDP at each step, we develop a replanning method based on model checking the current policy against the new model. Replanning occurs when the policy is no longer safe or becomes too costly, or when the agent has reached a goal state. While an online method may sometimes be able to identify a better goal state or a better path to the current goal state given the new model, it would require re-solving the EstMDP after every action execution. Given even modest computational constraints on the robot, this would hugely increase the time taken to explore the environment. Given that model checking the existing policy is less complex than new policy synthesis (Baier and Katoen, 2008), our replanning method maintains safety and computational tractability while allowing the agent to adapt to the environment dynamics. We discuss in detail why it is necessary to model check the policy during execution (compared to the previous chapter, where we do not do so) in Section 4.1.

The PKSMDP (then named a U-MDP) and EstMDP were originally introduced by Budd et al. (2020). The method in this chapter uses the models in a new replanning-based framework, which is better able to adapt to the environment dynamics. We also carry out extensive experimental evaluation of the method, including a real-world deployment. To enable the real-world experiment, the algorithm was integrated into a new frontier-based 3D map-building framework. The frontier-based system was designed and integrated by the other joint first author of this work. By incorporating the safe exploration algorithm into the framework, a limitation of previous frontier-based exploration methods—namely, the lack of safety considerations—is addressed (Akbari and Bernardini, 2021; Batinovic et al., 2021).

Other than broadening the class of environments that can be safely explored, our method outperforms existing methods in terms of exploration efficiency while achieving the same level of safety. We do this by reasoning about the safety and

cost of multiple-step paths to new goal states, rather than only single steps to new states (Turchetta et al., 2019; Wachi and Sui, 2020).

Citation:

Alex Stephens, Matthew Budd, Michal Staniaszek, Benoit Casseau, Paul Duckworth, Maurice Fallon, Nick Hawes, and Bruno Lacerda. Planning under uncertainty for safe robot exploration using Gaussian process prediction. *Autonomous Robots*, 48(7):18, 2024

Planning under uncertainty for safe robot exploration using Gaussian process prediction

Alex Stephens[1] · Matthew Budd[1] · Michal Staniaszek[1] · Benoit Casseau[1] · Paul Duckworth[2] · Maurice Fallon[1] · Nick Hawes[1] · Bruno Lacerda[1]

Received: 10 May 2023 / Accepted: 17 July 2024

Abstract
The exploration of new environments is a crucial challenge for mobile robots. This task becomes even more complex with the added requirement of ensuring safety. Here, safety refers to the robot staying in regions where the values of certain environmental conditions (such as terrain steepness or radiation levels) are within a predefined threshold. We consider two types of safe exploration problems. First, the robot has a map of its workspace, but the values of the environmental features relevant to safety are unknown beforehand and must be explored. Second, both the map and the environmental features are unknown, and the robot must build a map whilst remaining safe. Our proposed framework uses a Gaussian process to predict the value of the environmental features in unvisited regions. We then build a Markov decision process that integrates the Gaussian process predictions with the transition probabilities of the environmental model. The Markov decision process is then incorporated into an exploration algorithm that decides which new region of the environment to explore based on information value, predicted safety, and distance from the current position of the robot. We empirically evaluate the effectiveness of our framework through simulations and its application on a physical robot in an underground environment.

Keywords Safe exploration · Mobile robots · Markov decision processes · Gaussian processes

1 Introduction

Mobile robot tasks often involve navigating through environments with hazardous conditions. These hazards can take the form of steep terrain for planetary rovers, underwater currents or shallow water depth for underwater vehicles, or radiation levels in disaster recovery or nuclear inspection. In this paper, we propose a safe exploration method that handles cases where the values of the hazardous environmental features are *unknown* a priori. We consider two settings: one where the robot already has a map of the area to be explored and wants to learn about the hazards in a safe manner, and another where the robot lacks a map and needs to create one while considering the hazards and maintaining safety.

The core of our framework is based on modelling the robot's navigation under the hazardous features with unknown values as a partially known state Markov decision process (PKSMDP), and utilising a Gaussian process (GP) to estimate the unknown values across the environment. The GP's capability to quantify its *prediction uncertainty* is crucial, since ensuring safety with a high degree of certainty requires the robot to assess its confidence in the predictions

Alex Stephens, Matthew Budd have contributed equally to this work.

Alex Stephens
stephens@robots.ox.ac.uk

Matthew Budd
mbudd@robots.ox.ac.uk

Michal Staniaszek
michal@robots.ox.ac.uk

Benoit Casseau
benoit@robots.ox.ac.uk

Paul Duckworth
p.duckworth@instadeep.com

Maurice Fallon
mfallon@robots.ox.ac.uk

Nick Hawes
nickh@robots.ox.ac.uk

Bruno Lacerda
bruno@robots.ox.ac.uk

[1] Oxford Robotics Institute, University of Oxford, Oxford, UK

[2] InstaDeep, London, UK

Published online: 28 August 2024

about the environmental features. To plan considering this uncertainty, we propose encoding the GP uncertainties into the transition probabilities of the PKSMDP, yielding a model we call the *Estimated* MDP (EstMDP).

We propose two novel exploration algorithms that use the EstMDP to plan safe paths to states expected to be informative. These safe paths take into account the probability of falling into an unsafe state when navigating to an informative state, or when travelling back from an informative state to a state with guaranteed safety. The first algorithm, *SafeEstMDP-Process*, extends the work in Turchetta et al. (2016) to handle probabilistic transition models and support more complex safety specifications. SafeEstMDP-Process selects new goal locations that reduce the uncertainty of the GP estimates of the unknown hazard. The second algorithm, *SafeEstMDP-Map*, integrates this framework with an online mapping system, thereby extending it to settings where the map of the workspace the robot is operating in is also unknown. SafeEstMDP-Map incrementally builds a navigation PKSMDP online using data from onboard sensors, and selects new goal locations based on expected information gain of the underlying occupancy map representation of the workspace.

We extensively evaluate our algorithms in simulations of real-world environments, empirically showing that it significantly outperforms the work of Turchetta et al. (2016) and hand-crafted baselines, safely achieving increased environmental exploration while also minimising the distance travelled in the process. We also report on a trial performed using a Boston Dynamics Spot deployed in an underground mine section at Corsham, Wiltshire, UK.

The contributions of this paper are as follows:

- The PKSMDP, a novel model which formalises problems of safe exploration with unknown hazardous features under uncertain action outcomes;
- The encoding of GP predictions into the PKSMDP transition function, yielding the EstMDP planning model;
- Two exploration algorithms, under different assumptions of prior knowledge of the environment map, which make use of the EstMDP to decide where to explore next;
- An empirical evaluation emphasising the benefits of our approach based on GP predictions and planning under uncertainty.

2 Related work

Planning and exploration with MDPs and GPs. MDPs are a widely used formalism for planning under uncertainty for robots, e.g. Feyzabadi and Carpin (2017), Lacerda et al. (2019), Gopalan et al. (2017). GP-modelled unknown process exploration has commonly been investigated in the Bayesian optimisation setting. Sui et al. (2015) introduced the idea of using GP regression to maximise an objective function value while ensuring high probability of not sampling function values that break a safety bound. This is done by considering the upper confidence bounds given by the GP when exploring. This work has been extended to consider safety functions which are separate from the objective function (Sui et al., 2018) and to reason explicitly about the information gain about the safety of the parameters (Bottero et al., 2022), in order to improve sample efficiency.

Safe exploration of an MDP introduces the need to reason about state reachability, which is not required in the Bayesian optimisation setting where observations are assumed to be taken freely across the domain. An early work on safe exploration of MDPs was Moldovan and Abbeel (2012), which investigated the problem of exploring an MDP whilst ensuring returnability to the initial state but assumes full knowledge of the safety of states. Turchetta et al. (2016) introduced the *SafeMDP* algorithm for safe exploration of MDPs using GPs. SafeMDP reasons about both *reachability* from and *returnability* to the already visited set of states, which must be safe if the safety bound has not been broken. However, SafeMDP only considers new states to visit that are a single step from an already explored state. We extend the reachability and returnability concepts to reason about multiple-step safety probabilities along paths. Along with the inclusion of transition costs in the exploration objective, this makes our algorithm far more cost-efficient and less myopic when exploring. Finally, we alleviate SafeMDP's deterministic transition function assumption and explore MDPs with probabilistic transitions than allow for modelling the impact of the unknown process on the robot's dynamics. For example, if the unknown process represents underwater currents acting on an autonomous underwater vehicle (AUV), high magnitude currents could push the robot into waypoints other than the intended goal waypoint (Budd et al., 2022). Some actions may even be entirely infeasible because the water current velocity from the goal waypoint towards the AUV is higher than the maximum water-frame speed of the AUV. Wind can have similar effects on an unmanned aerial vehicle (Badings et al., 2022). If the unknown process is terrain steepness, higher terrain gradients could make it more likely that a wheeled robot slips into an unintended waypoint while trying to navigate to the goal waypoint (Rutherford et al., 2021). Other extensions of SafeMDP include multi-agent safe exploration (Zhu et al., 2020).

The standard MDP formulation assumes a fully known model of the environment, and full observability of the current state. For sequential decision-making under state uncertainty, the partially observable MDP (POMDP) formulation is commonly used (Kaelbling et al., 1998; Spaan et al., 2015; Lauri et al., 2019). Outside of the safety-constrained

setting, GPs have been used as POMDP belief models to carry out unknown process exploration in GP-modelled environments (Marchant et al., 2014; Morere et al., 2017; Flaspohler et al., 2019).

As with existing MDP safe exploration algorithms above, we do not use a partially observable formulation of the problem to avoid the computational complexity of POMDP planning. Although our approach will act more myopically than a POMDP formulation, it requires far less computation. Furthermore, given that models for exploration tend to be inaccurate, it is common to use a next-best-view approach as we propose here, rather than spend computational resources performing lookaheads based on an inaccurate model. Our approach also has similarities to a mixed observability MDP (Ong et al., 2010), where the robot's location is observable but the environment process is only partially observable.

Exploration of unknown environments. The general problem of exploration of unknown environments has been addressed through frontier-based approaches, which aim to guide the robot towards unknown areas of the map (Yamauchi, 1998; Yamauchi et al., 1998; Freda & Oriolo, 2005). The research on this topic has recently received a lot of attention due to the DARPA Subterranean Challenge (SubT), where teams of robots must autonomously explore underground environments. In the context of SubT, extensive teams of researchers have developed and tested complex integrated systems which address problems such as localisation and mapping in GPS-denied environments, navigation in adverse terrains, or coordination of heterogeneous multi-robot teams under communication constraints (Tranzatto et al., 2022; Rouček et al., 2022; Hudson et al., 2022; Morrell et al., 2022; Scherer et al., 2022). Numerous exploration algorithms have been developed in the context of SubT, a significant portion of which rely on frontier-based methods (Bayer & Faigl, 2019; Dang et al., 2020; Williams et al., 2020). Our SafeEstMDP-Map approach also uses a notion of frontiers to drive the robot towards unknown areas of the map. However, because it must maintain safety with regards to the unknown process, we also consider possible points which are not at the edge of known space but for which the GP prediction of the unknown process still has high variance. Given the large scale of the SubT domains, exploration approaches are often hierarchical, with the aim of ensuring thorough exploration at a local level whilst considering when to move to other unexplored areas of the map at a global level (Dang et al., 2020; Cao et al., 2021; Kim et al., 2021). Furthermore, approaches used in SubT have a strong focus on the *traversability* of edges in the navigation graph. In contrast, we either assume the traversability is given, in the form of a navigation MDP, or that traversability can be checked by simple ray casting, assuming that the robot can traverse an edge unless there is an obstacle on the way. Moreover, we do not separate the exploration algorithm between local and global levels. This is done because our focus is on the use of GPs to model an external unknown process and reason about safety with regards to it. Note that this aspect goes beyond what is considered in the SubT works. We also note that our approach can in principle be integrated with these more complex exploration approaches, ensuring an extra layer of safety against external processes.

3 Preliminaries

3.1 Markov decision processes & constrained reachability

Definition 1 An MDP is defined as a tuple $\mathcal{M} = \langle S, \bar{s}, A, T, c\rangle$, where S is a finite set of states; $\bar{s} \in S$ is the initial state; A is a finite set of actions; $T : S \times A \times S \to [0, 1]$ is a probabilistic transition function; and $c : S \times A \to \mathbb{R}_{\geq 0}$ is a cost function.

Examples of cost functions are the expected time to execute an action, or the expected energy required to do so. A *path* through an MDP is a sequence $w = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots$ where $T(s_i, a_i, s_{i+1}) > 0$ for all $i \in \mathbb{N}$. We denote the set of all paths of $\mathcal{M}$ starting from state s as $Path_{\mathcal{M},s}$. The choice of action to take at each step of the execution of an MDP is made by a *policy*. In this paper, we consider *deterministic, stationary* policies, defined as functions $\pi : S \to A$ that map each state $s \in S$ to the action to execute in s, and denote the set of all such policies as Π. Given an MDP $\mathcal{M}$ and a policy $\pi \in \Pi$, we can define a probability measure $Pr^{\pi}_{\mathcal{M},s}$ over the set of paths $Path_{\mathcal{M},s}$ (Kemeny et al., 1976). Furthermore, for a measurable function $X : Path_{\mathcal{M},s} \to \mathbb{R}$, we write $E^{\pi}_{\mathcal{M},s}(X)$ to denote the expected value of X with respect to $Pr^{\pi}_{\mathcal{M},s}$.

We consider *cost-optimal reach-avoid* problems for which the probability of satisfaction might be less than one. These involve identifying a policy to reach a set of goal states whilst avoiding a set of forbidden states.

Definition 2 Let $G \subset S$ be a set of goal states and $F \subset S$ be a set of forbidden states. We define the set of paths that reach G whilst avoiding F as:

$$\begin{aligned} reach_{\neg F,G} = \{(s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots) \in Path_{\mathcal{M},s_0} \mid \\ \text{exists } i \in \mathbb{N} \text{ such that} \\ s_i \in G \text{ and } s_j \notin F \text{ for all } j \leq i\}. \end{aligned} \tag{1}$$

We will consider policies that are cost-optimal, in the sense that they minimise the expected cumulative cost to reach either a goal state, or a state where reaching the goal is not possible.

Definition 3 Let $w = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots \in Path_{\mathcal{M},s_0}$. We define l_w as the timestep until which cost will be accumulated for path w:

$$l_w = \begin{cases} \min_l \text{ s. t. } s_l \in G & \text{if } w \in reach_{\neg F,G} \\ \min_l \text{ s. t. } & \\ Pr^{\max}_{\mathcal{M},s_l}(reach_{\neg F,G}) = 0 & \text{otherwise,} \end{cases} \tag{2}$$

where $Pr^{\max}_{\mathcal{M},s_l}(reach_{\neg F,G})$ denotes the maximum over Π of $Pr^{\pi}_{\mathcal{M},s_l}(reach_{\neg F,G})$.

Note that the second condition in the definition of l_w encompasses the case where a forbidden state is visited before a goal state, and the case where the path can never reach a goal state. We can now define the cost-optimal reach-avoid problem.

Problem 1 Let $cumul_{\neg F,G} : Path_{\mathcal{M},s} \to \mathbb{R}_{\geq 0}$ be a function that maps a path $w = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots$ to the cost accumulated up to l_w:

$$cumul_{\neg F,G}(w) = \sum_{i=0}^{l_w - 1} c(s_i, a_i), \tag{3}$$

and $\Pi^* = \{\pi \in \Pi \mid \pi = \arg\max_{\pi'} Pr^{\pi'}_{\mathcal{M},\bar{s}}(reach_{\neg F,G})\}$ be the set of policies that maximise the probability of reaching G whilst avoiding F. The cost-optimal reach-avoid problem is defined as finding the policy $\pi_{\neg F,G} \in \Pi^*$ that has minimal expected cumulative cost:

$$\pi_{\neg F,G} = \arg\min_{\pi \in \Pi^*} E^{\pi}_{\mathcal{M},\bar{s}}(cumul_{\neg F,G}). \tag{4}$$

The above optimisation problem is a variant of a safest and stochastic shortest path problem (Teichteil-Königsbuch 2012). The minimal expected cost for these problems is known to converge to a finite value. We will write $E^*_{\mathcal{M},\bar{s}}(cumul_{\neg F,G})$ to denote the (minimal) expected value corresponding to $\pi_{\neg F,G}$. In order to find $\pi_{\neg F,G}$, we encode the constrained reachability problem in *co-safe linear temporal logic* and use the approach presented in Lacerda et al. (2019).

3.2 Gaussian processes

A GP (Rasmussen & Williams, 2006) is defined as a collection of random variables, any finite number of which have a joint Gaussian distribution. A GP is fully specified by its mean function $m(s)$ and kernel function $k(s, s')$, i.e. $f(s) \sim \mathcal{GP}(m(s), k(s, s'))$. We let $m(s) = 0$ without loss of generality. The kernel function k is parameterised by hyperparameters θ that encode prior assumptions over the unknown function f. We make the standard modelling assumptions that f has bounded norm in the Reproducing Kernel Hilbert Space associated with the chosen kernel function, and also that it is Lipschitz continuous with respect to some metric $d(\cdot, \cdot)$ on S (Turchetta et al., 2016; Wachi et al., 2018).

Definition 4 Let $\mathcal{D} = \{s_i, z_i\}_{i=1}^{n_\mathcal{D}}$ be a dataset of $n_\mathcal{D}$ noisy observations of the form $z_i = f(s_i) + \epsilon_i$, where each $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$. For a test point s_*, the GP posterior is defined as:

$$P^{\mathcal{GP}}(f(s_*) \mid \mathcal{D}) \sim \mathcal{N}(\mathbf{k}_*^\mathsf{T}(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{z}, \\ k(s_*, s_*) - \mathbf{k}_*^\mathsf{T}(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{k}_*), \tag{5}$$

where $\mathbf{z} = [z_1, \ldots, z_{n_d}]^\mathsf{T}$; $\mathbf{I} \in \mathbb{R}^{n_\mathcal{D} \times n_\mathcal{D}}$ is the identity matrix; $\mathbf{K} \in \mathbb{R}^{n_\mathcal{D} \times n_\mathcal{D}}$ is the positive semi-definite kernel matrix such that $\mathbf{K}_{i,j} = k(s_i, s_j)$; and $\mathbf{k}_* = [k(s_1, s_*), \ldots k(s_{n_d}, s_*)]^\mathsf{T}$.

It is possible to account for noise in the dataset GP inputs s_i as well as observations z_i of $f(s_i)$ by transforming the GP to maintain the noise-free input assumption. This is done by adding a correction term to the GP observation noise (McHutchon & Rasmussen, 2011).

3.3 Occupancy map models of unknown environments

Consider a bounded environment $X \subset \mathbb{R}^n$, partitioned into free and occupied space. This is represented by sets X_{free} and X_{occ}, respectively. We will maintain these sets using *occupancy maps* (Elfes, 1989) as they provide a versatile representation of unstructured environments observed through noisy sensor measurements. To simplify the presentation, we will consider $n = 2$ for the remainder of the paper, i.e., we assume the robot is deployed in an environment which can be navigated in a horizontal 2D plane. However, the methods proposed here can generalise to consider a 3D environment. Thus, we represent the environment as a grid map $\mathbb{M} = \{m_1, \ldots m_\ell\}$ of ℓ cells $m \subset \mathbb{R}^2$.

We keep a classification of cells $m \in \mathbb{M}$ as either free space, occupied space, or unknown space. There are several ways to maintain such a classification. In our implementation, we use the OctoMap framework (Hornung et al., 2013), an octree-based library that provides a probabilistic occupancy estimate of each 3D voxel in the environment, projecting it to 2D by taking a slice of the OctoMap at the height of the robot's lidar.

We are interested in maximising exploration coverage over an unknown environment. To do so efficiently, we use the notion of *volumetric gain* from Dang et al. (2020), which estimates the unmapped volume that could be perceived by an onboard sensor from a given location $x \in X$.

Definition 5 Let N^x be the set of cells observed from location $x \in X$. We define volumetric gain at x as a weighted sum

over the cells in each class observable from x, by casting uniformly-spaced rays outwards in all directions:

$$\texttt{gain}(x) = \alpha_{unk} N^x_{unk} + \alpha_{free} N^x_{free} + \alpha_{occ} N^x_{occ}, \quad (6)$$

where N^x_c is the number of cells observable from x of class c, and $\alpha_{unk} > \alpha_{free} > \alpha_{occ} \geq 0$.

The α_c coefficients are normalised such that a gain value of 1 represents every ray cast seeing purely unknown space, and a value of 0 representing the robot being completely enclosed by occupied space. The specific values of the α parameters that produce the best performance will depend on the structure and connectivity of the environment: for example, whether it is made up of constrained corridors or wide-open spaces.

Note that our OctoMap-based environment representation already is a 3D representation that we project to 2D. The exploration algorithms we propose also generalise naturally to 3D, thus extending our framework to 3D is straightforward.

4 Modelling and Problem formulation

4.1 Underlying model

We consider a set of environmental processes which are unknown to the robot a priori. These processes represent environmental phenomena that vary across the environment X, such as radiation levels or underwater currents, for which the robot can draw noisy sensor measurements at its current location.

Definition 6 *(Unknown Processes)* The unknown processes at environment X are defined as $f : X \to O$ where $O = \mathbb{R}^m$ is the observation space of f, with m being the number of modelled unknown processes. The robot can observe noisy sensor measurement $\omega : X \to \mathbb{R}^m$ of the form:

$$\omega(x) = f(x) + \epsilon, \quad (7)$$

where $\epsilon = [\epsilon_1, \ldots, \epsilon_m]$ is an m-dimensional vector of Gaussian observation noise, i.e. $\epsilon_j \sim \mathcal{N}(0, \sigma_j^2)$ for all $j \in \{1, \ldots, m\}$.

For the remainder of the paper, we will consider $m = 1$, i.e., we assume there is a single unknown process. However, the methods proposed can easily generalise to more than one unknown process, either using a multi-output GP (Osborne et al., 2008) or multiple single-output GPs. The former assumes non-independent process dynamics, where learning about one process could improve predictions of another.

For the purpose of planning and navigation, we represent the environment as a set of d waypoints $\mathrm{V} = \{x_1, \ldots, x_d\} \subset X$, and consider a stochastic model of navigation between waypoints under the influence of the unknown process.

Definition 7 A *partially known state MDP (PKSMDP)* for navigation is a tuple $\mathcal{M}^O = \langle S^O, \overline{s^O}, A^O, T^O, C^O \rangle$ where:

- $S^O = \mathrm{V} \times O$, i.e., the state space is composed of a location in the environment v and the corresponding value $\underline{f}(\mathrm{v})$ of the unknown process;
- $s^O = (\overline{\mathrm{v}}, f(\overline{\mathrm{v}}))$, where $\overline{\mathrm{v}}$ is the robot's initial position;
- $A^O = V \times V$ is a set of navigation actions such that $(v, v') \in A^O$ if the robot can navigate from v to v' without visiting any other waypoint on the way;
- $T^O : (\mathrm{V} \times O) \times A^O \times \mathrm{V} \to [0, 1]$, where $T^O((v, o), (v, v'), v'')$ is the probability of the robot reaching waypoint v'' given that it attempted to reach waypoint v' from v, and the value of the unknown process at v is o;
- $C^O : (\mathrm{V} \times O) \times A^O \to \mathbb{R}_{\geq 0}$ is the cost function, where $C^O((v, o), (v, v'))$ is the cost of attempting to navigate from waypoint v to waypoint v' when the value of the unknown process at v is o.

Note that a PKSMDP is *not* a standard MDP as formalised in Definition 1. In particular, the transition function is defined such that the action outcomes are a distribution over only the set of waypoints. This is because the underlying state is uniquely defined by the value of the waypoint v, i.e., $s^O = (v, f(v))$. Furthermore, since f is unknown and the robot is only able to make noisy observations, the PKSMDP cannot be directly used for planning. Finally, note that S^O is not a discrete set because its component O is continuous. In Sect. 5, we will propose a model that estimates the PKSMDP.

Previous approaches generally model the unknown process as a cost function rather than as part of the state. By including unknown process values as part of the state, the PKSMDP can model the impact of these process values on the robot dynamics.

We also consider a safety function defined over the values of the known and unknown state features.

Definition 8 A safety function is a mapping $\phi : S^O \to \{0, 1\}$ where $\phi(s^o) = 1$ when s is considered safe and 0 otherwise. The sets of safe and unsafe states can then be defined as $safe_\phi = \{s^o \in S^O \mid \phi(s^o) = 1\}$ and $unsafe_\phi = S^O \backslash safe_\phi$.

A simple safety function which depends only on a single unknown value state feature could be an upper-bound $b \in \mathbb{R}$ on the value of that feature, as used in Sect. 8.1.1. However, our approach allows for safety to be defined as an arbitrary Boolean function over the state-space of the PKSMDP, something which is not possible with existing approaches. We demonstrate such a safety function in Sect. 8.1.2.

4.2 Problem formulation

We now pose the two problems addressed in this paper. First, we assume that the area has been previously spatially mapped and there is a known PKSMDP defined over the mapped area. The goal is to accurately estimate f whilst remaining safe.

Problem 2 Let $\mathcal{M}^O = \langle S^O, \overline{s^O}, A^O, T^O, C^O \rangle$ be a PKSMDP and $\phi : S^O \to \{0, 1\}$ a safety function. Estimate the unknown process $f : X \to O$ whilst avoiding states $s^o = (v, o)$ where $s^o \in \mathit{unsafe}_\phi$.

In Problem 2, we know the underlying spatial structure of the environment, and assume it has been discretised into a set of relevant waypoints. This discretisation can be achieved manually by a designer, or automatically, e.g. by employing Voronoi diagrams or gridding the space, removing edges and cells that intersect with the obstacles. Quantifying the quality of our estimate depends on the robot's objective and the approach used for estimation, thus we deliberately leave it undefined in the problem definition above. In our proposed approach, we use the variance of the GP predictions of f on the waypoints, as will be explained in Sect. 6.2.

Second, we remove the assumption of a known map – and hence of a known PKSMDP – and consider the problem of safely exploring and mapping the environment. To do so, we assume the robot is equipped with a sensor (e.g. lidar) that is able to build an occupancy map incrementally, which is in turn used to classify voxels as unknown, occupied or free as described in Sect. 3.3.

Problem 3 Let X be a bounded environment, $\mathbb{M}$ a grid map representation of X, and $\phi : S^O \to \{0, 1\}$ a safety function. Classify as much of $\mathbb{M}$ as possible as being in free or occupied, whilst avoiding states $s^o = (v, o)$ where $s^o \in \mathit{unsafe}_\phi$.

To solve Problem 3, we will adapt the solution of Problem 2 to incrementally build and solve estimates of a PKSMDP rather than consider the exploration of a fixed underlying PKSMDP.

5 GP-based PKSMDP estimation

Central to our approach is the construction of an MDP that estimates the unknown process represented in the PKSMDP given the data already observed by the robot. Our approach to do so is to use a GP, trained on observations up to timestep t, to iteratively estimate the mapping o between known state feature values and their corresponding unknown state feature values.

We start by partitioning the domain of the unknown process O into a finite set of intervals of possible values of the unknown process, allowing us to abstract the continuous aspect of the problem.

Definition 9 We consider the finite partitions of O defined as the sets of intervals $\mathcal{I}$. Given $o \in O$, we write $I[o]$ to denote the interval in $\mathcal{I}$ that contains o.

Partitioning the continuous space of the unknown process into a finite number of intervals enables the use standard solution techniques for MDPs, removing the need to consider continuous state spaces. As usual in this kind of discretisation, increasing the number of intervals leads to a more fine grained representation of the unknown process, at the cost of increased computational complexity.

We consider the GP posterior over these intervals.

Definition 10 Let $\mathcal{D}$ be a dataset of observations of the unknown process f, $v_* \in \mathrm{V}$ be a waypoint, and $I \in \mathcal{I}$ be an interval. The probability of the value of unknown process in v_* being in interval I is defined as:

$$P^{\mathcal{GP}}(f(v_*) \in I \mid \mathcal{D}) = \int_{o \in I} P^{\mathcal{GP}}(f(v_*) = o \mid \mathcal{D})\, do. \quad (8)$$

We can now define the *Estimated MDP* (EstMDP), which encodes both the probabilistic transition function of the PKSMDP and the GP model given the observed dataset $\mathcal{D}$.

Definition 11 Let $\mathcal{M}^O = \langle S^O, \overline{s^O}, A^O, T^O, C^O \rangle$ be a PKSMDP, $\mathcal{I}$ be the set of intervals, and $\mathcal{D}$ be a dataset of observations of the unknown process f.

The estimated MDP is a tuple $\mathcal{M}^{\mathcal{D}} = \langle S^{\mathcal{D}}, \overline{s^{\mathcal{D}}}, A^O, T^{\mathcal{D}}, C^O \rangle$, where $T^{\mathcal{D}} : (\mathrm{V} \times \mathcal{I}) \times A^O \times (\mathrm{V} \times \mathcal{I}) \to [0, 1]$, where:

- $S^{\mathcal{D}} = V \times \mathcal{I}$, i.e., the state space are pairs of waypoints and intervals over the unknown process value at those waypoints;
- $\overline{s^{\mathcal{D}}} = (\overline{v}, I[\omega(\overline{v})])$, where $\overline{v} \in V$ is the initial waypoint of the robot;
- The transition function is defined as

$$T^{\mathcal{D}}((v, I), (v, v_g), (v', I')) = T^O((v, I), (v, v_g), v') P^{\mathcal{GP}}(f(v') \in I' \mid \mathcal{D}), \quad (9)$$

where $T^O\big((v, I), (v, v_g), v'\big)$ is the average of $T^O((v, o), (v, v_g), v')$ along interval $I = [l, u]$:

$$T^O((v, I), (v, v_g), v') = \frac{\int_l^u T^O((v, o), (v, v_g), v')\, do}{u - l}. \quad (10)$$

The EstMDP defines a transition function for which we can plan for using standard techniques. It does so by (i) defining the transition function over intervals of O rather than O itself, by averaging it along each interval; and (ii) completing the transition definition, by weighting the transition probabilities in T^O the GP estimate with the GP posterior over each

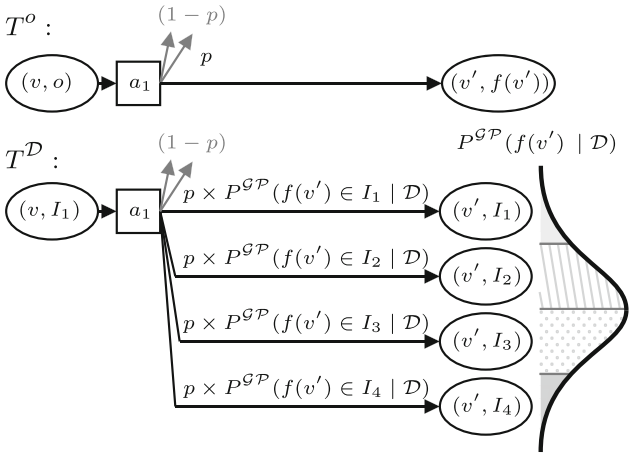


Fig. 1 Construction of the estimated MDP transition function $T^{\mathcal{D}}$, combining the PKSMDP probabilistic transition function T^O and GP predictive posterior given the dataset

possible interval, given the set of observed data $\mathcal{D}$. Figure 1 depicts an example $T^{\mathcal{D}}$.

The EstMDP is at the core of the exploration algorithms presented in the next two sections. To reason about safety on the EstMDP, we define the set of unsafe EstMDP states.

Definition 12 Let $\mathcal{M}^{\mathcal{D}}$ be an EstMDP. The set of unsafe states of $\mathcal{M}^{\mathcal{D}}$ is defined as

$$S^{unsafe} = \{(v, I) \in S^{\mathcal{D}} \mid \exists o \in I \text{ s.t. } \phi((v, o)) = 0\}, \quad (11)$$

and the set of safe states of $\mathcal{M}^{\mathcal{D}}$ is defined as

$$S^{safe} = S^{\mathcal{D}} \setminus S^{unsafe}. \quad (12)$$

States (s, I) are considered unsafe if they are unsafe for any $o \in I$. This is an overestimate of the set of unsafe states, which in turns lead to a conservative definition of safe states. For convenience, we also define the set of safe intervals for a specific waypoint v:

$$\mathcal{I}^{safe}(v) = \{I \in \mathcal{I} \mid (v, I) \in S^{safe}\}. \quad (13)$$

6 SafeEstMDP-Process: modelling an unknown hazard

In this section, we present our approach to Problem 2, i.e., when we assume a known underlying navigation graph V and our goal is to predict f as well as possible. Given that we use a GP to estimate f, we stop exploration when the predictive variance of the GP is within a bound for all waypoints in V that can be reached safely.

6.1 Algorithm description

The flow diagram (Fig. 2) provides a high-level description of the exploration approach. At each exploration step, the robot uses its current knowledge of the PKSMDP to determine which *goal* state it should next visit in order to best improve its knowledge of other uncertain states. It uses the EstMDP to represent its current knowledge of the PKSMDP and makes decisions based on this estimation.

While executing the current policy π that has been chosen to reach the current goal state, the robot carries out a *policy safety check* at each state. Carrying out the safety check allows the robot to replan when the probability of safely reaching the goal falls below the threshold $p_{\min}$. This check is particularly key when the transition function of the PKSMDP depends on the value of the unknown process, as the policy may entirely fail to reach the goal state if the values of the unknown process are not as the algorithm expected at goal selection time.

Constructing the EstMDP always uses the most up-to-date GP posterior, which is maintained within the GP manager module, and the PKSMDP given as input. In the next section, we will discuss how we can also build an PKSMDP incrementally, based on occupancy data sensed by the robot.

Algorithm 1 SAFE EXPLORATION (*SafeEstMDP-Process*)

Input: PKSMDP $\mathcal{M}^o$, safety function ϕ, safety probability threshold $p_{\min}$, kernel $k(x, x')$, initial location $\overline{v} \in V$
Output: GP posterior over unknown process

1: $v \leftarrow \overline{v}$
2: $v_g \leftarrow \overline{v}$
3: *visited* $\leftarrow \emptyset$
4: $\mathcal{D} \leftarrow \emptyset$
5: **while** $v_g \neq nil$ **do**
6: **if** $v \notin$ *visited* **then**
7: *visited* $\leftarrow$ *visited* $\cup \{v\}$
8: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(v, \omega(v))\}$
9: Rebuild $\mathcal{M}^{\mathcal{D}}$ (Definition 11) to consider current $\mathcal{D}$
10: **end if**
11: **if** $v = v_g$ **or** $Pr^{\pi}_{\mathcal{M}^{\mathcal{D}}, (v, I[\omega(v)])}\left(reach_{\neg S^{unsafe}, \{(v_g, I) \mid (v_g, I) \in S^{safe}\}}\right) < p_{\min}$ **then**
12: $v_g, \pi \leftarrow$ CHOOSEGOAL (Algorithm 2, Sect. 6.2)
13: **end if**
14: $(v, I[\omega(v)]) \leftarrow$ execute $\pi((v, I[\omega(v)]))$ and observe outcome
15: **end while**
16: No new goal state identified, end exploration

We provide further detail on the approach in Algorithm 1. The algorithm receives a PKSMDP $\mathcal{M}^o$ to be explored, a safety function ϕ, a minimum safety probability $p_{\min}$, a GP kernel k and an initial location $\overline{v}$.

The exploration algorithm repeatedly finds new goal states to explore, until no more goals are available (line 5). The algorithm maintains a set of waypoints *visited* that have already been explored (and must therefore be safe) and a dataset $\mathcal{D}$

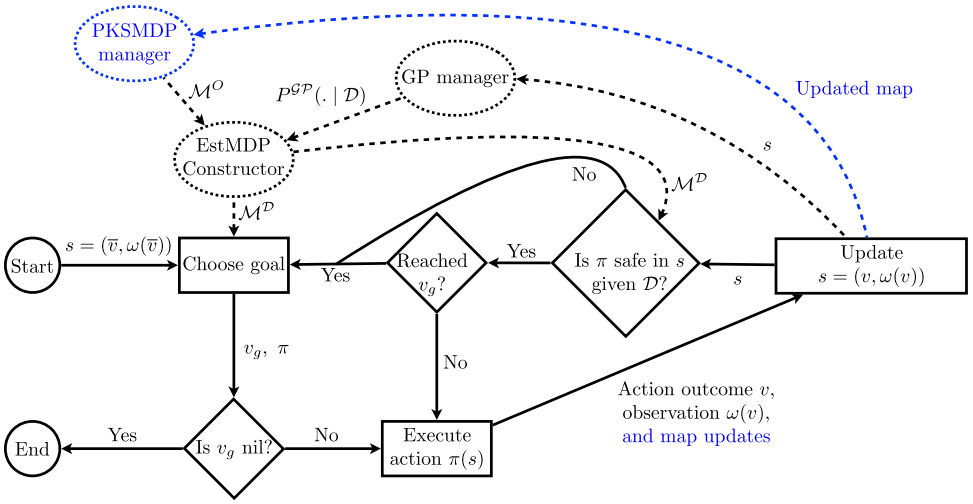


Fig. 2 Flow diagram of the exploration method. Full lines represent main algorithm loop, and dashed lines represent structures responsible for data gathered during deployment and construction of the EstMDP given that data. Parts in blue are specific to the approach presented in Sect. 7 (Color figure online)

Algorithm 2 CHOOSEGOAL

Input: EstMDP $\mathcal{M}^{\mathcal{D}}$, visited waypoints *visited*, current state $s = (v, I[\omega(v)])$, minimum safe probability $p_{\min}$, accuracy threshold η
Output: New goal waypoint v_g and corresponding policy π

1: $V' \leftarrow$ GETCANDIDATES (Algorithm 3)
2: **while** $V' \neq \emptyset$ **do**
3: $V_N \leftarrow$ set of the first N elements v of V'
4: $V' \leftarrow V' \setminus V_N$
5: **for** $v' \in V_N$ **do**
6: $p_{reach}(s, v') \leftarrow P^{\max}_{\mathcal{M}^{\mathcal{D}},s}\left(reach_{\neg S^{unsafe},\{(v',I)|(v',I)\in S^{safe}\}}\right)$
7: $p_{return}(v') \leftarrow \sum_{I\in\mathcal{I}^{safe}(v')} P^{\max}_{\mathcal{M}^{\mathcal{D}},(v',I)}\left(reach_{\neg S^{unsafe},\{(v'',I[\omega(v'')])|v''\in visited\}}\right) \cdot \frac{P(f(v')\in I|\mathcal{D})}{\sum_{I'\in\mathcal{I}^{safe}(v')} P(f(v')\in I'|\mathcal{D})}$
8: **if** $p_{reach} < p_{\min}$ or $p_{return} < p_{\min}$ **then**
9: Remove v_{cand} from V_N
10: **end if**
11: **end for**
12: **if** $V_N \neq \emptyset$ **then**
13: $v_g \leftarrow \arg\max_{v'\in V_N} \text{score}^{\mathcal{D}}(s, v')$
14: **return** v_g, and corresponding policy $\pi_{\neg S^{unsafe},\{(v_g,I)|(v_g,I)\in S^{safe}\}}$
15: **end if**
16: **end while**
17: **return** *nil*

of observations of the unknown process at the explored waypoints. These are updated whenever the robot visits a new state (lines 7–8).

Before executing an action from the current policy, the robot checks whether the policy can continue being executed safely from the current state. It does this by evaluating the same constrained reachability problem that was used to select the current goal state, but now from the current state. Evaluating π in the policy safety check is significantly less computationally demanding than generating π as part of CHOOSEGOAL. When the policy safety probability falls below a user-defined threshold $p_{\min}$ or the robot reaches its current exploration goal (line 11), then a new goal state must be selected. Where the PKSMDP transition and/or cost function are dependent on unknown value state features, one may also check the expected cumulative cost of continuing to execute π, and abandon the current goal if the cost grows significantly beyond the expected cost calculated at goal selection time.

The CHOOSEGOAL routine and the policy safety check make use of the EstMDP $\mathcal{M}^{\mathcal{D}}$, obtained from $\mathcal{M}^o$ and the GP posterior given the dataset $\mathcal{D}$, as explained in Definition 11. The robot uses $\mathcal{M}^{\mathcal{D}}$ to choose a new waypoint to be explored and computes the corresponding policy, according to CHOOSEGOAL (line 12, Algorithm 2 to be explained next). Finally, in line 14, the robot executes the action prescribed by its current policy, observes the outcome location, senses the value of the unknown process at that location, and updates the new state accordingly.

6.2 Choice of goal waypoint

The choice of goal waypoint makes use of the EstMDP model, and is detailed by Algorithm 2. A good goal way-

point should provide information when observed (i.e. have a high predictive variance before observation) relative to the cost taken to reach it and, crucially, be safe to reach and return from.

Algorithm 3 GETCANDIDATES

Input: EstMDP $\mathcal{M}^{\mathcal{D}}$, visited waypoints *visited*, minimum safe probability $p_{\min}$, accuracy threshold η,
Output: Candidate goal waypoint ordered sequence V'
1: $V^{safe} \leftarrow \{v' \in V \setminus visited \mid P(I[f(v')] \in \mathcal{I}^{safe}(v') \mid \mathcal{D}) > p_{\min}\}$
2: **return** $\{v' \in V^{\text{safe}} \mid Var(f(v') \mid \mathcal{D}) \geq \eta\}$, ordered by decreasing value of $Var(f(v') \mid \mathcal{D})$

In line 1 the set of candidate goal waypoints is generated by GETCANDIDATES, as defined in Algorithm 3. This set consists of unvisited waypoints that (i) are considered safe with high probability and (ii) have a GP predictive variance greater than the accuracy threshold η. We then enter a loop where we analyse batches of N waypoints (lines 3 and 4).

Because Algorithm 3 orders candidates in V' by highest predictive variance, this will be the N waypoints in V' where the GP is less accurate. Each of these waypoints v_g is checked for *reachability* and *returnability* probabilities. Specifically, the reachability probability (line 6) is defined as the maximum probability of safely reaching the set of states in $\mathcal{M}^{\mathcal{D}}$ corresponding to v_g from the current state s. The returnability probability (line 7) is computed as the weighted average of the probabilities of safely returning from one of the states in $\mathcal{M}^{\mathcal{D}}$ corresponding to v_g to a state corresponding to an already visited (hence, safe) waypoint. For the returnability check it is assumed that the initial state is safe, hence the normalisation at the end of line 7. If either of these probabilities is below the safety threshold $p_{\min}$, they are removed from the current batch of candidates (lines 8 and 9). Then, if there are still waypoints in the current batch of candidates, the remaining waypoint with the highest score (we propose a scoring function in the next section) is returned as the new goal waypoint (lines 12–14). The policy returned is the policy generated from the reachability check for the chosen goal waypoint. If the current batch of candidates is empty, the process is repeated for the next N highest variance waypoints. If no waypoint passing the reachability and returnability checks is found, then all waypoints that could be safely explored have been visited, and the algorithm returns nil, finishing the exploration (line 17).

Goal scoring function. The goal scoring function $\text{score}^{\mathcal{D}} : S^{\mathcal{D}} \times V \to \mathbb{R}$ is designed to indicate how beneficial a waypoint v would be to visit and observe from current state s, given the current observed dataset $\mathcal{D}$. This score should take into account the GP's predictive variance at v, the optimal expected cost to safely reach a state in $\{(v, I) \mid (v, I) \in S^{safe}\}$ from current state s, and the reachability/returnability probabilities for v. We propose the following scoring function:

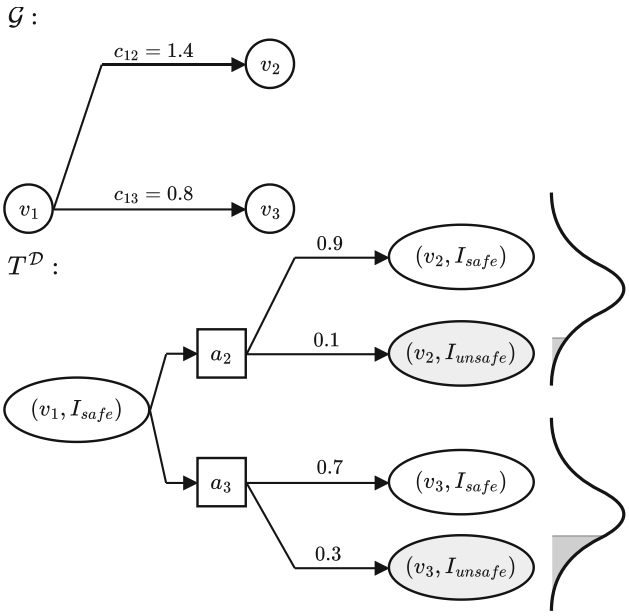


Fig. 3 An example of a navigation graph $\mathcal{G}$ and corresponding Estimated MDP transition function $T^{\mathcal{D}}$, defined based on an interval set $\mathcal{I} = \{I_{\text{safe}}, I_{\text{unsafe}}\}$. Numerical values are derived from the GP posterior given $\mathcal{D}$. Navigation graph edges are annotated with costs, transition function edges with transition probabilities

$$\begin{aligned}\text{score}^{\mathcal{D}}(s, v) = \\ & Var(f(v) \mid \mathcal{D}) \times \\ & E^{*}_{\mathcal{M}^{\mathcal{D}}, s}\left(cumul_{\neg S^{unsafe}, \{(v,I) \mid (v,I) \in S^{safe}\}}\right)^{-\gamma_1} \times \\ & (p_{reach}(s, v) p_{return}(v) - p_{\min}^2)^{\gamma_2}, \end{aligned} \tag{14}$$

where the parameters γ_1 and γ_2 provide relative weightings on different parameters.

Figure 3 shows an example EstMDP that might be used by SafeEstMDP-Process. In this scenario, starting at v_1 and supposing a safety threshold of $p_{\min} = 0.5$, the selection of the next goal state (between either v_2 or v_3) would weigh up several factors: the cost to reach v_2 is higher, but taking action v_3 comes with a significantly higher risk of ending up in an unsafe state according to the GP. Given that the GP variances are also similar, v_2 would likely be selected as the next goal node, whereas v_3 would only be selected if its variance were significantly higher.

7 SafeEstMDP-Map: exploring an unknown environment

In this section, we present our approach to Problem 3, i.e., when there is no assumption over the underlying map and

the goal is to safely explore a bounded environment X represented by a grid map $\mathbb{M}$, and classify, as free or occupied, as many cells in $\mathbb{M}$ as possible. For this problem, we will incrementally build the navigation MDP $\mathcal{M}^O$ (Definition 7) and stop exploration when the volumetric gain associated to all safely reachable waypoints in V is less than a specified threshold.

7.1 Algorithm description

There are two main differences with regards to the approach presented in the previous section: (i) there is a navigation graph construction process running in parallel with Algorithm 1. Instead of receiving a complete PKSMDP a priori, Algorithm 1 constructs an updated $\mathcal{M}^O$ whenever it needs to build an estimated MDP $\mathcal{M}^{\mathcal{D}}$, making use of the most up-to-date navigation graph representation of the currently explored environment; and (ii) the candidate waypoint selection, scoring function and stopping condition of ChooseGoal (Algorithm 2) are updated for the new exploration objective of safely mapping the environment. The blue part of the diagram in Fig. 2 highlights this additional step.

7.2 Incremental navigation graph construction

Our exploration system requires a mechanism for incrementally growing $\mathcal{M}^O$ based on local sensor measurements. This mechanism assumes that the robot navigation is *deterministic*, i.e., the transition function is such that $T^O((v, o), (v, v'), v'') = 1$ if $v'' = v'$, and 0 otherwise. Furthermore, we consider the MDP cost to be based on the Euclidean distance, i.e., $C((v, o), (v, v')) = \|v, v'\|_2$.

The deterministic navigation assumption is used so we do not need to estimate the outcome distribution of traversing a specific edge from sensor data, which is outside the scope of this paper.

However, if we have a mechanism that can do so, then our approach can deal with stochastic navigation, as there is nothing in the exploration algorithm that assumes deterministic dynamics. A PKSMDP with a deterministic transition function and Euclidean distance cost can be fully defined by its underlying graph $\mathcal{G} = (V, A^O)$. In the remainder of this subsection, we describe how to incrementally build $\mathcal{G}$. Note that any other cost function that maps pairs of waypoints to a positive number can be considered; we use Euclidean distance for ease of presentation. Furthermore, whilst we consider ray casting in a grid map which is derived from lidar scans, any other approach to calculate volumetric gain could be used. The graph-extending process is described in Algorithm 4.

The algorithm casts rays outwards from the robot's current location v in all directions within a horizontal plane, in angular increments of δ. Along each ray, stepping outwards based on a lengthscale ℓ, a new query waypoint v_q is created (line 4). If v_q is too close to the nearest pre-existing waypoint v_c, then the nearby waypoint is used as the query waypoint instead (line 7). Then, the occupancy map is queried to check if there is a path through free space between the previous query waypoint v_{prev} and v_q (line 9). If such a path exists, v_q is added to the vertices in the navigation graph, and a bidirectional edge is added linking it to the previous vertex. Otherwise, the current chain of waypoints ends, as it is presumed blocked by an obstacle. In this way, the algorithm constructs chains of waypoints outwards from the current waypoint v through free space, connecting them to existing nearby waypoints whenever possible.

Algorithm 4 ExtendGraph

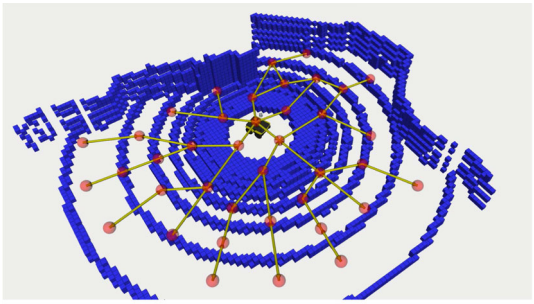
Input: Existing graph $\mathcal{G} = (V, A^O)$, current waypoint v, angle increment δ, range r, lengthscale ℓ, proximity factor f
Output: Updated graph $\mathcal{G}$

1: **for** $\theta \in [0, 2\pi]$ in increments of δ **do**
2: $v_{prev} \leftarrow v$
3: **for** $d \in [\ell, r]$ in increments of ℓ **do**
4: $v_q \leftarrow v_{prev} + d\mathbf{u}_\theta$ ▷ $\mathbf{u}_\theta$ is the unit vector in direction θ.
5: $v_c \leftarrow$ ClosestVertex(V, v_q)
6: **if** $\|v_q, v_c\|_2 \leq f\ell$ **then**
7: $v_q \leftarrow v_c$ ▷ Use existing vertex.
8: **end if**
9: **if** PathBetween(v_{prev}, v_q) **then**
10: $V \leftarrow V \cup \{v_q\}$
11: $A^O \leftarrow A^O \cup \{(v_{prev}, v_q), (v_q, v_{prev})\}$
12: **else**
13: **break**
14: **end if**
15: $v_{prev} \leftarrow v_q$
16: **end for**
17: **end for**
18: **return** $\mathcal{G} = (V, A^O)$

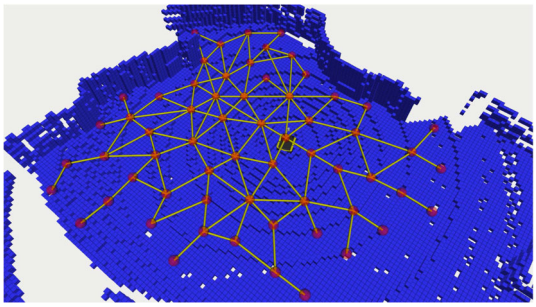
Figure 4 demonstrates the navigation graph construction algorithm in action, showing the graph itself as well as a 3D occupancy map obtained from a Clearpath Jackal robot. The initial navigation graph (Fig. 4a) has edges radiating a few metres outwards from the robot's initial location, in all directions except where the lidar's line of sight is obstructed. Figure 4b shows a more developed navigation graph after a few actions have been taken.

7.3 Choice of goal waypoint

The candidate goal selection routine in ChooseGoal (Algorithm 2, line 1) now calls the SafeEstMDP-Map version of GetCandidates (Algorithm 5). The *primary* candidate set of states (line 2) is similar to the one returned by Algorithm 3. The only two differences are: (i) replacing the GP predictive variance term with volumetric gain to reflect the information gain objective, and (ii) specifying a minimum volumetric gain $\eta_{\texttt{gain}}$ that a waypoint must have to be considered as a primary candidate.



(a) Initial navigation graph.



(b) Navigation graph after a few actions have been taken.

Fig. 4 Navigation graph construction on a Clearpath Jackal robot simulated in Gazebo. Red spheres represent waypoints, yellow lines traversal actions (Color figure online)

Algorithm 5 GETCANDIDATES (SafeEstMDP-Map)

Input: Visited waypoints *visited*, minimum safe probability $p_{\min}$, information gain threshold $\eta_{\texttt{gain}}$, current waypoint v
Output: Candidate goal waypoint ordered sequence V'
1: $V^{safe} \leftarrow \{v' \in V \setminus visited \mid P(I[f(v')] \in \mathcal{I}^{safe}(v') \mid \mathcal{D}) > p_{\min}\}$
2: $V^{prim} \leftarrow \{v' \in V^{safe} \mid \texttt{gain}(v') \geq \eta_{\texttt{gain}}\}$
3: $V^{prog} \leftarrow \{v' \in V^{safe} \setminus V^{prim} \mid \min_{v'' \in V^{prim}} \|v', v''\|_2 < \min_{\tilde{v} \in visited, v'' \in V^{prim}} \|\tilde{v}, v''\|_2\}$
4: **return** $V' = V^{prim} \cup V^{prog}$ as an ordered set, with elements of V^{prim} first, ordered by decreasing $\texttt{gain}(v')$, and elements of V^{prog} after, ordered by increasing $\|v, v'\|_2$

However, Algorithm 5 now also returns *progression candidates* (line 3), which are waypoints that progress the robot towards primary candidates. These are states for which $\texttt{gain}(v) < \eta_{\texttt{gain}}$, but visiting that state would decrease the minimum distance between the visited set of states and any primary candidate state. The addition of progression candidates is necessary because of the decoupling of information gain from the safety function. For most useful GP kernels, taking measurements increasingly closer to a primary candidate state will provide more information about that state's safety.

Figure 5 shows an example of this behaviour. The robot is at the start of a row of waypoints along a corridor, ending in a 90° corner. At the first timestep (Fig. 5a), the robot is at v_i. For this example, $p_{\min} = 0.99$. The waypoint v_c provides a view around the corner, so has a high information gain and is a primary candidate. However, as it is several navigation edges away from the robot, $p_{reach}(v_i, v_c) < p_{\min}$: the robot will not choose v_c as a goal state. The waypoint v_a is safely reachable, but has information gain below the minimum threshold so cannot be a primary candidate. However, v_a is a progression candidate because visiting it would decrease the minimum distance between the visited set of states and v_c. At timestep 2, the robot has chosen v_a as its new goal and transitioned to it. Having taken a measurement at v_a, it is now more certain that v_b and v_c are safely reachable. It can then continue to choose v_b, which is now safely reachable with probability greater than $p_{\min}$, as a new goal state and progress further towards v_c. Without the addition of progression candidates, the robot would be unable to move along the corridor in this manner.

Goal scoring function. Our scoring function is designed to encourage efficient exploration of the unknown space – a waypoint v is beneficial to observe if its volumetric gain is high, as this indicates it would likely be helpful in growing the robot's map of the environment. Thus, we replace the GP variance component in Eq. 14 with the information gain term $\texttt{gain}(v)$ defined in Eq. 6. This mirrors Eq. 14, except with the exploration term now driven by increase of information over the map rather than minimisation of GP variance:

$$\begin{aligned}\text{score}^{\mathcal{D}}(s, v) = \\ &\texttt{gain}(v) \times \\ &E^{*}_{\mathcal{M}^{\mathcal{D}}, s}(cumul_{\neg S^{unsafe}, \{(v', I) \mid (v', I) \in S^{safe}\}})^{-\gamma_1} \times \\ &(p_{reach}(s, v) p_{return}(v) - p_{\min}^2)^{\gamma_2}. \end{aligned} \tag{15}$$

Again, γ_1 and γ_2 specify the relative weightings of the components.

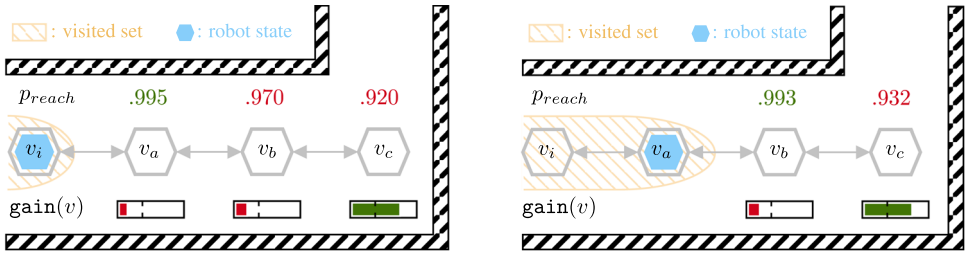
8 Experiments

We present a series of experiments in simulation to demonstrate the performance of the SafeEstMDP-Process and SafeEstMDP-Map algorithms.

For all experiments, the MDP is solved via PRISM (Kwiatkowska et al., 2011) using nested value iteration (Lacerda et al., 2019), and the GP framework used is GPy (GPy, 2012).

8.1 Evaluation domains

We use two sets of domains for evaluation of our algorithms in simulation. In the first set of domains (Sect. 8.1.1), the unknown process is nuclear radiation, and the transition dynamics are independent of the unknown process. The sec-



(a) Timestep 1. (b) Timestep 2.

Fig. 5 Robot progress towards high-information-gain states using progression candidates (Sect. 7.3)

ond set (Sect. 8.1.2) considers exploration in the presence of unknown ocean currents, which affect the transition dynamics.

8.1.1 Nuclear

We consider the safety hazard of gamma radiation exposure, which can be harmful at high levels to both robots and humans. Furthermore, standard gamma radiation survey sensors only provide noisy, local measurements. This matches our locally observable safety formulation.

We evaluate with the nuclear radiation hazard in two simulated domains, illustrated in Fig. 6.

1. *Reactor room*, Fig. 6a: a simulated world representing a 20 m × 20 m nuclear reactor room, from Wright et al. (2021).
2. *Research mine*, Fig. 6b: A 35 m × 28 m map built from 3D SLAM in a real underground mine section at Corsham, UK.

The evaluation of SafeEstMDP-Process in Sect. 8.2 requires a PKSMDP to be provided. For each domain, PKSMDPs are pre-generated using 8-connected grid maps with 1 m grid units, which can be seen in Fig. 6. In contrast, in Sect. 8.4 the PKSMDP is sequentially built over the map by SafeEstMDP-Map.

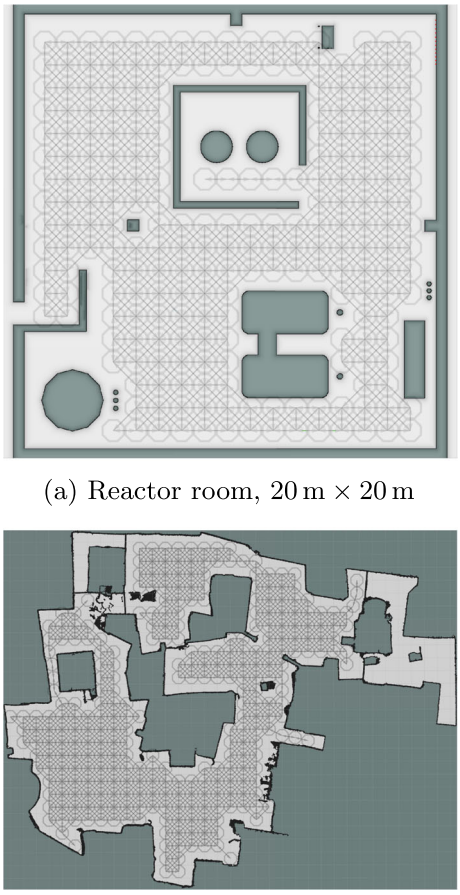
Simulation of radioactive environments.

Radiation is simulated using $1/r^2$ "solid angle" radiation physics (Wright et al., 2021). Radiation sources $\{(\chi_i^{src}, x_i^{src})\}_{i=0}^{n_{src}}$ have strength χ_i^{src} and pose x_i^{src}. Source strength is the exposure value at a distance of 1 m. The radiation exposure $\lambda(x)$ at robot pose $x \in X$ from these radiation sources is then:

$$\lambda(x) = \sum_{i=1}^{n_{src}} \frac{\chi_i^{src}}{\|x - x_i^{src}\|_2^2}. \tag{16}$$

For each exploration scenario, a PKSMDP (Sect. 8.2) or a 3D simulated environment (Sect. 8.4) is combined with a randomly generated distribution of radiation sources. Sampled radiation distributions are discarded when they result in trivial exploration results with too much of the environment being safe or unsafe. In this work we generate radiation distributions that result in a minimum of 40% of the environment's states being safely reachable from the initial state, and a maximum of 90%.

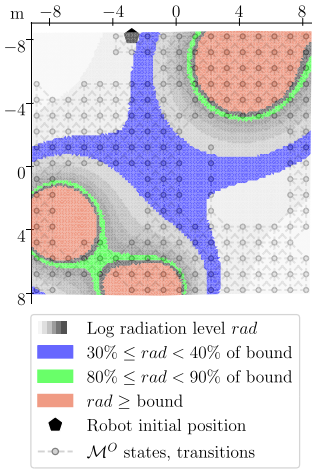
Random radiation fields are generated in both of the following ways, with each method contributing 50% of the evaluation environments:



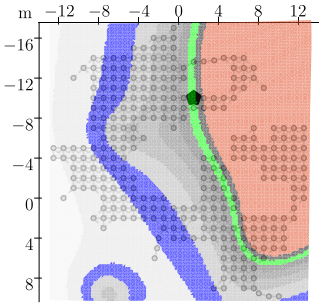
(a) Reactor room, 20 m × 20 m

(b) Research mine, 35 m × 28 m

Fig. 6 2D occupancy maps of both simulated experiment domains, with the navigation map used to build the PKSMDP for SafeEstMDP-Process shown in grey (Color figure online)



(a) An example of a random point-source (method 1) radiation distribution, in the reactor room domain. Three high-intensity point sources can be seen in the top right and bottom left corners. In the results plots in Figure 10, this scenario is represented by the dark green cross marker ✖.



(b) An example of a random Gaussian field (method 2) radiation distribution, in the research mine domain. In the results plots in Figure 10, this scenario is represented by the dark purple circle marker ●.

Fig. 7 Two valid random radiation distributions, generated according to Sect. 8.1.1. A table of all radiation distributions used in Sect. 8.2 can be found in Appendix A.4

1. Random point-source distribution: insert a randomly chosen number of sources ($5 \leq n_{src} \leq 30$), with uniformly random sampled poses across the environment's domain, with randomly sampled z position values $z \in \{1.0, 1.5, 2.5\}$ and randomly sampled strengths $\chi^{src} \in \{250, 500, 1000, 2000, 5000\}$. x and y position values are sampled uniformly within the bounds of the map ± 2 m.
   An example of this type of radiation distribution can be seen in Fig. 7a.
2. Randomly generated Gaussian random field distribution: evenly cover the map with radiation sources at $z = 1.0$ and draw their log strengths from a Gaussian random field.
   The Gaussian random field is generated with a radial basis function kernel, using uniformly sampled length-scale hyperparameter $l \in \{3.0, 5.0, 7.0\}$ and variance hyperparameter $\sigma \in \{20, 30, 50, 75\}$.
   An example of this type of radiation distribution can be seen in Fig. 7b.

GP modelling of radioactive environments.

To explore while ensuring safety with high probability, an exploration algorithm's GP model must be able to well-model the ground-truth hazard function. Previous works (Silveira et al., 2018; West et al., 2021; Khuwaileh & Metwally, 2020) have demonstrated GP regression as capable of accurately modelling radiation fields in the real world. Silveira et al. (2018), West et al. (2021) log transforms the radiation level measurement data, in a similar manner to the log-warped GP model we describe below. Several features of radiation fields present challenges for standard GP regression. Firstly, the values of the radiation intensity function over 3D space may vary over several orders of magnitude. This is particularly evident when high-intensity radiation sources are present. Furthermore, since standard GP predictive Gaussian posterior distributions are unbounded, the GP will assign some probability to a radiation level < 0. This is clearly nonphysical, as the ground-truth radiation level is non-negative.

Secondly, the standard GP assumption of zero-mean Gaussian observation noise with input-independent standard deviation does not hold. At an abstract level, radiation measurements are based on counts of discrete detection events: for example, counts produced by a Geiger-Muller tube. Statistically these are samples from a Poisson distribution with rate parameter λ being the ground-truth radiation level. Samples from this distribution have standard deviation $\sqrt{\lambda}$. Therefore, as the radiation level increases, the absolute measurement noise increases and the relative measurement noise decreases. Due to this effect and the non-linear behaviour of the Geiger-Muller tube, real-world radiation sensors are generally specified with a percentage rather than absolute reading accuracy (Knoll, 2010).

GPs with a non-Gaussian or heteroscedastic observation function are non-conjugate and therefore cannot be solved in closed form. Computationally intensive methods, including Markov Chain Monte Carlo sampling and Laplace approximation, are required (Williams & Barber, 1998).

To alleviate these issues, we follow the approach of Snelson et al. (2003) and use a log-warped Gaussian process regression. In this formulation, we model the log of the radiation level value $f(\log(rad))$ with a GP. This transforms a normally distributed percentage observation noise with standard deviation $\sigma_{\%}$ in the radiation level into a normally distributed observation noise with standard deviation σ, where:

$$\exp\left(\log(rad) + \sigma\right) = rad \cdot (1 + \sigma_{\%}),$$
$$\sigma = \log\left(1 + \sigma_{\%}\right). \quad (17)$$

The relative measurement noise formulation therefore better matches our sensor's percentage measurement noise. Furthermore, the log-warped GP is also significantly better able to handle order-of-magnitude variation in the radiation level compared to a standard GP regression.

EstMDP modelling of radiation levels.

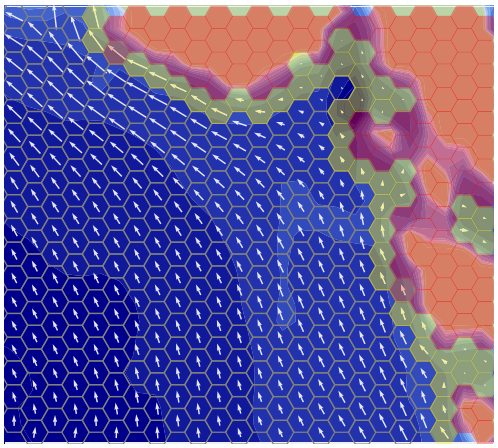
In this domain, the unknown process $f : V \to \mathbb{R}$ is the value of the radiation level at each state. As the radiation level does not affect waypoint transitions, we need only define two intervals $\mathcal{I} = \{I_{\text{safe}}, I_{\text{unsafe}}\}$ where $I_{\text{safe}} = [0, \log(\lambda^{\max})]$ and $I_{\text{unsafe}} = [\log(\lambda^{\max}), \infty)$ and $\lambda^{\max}$ is the radiation level safety bound. If a non-log GP were used, the EstMDP structure would be identical, with equivalent intervals $I_{\text{safe}} = [0, \lambda^{\max}]$ and $I_{\text{unsafe}} = [\lambda^{\max}, \infty)$. The exploration GP models the 1D radiation level at each state: $P^{\mathcal{GP}} : \mathbb{R}^2 \to \text{Dist}(\mathbb{R})$. The GP input is $\{x, y\}$, in units of metres in a local coordinate frame.

8.1.2 Unknown currents

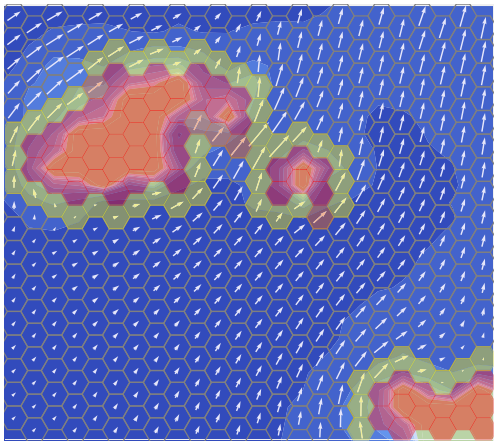
In this set of domains, the autonomous agent is an underwater autonomous vehicle (AUV) exploring the open ocean. The unknown process is the underwater current, which probabilistically affects the AUV's transition dynamics. This is different from the setting of Sect. 8.1.1, where the PKSMDP transition function T^O is deterministic and independent of O.

The AUV travels in a given direction by diving to a fixed depth and then travelling at a fixed speed relative to the water surrounding it, surfacing after a fixed length of time. Depending on the direction and magnitude of the water currents, this can result in the AUV surfacing in different locations. We discretise the ocean into a grid of hexagonal cells, as shown in Fig. 8. The figure illustrates the three evaluation domains, *Faroe*, *Guernsey*, and *Norway*, each of which is a 43.2 km × 37.4 km area covered by a 24 × 18 hex grid of side length 1.2 km.

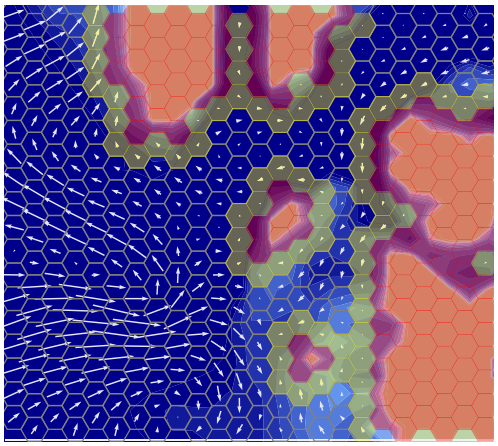
The AUV fails its safety specification when it enters a cell with water depth ≤ 10 m, as it is at risk of colliding with the seabed. These "*definitely unsafe*" states are known to be unsafe a priori, based on bathymetry data described below, and are shown in red in Fig. 8. The AUV also fails its safety specification when it enters a cell where the water current magnitude is too high in the direction of an a priori unsafe



(a) Faroe map, 61.7-62.1°N, 7.6-6.8°W

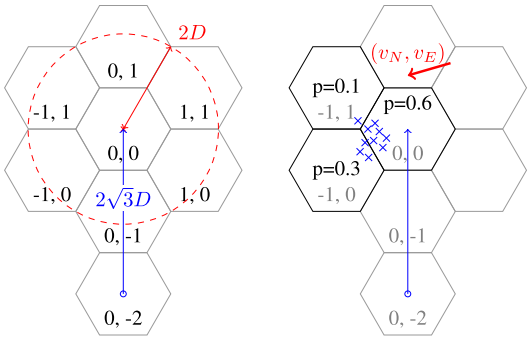


(b) Guernsey map, 49.2-49.6°N, 2.8-2.1°W



(c) Norway map, 58.9-59.3°N, 5.0-5.7°E

Fig. 8 AUV domain areas, showing ground-truth currents (arrows) and state safety (hex cell colour) (Color figure online)



(a) Possible outcomes of the action. The red circle shows the possible surfacing locations given a minimum AUV velocity.

(b) For a current velocity (v_N, v_E), 10 sampled simulation outcomes (blue crosses) estimate cell outcome probabilities.

Fig. 9 Illustration of possible and sampled outcomes of the AUV navigating from $(0, -2)$ to $(0, 0)$ in the hex grid

state. This would result in the AUV being carried into an unsafe state before it can dive again. States which may be unsafe depending on the current value at that state, "*maybe unsafe*" states, are shown in yellow in Fig. 8. Dark orange cells show "maybe unsafe" states which are in fact unsafe given the ground-truth current value at that state. This is an example of a more complex safety specification, which is defined on both known and unknown state feature values.

Actions are to attempt to travel 2 cells in one of the 6 hexagonal directions, with the AUV surfacing at the end of the action. An action in the north direction from cell $(0, -2)$ to cell $(0, 0)$ is illustrated by the blue arrow in Fig. 9. To limit the number of possible action outcomes, we wish to ensure that the AUV can only surface in the target cell or one of its immediate neighbours. We do this by calculating the minimum required AUV velocity for the maximum current magnitude in the dataset.

As illustrated in Fig. 9a, the distance from an initial cell to an action's target cell is $2\sqrt{3}D$ where D is the hex cell side length. The time taken for the AUV to travel this distance is $2\sqrt{3}D/v_{\text{AUV}}$ seconds. The maximum distance that the AUV can be carried by the current in this time is $v_{\max} \times 2\sqrt{3}D/v_{\text{AUV}}$, where $v_{\max}$ is the maximum current velocity. If this distance is less than $2D$, then the AUV can only surface in the target cell or one of its immediate neighbours. The minimum AUV velocity to ensure this is therefore $v_{\text{AUV}} > \frac{1}{\sqrt{3}v_{\max}}$.

We assume that throughout the dataset the maximum current velocity in each axis is $0.65\,\text{m}\,\text{s}^{-1}$. We select $v_{\text{AUV}} = 1.8\,\text{m}\,\text{s}^{-1}$, as $1.8 > \frac{1}{\sqrt{3}\sqrt{2(0.65^2)}}$.

Action costs are the constant time taken to execute an action: $C((v, o), (v, v')) = 1$.

Simulation of unknown currents and underwater autonomous vehicle.

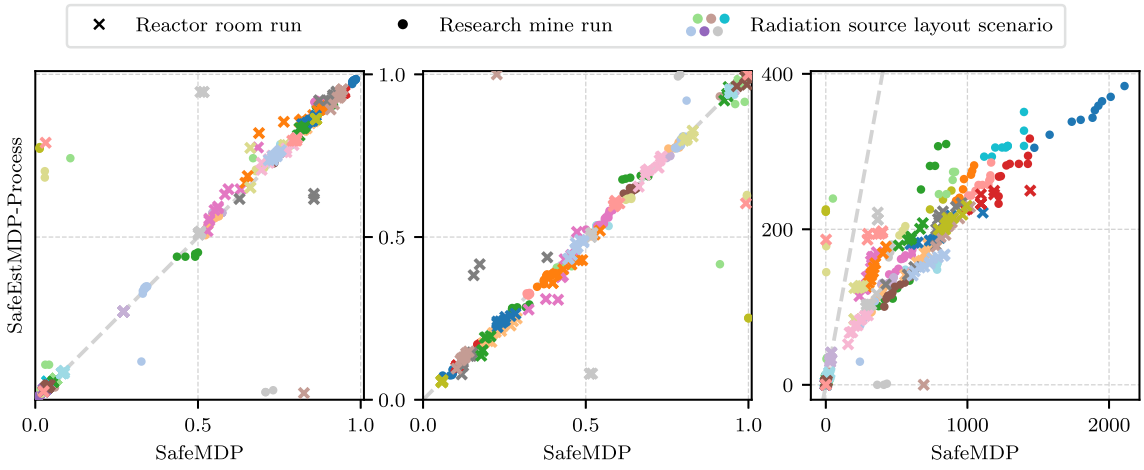
Ground-truth bathymetry and water current velocity data is taken from the NORTHWEST_ SHELF_ANALYSIS_ FORECAST_PHY_004_013 dataset[1] , which covers the northwest European shelf area. A kinematic AUV simulator (Budd et al., 2022) is used to forward simulate vehicle control, actuation and stochastic disturbances in a Monte Carlo manner, with a 1 Hz simulation frequency. The simulated AUV is a small, moving-mass propellor-driven vehicle with neutral buoyancy and noisy actuation. Water currents acting on the simulated vehicle are linearly interpolated from the ground-truth dataset points. The "true" outcome of a navigation action is sampled by running this simulator with ground-truth current values, as illustrated in Fig. 9b. This results in an implicit "ground-truth" MDP with probabilistic action outcomes.

EstMDP modelling of unknown current navigation.

In this domain, the unknown process $f : V \to \mathbb{R}^2$ is the 2D vector of water current velocities at each state. Intervals $\mathcal{I}$ therefore represent a range of values for the north and east current velocity components v_N and v_E. We define the intervals as the cartesian product of two 1D intervals, one for each component of the 2D vector, i.e. $\mathcal{I} = \mathcal{I}_N \times \mathcal{I}_E$ where $\mathcal{I}_N = \mathcal{I}_E$ for simplicity. We assume that the AUV noisily measures current velocity at states it passes through, using an onboard sensor or by analysing its end position after surfacing. The exploration GP is a coregionalised GP which models the 2D vector (v_E, v_N) of east/north current velocity at each state: $P^{\mathcal{GP}} : \mathbb{R}^2 \to \text{Dist}(\mathbb{R}^2)$. The GP input is $\{x, y\}$, in units of metres in a local coordinate frame.

To apply SafeEstMDP-Process to the AUV domain, we must define the interval-dependent transition function $T^O((v, I), (v, v_g), v')$. We do not have a closed-form expression for $T^O((v, o), (v, v_g), v')$ to use with Eq. 10: we can only sample navigation outcomes from the AUV simulator. We therefore build a *transition kernel* which estimates the probability of each possible outcome, relative to the cell the action was taken in, given the 2D interval over the current velocity at that start state. For a specific interval I, an evenly spaced grid of 10×10 current velocity values (v_N, v_E) are taken from the interval. We run 20 repeats of the AUV simulator for each of these values, and record the resulting surfacing cell. Figure 9b shows 10 sampled outcomes for a specific (v_N, v_E) value. This process gives us an estimate of the probability of each possible transition outcome given the interval I, and is independent of start state cell.

[1] UK Met Office Marine Data Service. Available: https://www.metoffice.gov.uk/services/data/met-office-marine-data-service.



(a) Safely explored proportion (b) Final KL divergence (c) Total cumulative cost

Fig. 10 Scatter plots comparing SafeEstMDP-Process, on the y-axis, to SafeMDP, on the x-axis. The two algorithms are compared on **a** the proportion of the ground-truth safely reachable set the algorithm is able to correctly mark as safe, **b** the final KL divergence from the algorithm's GP model to a "ground truth" GP model, and **c** the total cost the algorithm has incurred by the end of exploration. Total 400 runs of SafeEstMDP-Process and SafeMDP (2 domains, 20 randomly generated radiation layouts, 10 repeats). All runs terminated without breaking the safety specification. See Fig. 21 (appendix) for plots of all scenarios shown here

8.2 SafeEstMDP-Process: mapping an unknown hazard

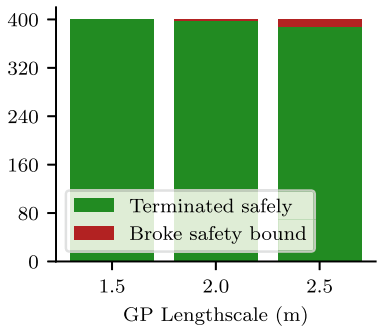
In this section, we compare SafeEstMDP-Process to SafeMDP (Turchetta et al., 2016), a prior MDP safe exploration algorithm which aims to determine the ground-truth maximum reachable safe set of states. To generate a navigation MDP and unknown process to explore, we generate 8-connected grid maps with side length 1 m in the reactor room and research mine domains, combined with randomly generated radiation distributions according to Sect. 8.1.1.

For all experiments $p_{\min} = 0.99$, the goal selection weights are $\gamma_1 = 1.0$ and $\gamma_2 = 0.8$, the batch evaluation size $N = 8$ and $\eta = 0.01$. Unless otherwise stated, the GP kernel was an RBF with variance 1.0 and lengthscale 2.0 m. We also ran these experiments with a Matern 3/2 kernel, producing qualitatively similar results. The GP observation noise $\sigma^2 = 0.0009$, corresponding to $\sigma_{\%} \approx 3\%$ (Eq. 17). The robot's initial position is randomly sampled from a set of 4 waypoints. The safety bound $\lambda^{\max} = 1000$.

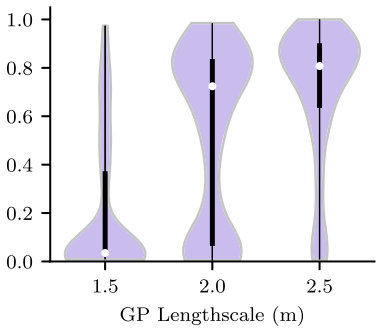
Figure 10 compares SafeEstMDP-Process and SafeMDP across 400 runs for each algorithm: for each of the 2 domains, 20 radiation layouts are randomly generated and used for 10 repeats. Figure 10a shows the proportion of the ground truth safely reachable state space that each algorithm has correctly determined as safe, at the point of termination (larger is better). Similarly, Fig. 10b shows the KL divergence at termination between each algorithm's GP model and a *ground-truth GP* trained on noiseless observations of every state in the PKSMDP (smaller is better). KL divergence is evaluated at all states in the PKSMDP, and is normalised relative to the KL divergence of the algorithm's initial GP at the start of execution. Figure 10c shows the cumulative cost incurred by the robot at termination time. Each figure includes a grey, dashed equal performance line. Note the different axis scaling in Fig. 10c, which we use to make visualisation easier. The need to do so highlights how SafeEstMDP-Process clearly outperforms SafeMDP in terms of cost. Finally, for a given marker on these scatter plots, an illustration of the corresponding exploration scenario can be found in Appendix A.4.

The symmetry and strong diagonal distributions of 10a and 10b illustrate that SafeEstMDP-Process and SafeMDP are equally capable of exploring the safely reachable state space and building an accurate model of the unknown process. The major difference is seen in 10c: SafeEstMDP-Process is capable of achieving the same results with far lower cost incurred. This is particularly true when exploring MDPs with much of the state space safely reachable, e.g. the dark blue circle markers ● at the top right of a and c, bottom left of b. In this scenario SafeEstMDP-Process incurs approximately 5× less cost than SafeMDP (note that the axes are not equally scaled).

The distribution of markers in these figures illustrates that, for both algorithms, it is harder to fully explore the safely reachable state space in some scenarios than in others. The dark blue circle marker ● scenario consistently terminates with close to 100% of the safe reachable state space explored and close to 0 final KL divergence. Conversely, the dark purple circle marker ● scenario is consistently barely explored,



(a) Number of SafeEstMDP-Process runs that finished exploring safely or that broke the safety bound during execution, for differing values of the GP kernel lengthscale.



(b) Proportion of the ground-truth safely reachable set that SafeEstMDP-Process is able to correctly mark as safe, for differing values of the GP kernel lengthscale.

Fig. 11 The effects of varying GP kernel lengthscale (x axis) on the behaviour of SafeEstMDP-Process. Total 400 runs per GP lengthscale value (2 domains, 20 randomly generated radiation layouts, 10 repeats)

with the explored proportion close to 0%. As can be seen in the plot of this scenario in Appendix A.4 Fig. 21b, the initial location for the robot immediately neighbours an unsafe area. This results in the robot being unable to choose a location to move to while ensuring safety with sufficient certainty.

One scenario which shows more complex behaviour is that represented by the dark green cross marker ✖. In this scenario, most runs terminate with close to 80% of the safe reachable area explored, but some terminate with less than 10%. This scenario is plotted in Fig. 7a, which shows that the robot initial location is at the start of a corridor leading to the rest of the map. To reach the rest of the map, the robot must travel towards a more dangerous area before rounding the corner. In some runs, the robot will sample higher-than-average noisy radiation level measurements. It may conclude that it cannot safely round the corner, and will terminate without exploring the rest of the map.

Overall, these results demonstrate SafeEstMDP-Process's improved cost efficiency over SafeMDP. SafeEstMDP-Process is able to choose goal states that are multiple steps away from the currently explored set, while SafeMDP will only choose goal states neighbouring the currently explored set. By evaluating the safety and cost of paths to goal states, SafeEstMDP-Process is able to trade off exploration information gain vs reachability cost, so is able to explore the state space more efficiently. Considering multi-step paths to goal states also offers the option of taking measurements only at goal points (rather than every visited state), which is useful if measurements are expensive or time-consuming.

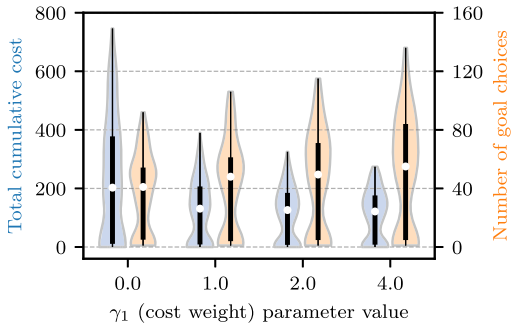
In Fig. 11 we analyse the effect of the GP kernel lengthscale hyperparameter on safe exploration with SafeEstMDP-Process. Figure 11a illustrates the number of runs that terminate safely compared to those that terminate by breaking the safety bound. Figure 11b shows distributions of the explored ground-truth safely reachable state space over many runs, this time depicted as a violin plot to facilitate comparison across the three tested lengthscale values.

It can be seen that a longer lengthscale value allows the exploration algorithm to reliably explore more of the ground-truth safely reachable state space, resulting in a higher concentration of runs with safely explored proportion close to 1.0 in 11b. With a lengthscale value of 1.5 m, almost all runs are unable to explore more than 20% of the safe state space.

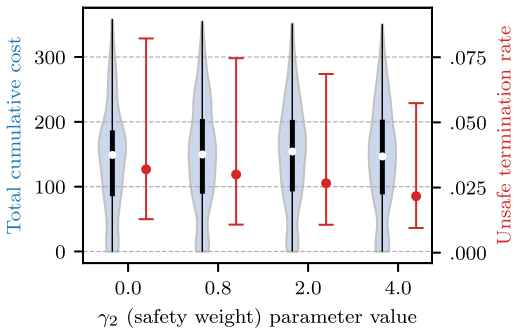
However, the safety guarantees provided by GP safe exploration algorithms are entirely dependent on accurate modelling of the hazard function with the GP. Despite $p_{\min}$'s value of 0.99, a lengthscale of 2.5 m does not model the radiation hazard function well enough, which yields overly aggressive robot behaviour. This results in the robot breaking the safety bound in 3.25% of the runs, compared to only 0.5% of runs for lengthscale 2.0 m.

In Appendix A.3, we show the results from repeating these experiments using a GP without the log-warping, which results in a significantly higher rate of safety bound violation, since the model less accurately captures the true behaviour of the radiation hazard. Note that, because $p_{\min}$ attempts to bound the safety of an *individual* goal choice in SafeEstMDP-Process, $p_{\min}$ is not directly comparable with success rates over exploration episodes with many goal choice steps.

We carry out analysis of the effect of the γ_1 and γ_2 goal choice hyperparameters (Eq. 14) on SafeEstMDP-Process in Fig. 12. Figure 12a shows that as more priority is placed on finding low-cost goal states by increasing γ_1, the final cumulative cost incurred by the robot decreases. This is expected, as the robot is more likely to choose goal states that are closer to the current state, and likely takes a more efficient path throughout execution. As well as decreasing the relative importance of the other goal candidate score components, increasing γ_1 also increases the number of goal choice steps



(a) Effect of varying the cost-to-candidate-goal weight γ_1 on the final cumulative cost incurred, and number of goal choice steps taken, for SafeEstMDP-Process.



(b) Effect of varying the safety-to-candidate-goal weight γ_2 on the final cumulative cost incurred, and the rate of safety bound violation, for SafeEstMDP-Process. Results from 600 runs.

Fig. 12 Ablation behaviour for SafeEstMDP-Process goal choice hyperparameters. Within violin plots, white dots show median values and thick black lines show interquartile ranges (Color figure online)

taken throughout exploration, as the robot is more likely to choose goal states that are closer to the current state. This will increase the computational cost of the algorithm.

Figure 12b shows a possible trend in safety violations as γ_2 increases. Confidence intervals are large as the proportion of safety violations is low, but the trend is consistent across the three tested values. Confidence intervals are produced by Jeffreys credible interval analysis using 600 runs across all radiation domain environments. The figure shows that the final cumulative cost incurred by the robot is largely insensitive to the value of γ_2. There was also no significant effect on the number of calls to the goal choice algorithm. This ablation used a 2.5 m lengthscale value to ensure some safety violations occurred.

8.3 SafeEstMDP-Process: exploration in the presence of unknown transition dynamics

In this section we demonstrate the ability of SafeEstMDP-Process to explore safely in the presence of unknown transition dynamics. Existing MDP safe exploration algorithms such as SafeMDP are not capable of reasoning about probabilistic and uncertain transition dynamics, so we have no baselines to compare SafeEstMDP-Process to.

Current velocity measurements have standard deviation $0.03\,\mathrm{ms}^{-1}$ in each axis. The GP kernel is an RBF with variance 0.2 and lengthscale 4 km, which were found to be suitable for the NWES dataset area. The GP likelihood noise was fixed to $\sigma^2 = 0.0009$. We define current value intervals $\mathcal{I}_N = \mathcal{I}_E = \{(-0.65, -0.25), (-0.25, 0.25), (0.25, 0.65)\}$, resulting in 9 intervals. For all AUV experiments $p_{\min} = 0.95$, the goal selection weights are $\gamma_1 = 1.0$ and $\gamma_2 = 0.8$, the batch evaluation size $N = 3$ and $\eta = 0.01$. "Maybe unsafe" states are unsafe if $v_E > 0.25$ and the state to the east is "definitely unsafe", and similarly for $v_E < 0.25$ and a "definitely unsafe" state to the west. "Maybe unsafe" states are also unsafe if $v_E > 0.25$ and $v_N > 0.25$ and the state to the north-east is "definitely unsafe", and similarly for the other 3 diagonal directions in the hex grid.

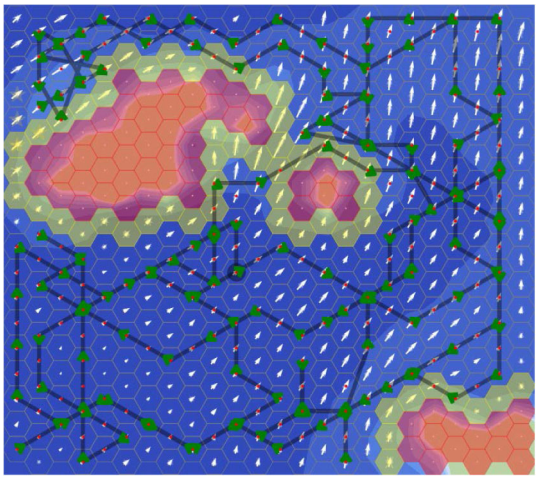
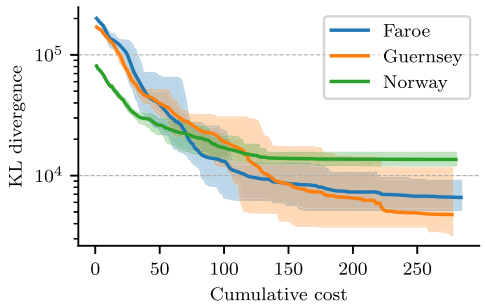


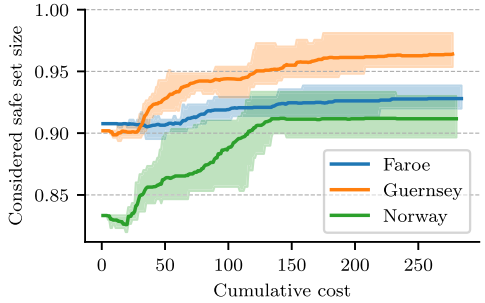
Fig. 13 An illustrative execution history of SafeEstMDP-Process in the AUV domain. White arrows show the final GP model. Dive locations are marked by green arrows. See Fig. 8b for the ground-truth current field and safe set (Color figure online)

SafeEstMDP-Process was evaluated with 10 runs in each of the Faroe, Guernsey, and Norway maps. SafeEstMDP-Process did not visit an unsafe state in any of the 30 runs. Figure 13 shows an illustrative execution history of SafeEstMDP-Process in the Guernsey map, and Fig. 14 shows quantitative results.

Figure 14a illustrates the evolution in GP fit quality as the AUV incurs cost while exploring. GP fit quality is measured by Kullback–Leibler (KL) divergence from the exploration GP to a *ground-truth* GP trained on noiseless observations of the ground-truth current value at every hex cell. As the AUV



(a) KL divergence from exploration GP to a ground-truth GP using noiseless ground truth data.



(b) The size of the set of states the algorithm considers to be safe with probability $> p_{\min}$. Values are normalised by the ground-truth safe set size.

Fig. 14 Exploration progress vs cumulative cost (time taken). For each map, the solid line shows mean value and shaded area shows the range of values across 10 repeats (Color figure online)

explores and incurs more cost, the GP model becomes more accurate and the KL divergence therefore decreases.

Similarly, Fig. 14b shows the algorithm's progress in identifying the ground-truth safe set of states. The algorithm starts off immediately considering a large proportion of the state space to be safe. This is because a majority of states ($\sim$ 83% of states for Norway and $\sim$ 90% for Faroe and Guernsey) do not neighbour an unsafe state so are therefore always safe. As the AUV explores, it becomes more confident in the (un)safety of "maybe unsafe" states, and the proportion of the state space considered safe increases.

SafeEstMDP-Process is able to identify most of the ground-truth safe set of states and build an accurate model of the unknown process in all three maps. The hardest map to explore is Norway, where the best runs were able to identify $\sim$ 93% of the ground-truth safe set. This is because no runs successfully reached the top right corner of the map, which requires passing through a narrow channel with unknown state safety either side.

8.4 SafeEstMDP-Map: exploring unknown environments

We evaluate the SafeEstMDP-Map algorithm in the two nuclear domains shown in Fig. 6, this time in a full physics simulation in Gazebo (Koenig & Howard, 2004). The algorithm is evaluated in each environment with three radiation layouts generated randomly as described in Sect. 8.1.1.

For this set of experiments, and those in the following section, we use $p_{\min} = 0.99$, goal selection weights $\gamma_1 = 1.5$ and $\gamma_2 = 0.8$, batch evaluation size $N = 30$, gain threshold $\eta_{\texttt{gain}} = 0.1$, and volumetric gain parameters $\alpha_{unk} = 20.0$, $\alpha_{free} = 1.0$, $\alpha_{occ} = 0.0$. The GP kernel is an RBF with variance 1.0 and lengthscale 2.0 m. The GP observation noise $\sigma^2 = 0.0009$, corresponding to $\sigma_{\%} \approx 3\%$ (Eq. 17). The robot's initial position is randomly sampled from a set of 4 waypoints.

We evaluate our algorithm against two baselines. The first is a threshold-based algorithm which we denote ThresholdBasedExplorer (TB), which selects exploration goals using the same scoring function as SafeEstMDP-Map, but does so without an explicit hazard model. Instead, the safety function ϕ for ThresholdBasedExplorer treats states as safe to visit simply if the hazard value is no more than some safety limit L, or unsafe otherwise. It then uses a threshold fraction $\varphi \in (0, 1)$ of the safety limit to prevent it from entering unsafe states. If the robot reaches a state at which the radiation level is more than φL on the way to its goal, it marks all adjacent *unvisited* states as unsafe, and selects a new goal from the remaining safe states, thereby turning back from the path to the previous goal. The second baseline, MeanPredictionExplorer (MP), again uses the same goal selection process as SafeEstMDP-Map, except with a limited version of the full EstMDP. Rather than weighting the EstMDP intervals based on the GP posterior at each state, this baseline instead assigns all probability mass to the posterior mean value at each state. This baseline therefore classifies states as safe or unsafe based purely on the GP predictive mean, with no consideration of predictive uncertainty. We evaluate SafeEstMDP-Map against MeanPredictionExplorer and ThresholdBasedExplorer with $\varphi = 0.3$ and $\varphi = 0.8$, simulating each configuration for 30 exploration runs.

The plots in Figs. 15 and 16 show that SafeEstMDP-Map is able to explore more effectively than the baselines across the variety of simulated configurations. The mean-prediction approach frequently results in unsafe termination, since without the full EstMDP model the safe reachability check is much less accurate, incorrectly classifying many

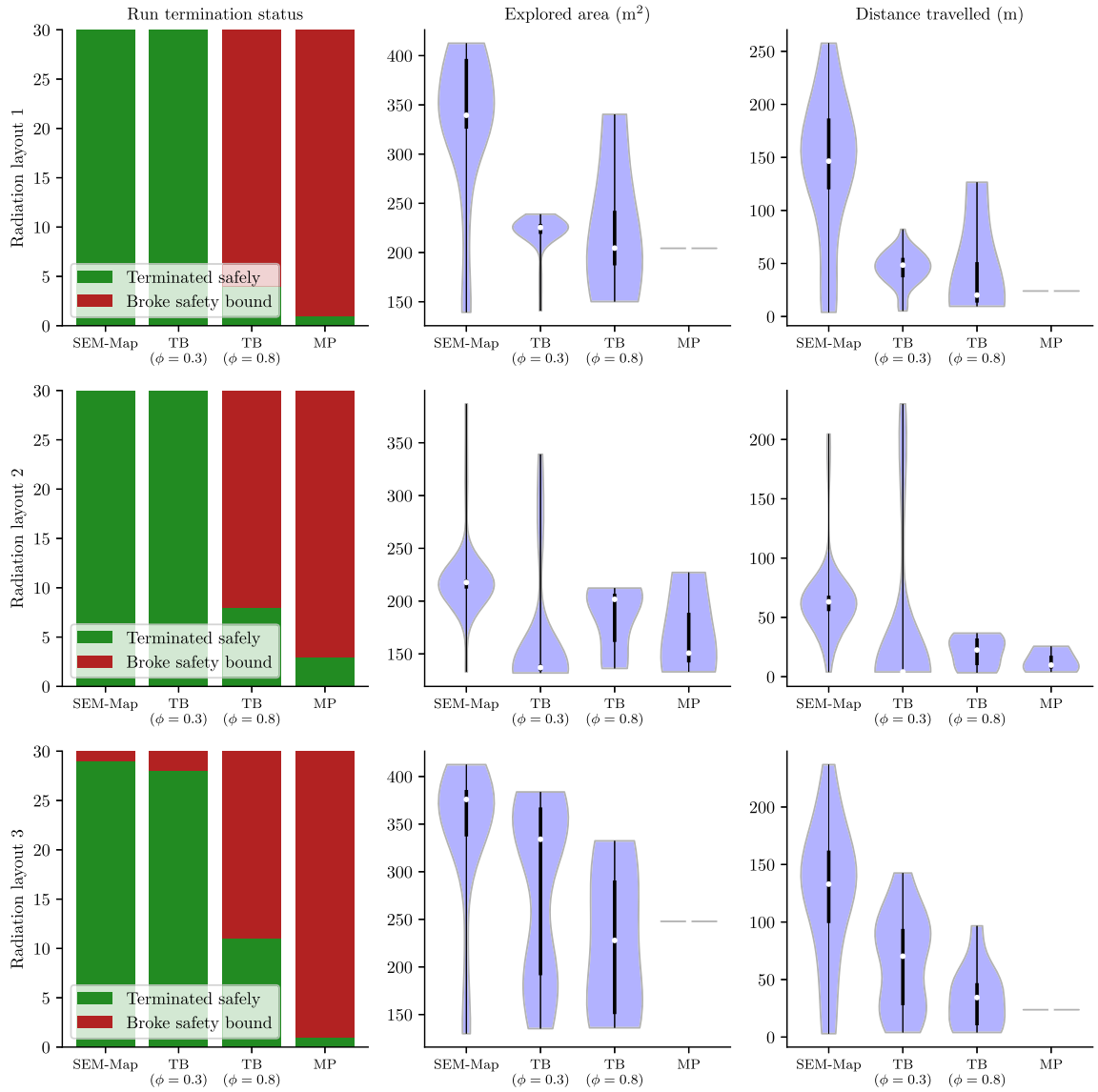


Fig. 15 Experiments for SafeEstMDP-Map in the reactor room domain. Each algorithm is tested on three randomly generated radiation layouts, with each configuration repeated 30 times. Left: number of runs (out of 30) terminated successfully: middle: violin plots summarising the statistics over area explored; right: violin plots summarising the statistics over of distance travelled (Color figure online)

unsafe states as safe. Of the threshold-based baselines, the more conservative one ($\varphi = 0.3$) is able to remain safe on almost every run. However, its exploration coverage on many of them is poorer than SafeEstMDP-Map's, as it will tend to turn back as soon as its radiation reading begins to increase, whereas SafeEstMDP-Map's radiation model allows it in some cases to continue forwards cautiously. On the other hand, the more aggressive baseline ($\varphi = 0.8$) explores more area, at the cost of far more runs terminating unsafely due to violating the safety specification.

SafeEstMDP-Map is able to use its GP model to explore more effectively, pushing forwards when an unexplored area is likely to be safe, and remaining cautious when it is believed unsafe. It also scales well with map size, as the EstMDP model is usually not much larger than the underlying navigation graph – even when the map is large, most waypoints will have been visited or be safe with probablility close to 1, so only the outer areas of the map will have significant uncertainty over radiation level. Consequently, the robot can evaluate dozens of goal choice waypoints per second during exploration, allowing for online planning with minimal delay, even in the research mine environment with maps of up to 150 waypoints. In some radiation layouts, SafeEstMDP-Map does still violate the safety constraint, although this is likely due to the GP model failing to accurately model the radiation field. This issue could be mitigated through further

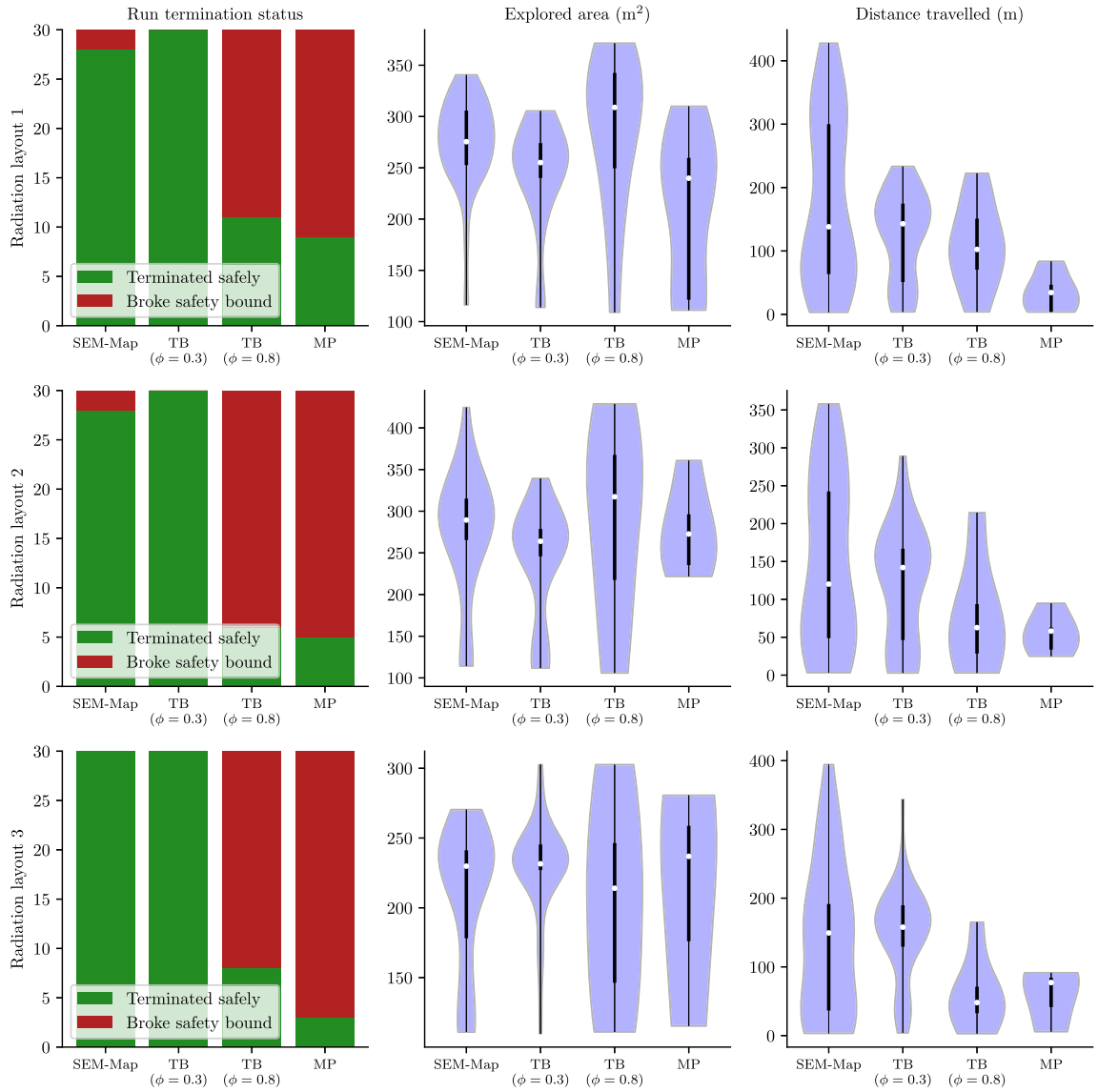


Fig. 16 Experiments for SafeEstMDP-Map in the research mine domain for different radiation layouts. Each algorithm is tested on three randomly generated radiation layouts, with each configuration repeated 30 times. Left: number of runs (out of 30) terminated successfully: middle: violin plots summarising the statistics over area explored; right: violin plots summarising the statistics over of distance travelled (Color figure online)

tuning or, more generally, applying domain knowledge in the design of the GP kernel.

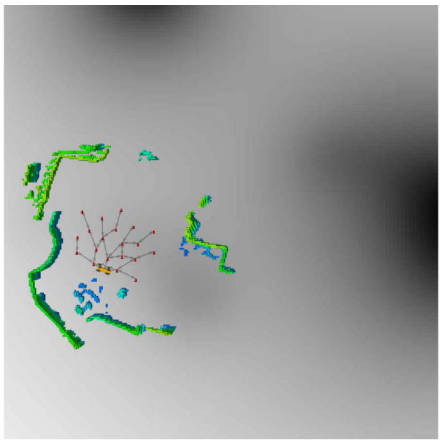
9 Deployment in a physical environment

We further validated our exploration algorithm by deploying it in a physical environment – the Corsham Research Mine – integrating it with a complete robotic hardware and software stack. Our experiments used simulated radiation sources, due to the health and safety difficulties of working with real radiation.

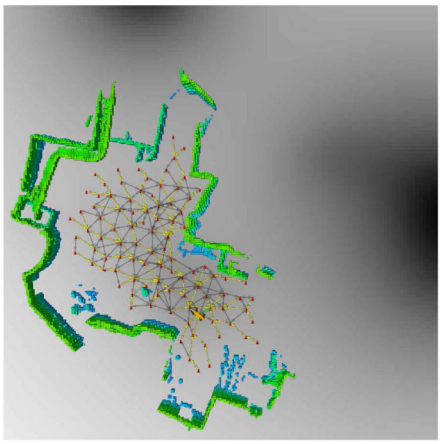
9.1 Hardware and system configuration

Our experiments were conducted on a Boston Dynamics Spot robot, shown with its sensor payload in Fig. 17a. The robot's on-board computer was an Intel NUC with an Intel i7-8559U processor, a 256 GB SSD and 16 GB of RAM. For perception, the robot was equipped with an Ouster OS0-64 3D lidar with a full 360° field of view around the robot. Localisation and mapping of the environment were provided by the VILENS SLAM system (Wisth et al., 2022), and interfaces between modules were through the robot operating system (ROS) Quigley et al. (2009).

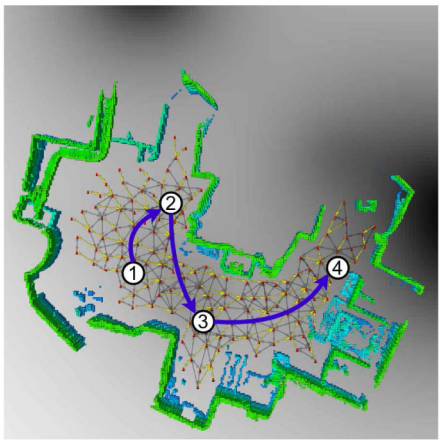
(a) Our Boston Dynamics Spot robot exploring Corsham Research Mine in Wiltshire, UK.



(b) The map and topological graph constructed by the SafeEstMDP-Map upon initialisation.



(c) The map and topological graph mid-way through the run.



(d) The final configuration and path followed by the robot during the run.

Fig. 17 Progression of a single run of SafeEstMDP-Map in the Corsham Research Mine. **b–d** Show a 3D occupancy map representing the robot's knowledge of the world around it (filtered for visual clarity), as well as a topological graph constructed online for planning and navigation. The grayscale background represents (ground truth) log radiation levels as a heatmap, with darker regions corresponding to higher levels

The radiation simulation system (Sect. 8.1.1) must know the ground truth position of the robot and the radiation sources. In the fully simulated experiments, the ground-truth robot position is readily available. However, there was no external ground-truth source (e.g. a motion-capture system) available in Corsham Research Mine. To provide ground-truth robot positioning, we therefore ran a separate instance of the VILENS SLAM system in localisation mode (i.e. using a pre-built full coverage 3D map of the environment). Independently from the ground-truth localisation system, an *online* VILENS instance provided SLAM for each experiment run. The navigation graph construction component (Algorithm 4) used the coordinate system and OctoMap established by the online SLAM system.

9.2 Deployment results

In order to illustrate the behaviour that can be obtained from the SafeEstMDP-Map in the real world, this section first steps through a successful run of the algorithm in the Corsham Research Mine. The output from this particular run is shown in Fig. 17. We then also comment on the exploration performance of the system across further deployments.

The robot started at the location shown in Fig. 17b, from which SafeEstMDP-Map constructed its initial topological map. The map at this point extended at most only a few metres from the robot's position, in directions in which line-of-sight was unobstructed.

This area of the map had relatively low and uniform radiation levels, so the variation in the SafeEstMDP-Map scoring function (Eq. 15) was dominated by the volumetric gain term.

The robot's first plan was therefore to move up towards Location 2 in Fig. 17d, since the nodes in this direction were slightly more open than those to the right, and were therefore expected to provide more information for growing the map.

Upon reaching Location 2, the robot found the corridor blocked. Additionally, now that its map of this area was more complete, there was very little unobserved space nearby, so all nearby nodes had volumetric gain values below the required threshold to act as exploration candidates. Therefore, SafeEstMDP-Map turned back towards its other unexplored frontier at Location 3, where the nodes still had high exploration scores and were considered safe.

From Location 3, the robot was able to map the entire "room" located below it (Fig. 17c), thus there was no more unexplored space there and no need to explore further downwards. Instead, the robot moved to the right, incrementally extending its map towards Location 4 as shown in the figure. It is worth noting here that Fig. 17d does not show *every* goal selected by SafeEstMDP-Map during the run, but rather a simplified intuitive representation of the path followed – in reality, when moving into unexplored territory, the algorithm tends to plan only a few nodes ahead at a time, since far-away nodes cannot be known with high confidence to be safe.

As the robot approached Location 4, the radiation levels it measured began to increase, as indicated by the darker patch in Fig. 17d. As this happened, SafeEstMDP-Map became more cautious, tending to generate plans that only moved the robot forwards one node at a time, or had it visit additional nodes near the frontier rather than pushing aggressively into the unknown region. Ultimately, when the robot did reach Location 4, the unvisited nodes ahead of it were deemed unsafe based on the GP model. The unvisited nodes in the previously explored areas of the map, on the other hand, were still safe to visit, but their volumetric gain values were below the required threshold. There were therefore no more nodes that could serve as candidates for SafeEstMDP-Map, so the run terminated at Location 4.

This is overall a successful run of SafeEstMDP-Map, exhibiting the desired behaviours of the algorithm. It balanced exploitation of volumetric gain information with safety, exploring more aggressively when confident about safety, and becoming more cautious as safety became less certain.

In total, three full runs of SafeEstMDP-Map were conducted in the Corsham Research Mine using the radiation layout shown in Fig. 17. In all runs, the system was able to plan online and successfully explore the majority of the safely accessible area. With a total accessible area of approximately 400 m^2, the system achieved consistent results of 348.31 m^2, 321.93 m^2, 345.07 m^2 of the environment explored, without violating the safety specification.

10 Conclusions

In this paper, we presented two algorithms, based on decision-making under uncertainty and GP modelling of unknown processes, for safe exploration and mapping of unknown environments. The first algorithm, SafeEstMDP-Process, assumes a known map and safely learns the distribution of an unknown hazard, whilst the second algorithm, SafeEstMDP-Map, drops the known map assumption and focuses on safely building an environment map. Both these algorithms rely on the EstMDP, a novel model which considers the uncertainty over a GP posterior of the underlying unknown process as part of its transition function. Our experiments show that our algorithms are able to safely explore in a two domains with a range of radiation configurations, and that the approach can be run in a real robot. In the future, we intend to extend the approach to goal-driven behaviour in unexplored regions, and introduce lookahead when considering where to explore, in contrast with the next-best-state selection approach presented in this paper.

A Appendix

A.1 Computational complexity

In SafeEstMDP-Process and SafeEstMDP-Map, each goal selection step requires building and solving an EstMDP, which involve respectively (i) performing GP inference for each waypoint, and (ii) solving the resulting MDP using value iteration. Both of these problems have polynomial computational complexity with respect to the number of waypoints. In

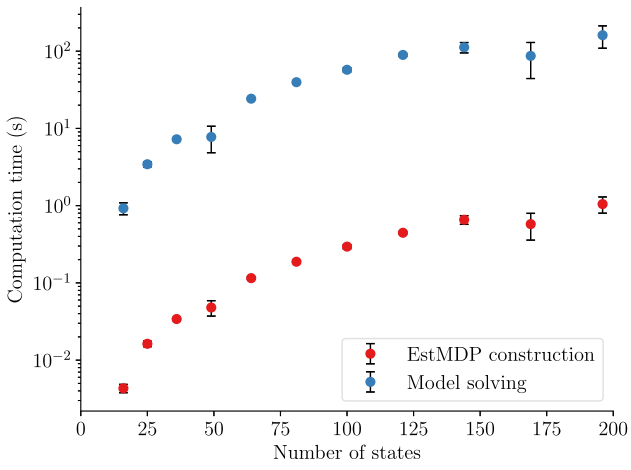


Fig. 18 Scaling of computation time used for EstMDP construction (which includes all GP inference) and model solving versus number of states. Values are cumulative over complete runs of SafeEstMDP-Process on nuclear grid map domains of different sizes, using flat radiation layouts where all states are safely reachable. Each configuration was repeated 50 times

contrast, the POMDP planning methods mentioned in Sect. 2 are PSPACE-complete for finite horizons, and would not scale to problems of this size and larger. Figure 18 shows that, for the domain sizes considered in this work, the Est-MDP solution step occupies significantly more computation time than the model construction step. As such, GP inference does not act as a computational bottleneck to the scalability of our approach.

A.2 Autonomous underwater vehicle domain details

See Appendix Fig. 19.

For the AUV domain we implemented a policy cost check alongside the policy safety check, and abandoned the current goal when the cost of the policy to reach that goal from the current state was more than 2 times the expected cost-to-goal from that state as was calculated at goal selection time.

Due to the fact that (with the absence of current) the AUV's motion takes it 2 cells per action, we also edited goal policy generation and execution to allow the policy executor to visit a goal state by passing through it (hence observing it) rather than solely by surfacing at the goal state.

The AUV dive depth was 10 m.

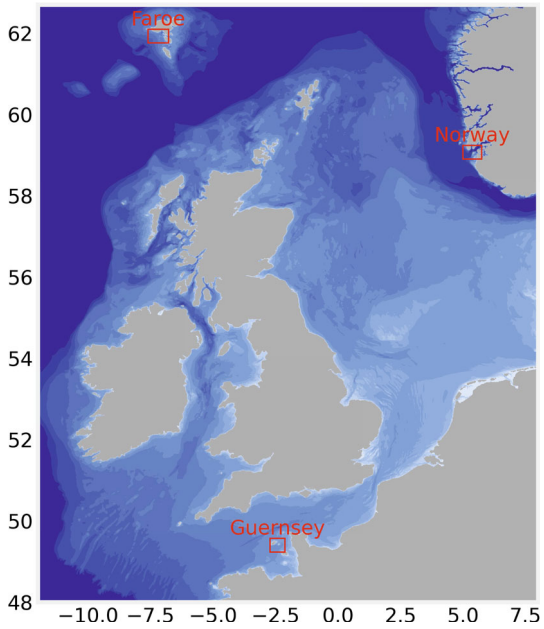
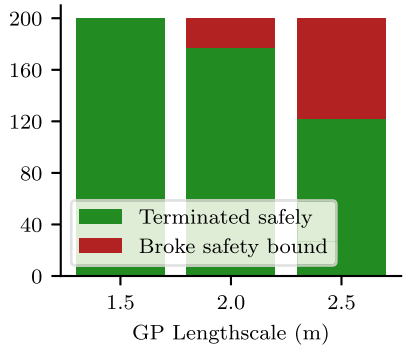


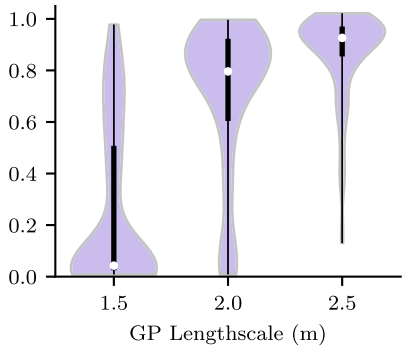
Fig. 19 Location of the evaluation areas in the northwest European shelf. Blue colour levels indicate bathmetry depth

A.3 GP comparison

See Appendix Fig. 20.



(a) Number of SafeEstMDP-Process runs that finished exploring safely or that broke the safety bound during execution, for differing values of the GP kernel lengthscale.

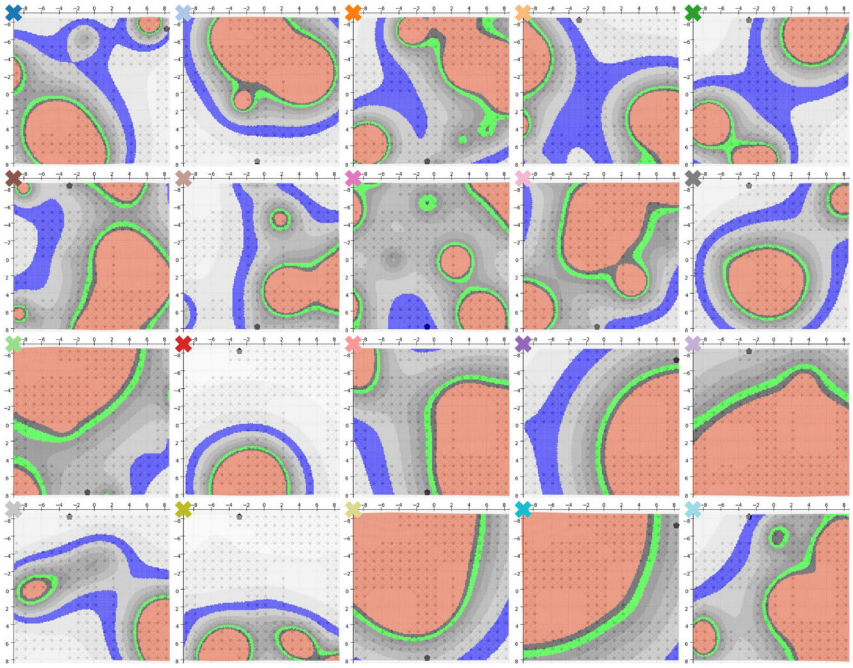


(b) Proportion of the ground-truth safely reachable set that SafeEstMDP-Process is able to correctly mark as safe, for differing values of the GP kernel lengthscale.

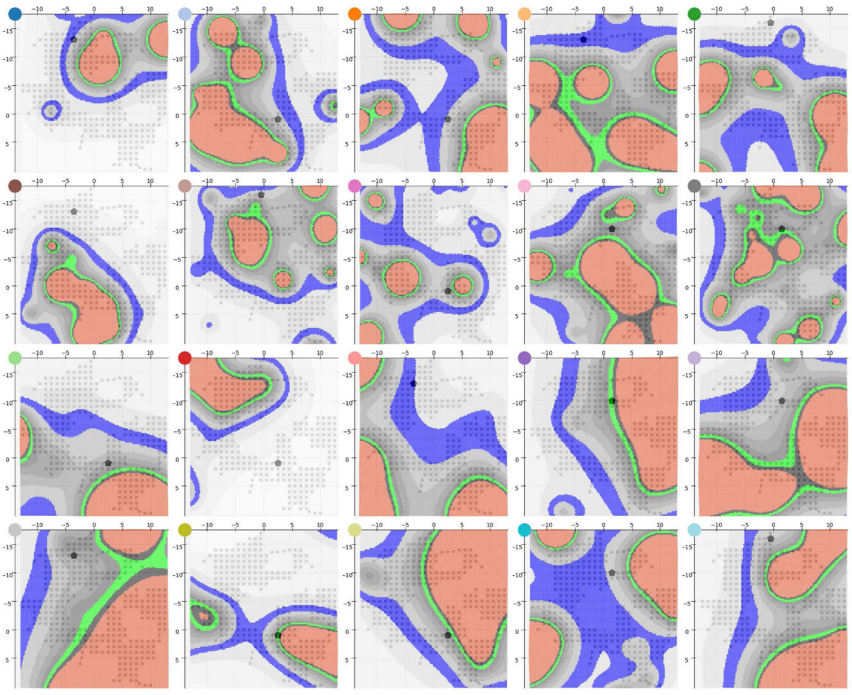
Fig. 20 The effects of varying GP kernel lengthscale (x axis) on the behaviour of SafeEstMDP-Process, using a GP model *without* log-warping. Total 200 runs per GP lengthscale value (2 domains, 20 randomly generated radiation layouts, 5 repeats)

A.4 Nuclear domain radiation scenarios

See Appendix Fig. 21.



(a) Reactor room domain sampled radiation scenarios. The top left of each scenario shows the marker used for that scenario in Figure 10.



(b) Research mine domain sampled radiation scenarios. The top left of each scenario shows the marker used for that scenario in Figure 10.

Fig. 21 Randomly sampled radiation scenarios used in Sect. 8.2. The black hexagon shows the robot's initial location. Contour levels are coloured blue to show a radiation level rad which is $30\% \leq rad < 40\%$ of the safety bound, and green to show $80\% \leq rad < 90\%$ of the safety bound. Red shows levels above the safety bound. The two rows of each figure were generated with the point source method, and the bottom two rows were generated with the Gaussian field method

Acknowledgements This work has been funded by UK Research and Innovation and EPSRC through the Robotics and Artificial Intelligence for Nuclear (RAIN) and Offshore Robotics for Certification of Assets (ORCA) hubs [EP/R026084/1, EP/R026173/1], the EPSRC Programme Grant "From Sensing to Collaboration" [EP/V000748/1], the Innovate UK AutoInspect grant [1004416], and a gift from Amazon Web Services. A.S. and M.B. were supported by Amazon Web Services Lighthouse scholarships, and A.S. was additionally supported by the AIMS Centre for Doctoral Training. M.F. was partly supported by a Royal Society University Research Fellowship.

Author Contributions M.B. led development of the SafeEstMDP-Process algorithm, conducted simulated experiments to evaluate it, and implemented the system on-robot. A.S. led development of the SafeEstMDP-Map algorithm, conducted simulated experiments to evaluate it, and implemented the system on-robot. M.S., B.C. provided engineering/technical support during development and deployment of the algorithms in simulation and on-robot. M.F., N.H., B.L. provided supervisory input throughout the research and writing process. P.D. provided supervisory input for the SafeEstMDP-Process algorithm. B.L. led the deployment on the physical robot. A.S., M.B., M.S. and B.L. conducted on-robot experimental evaluation of SafeEstMDP-Map. B.L, A.S., M.B. wrote the main manuscript text, all authors reviewed the manuscript.

Funding Engineering and Physical Sciences Research Council (EP/V-000748/1), UK Research and Innovation (EP/R026084/1, EP/R026173/1), Amazon Web Services.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

Badings, T. S., Abate, A., Jansen, N., Parker, D., Poonawala, H. A., & Stoelinga, M. (2022). Samplingbased robust control of autonomous systems with non-Gaussian noise.

Bayer, J., & Faigl, J. (2019). On autonomous spatial exploration with small hexapod walking robot using tracking camera intel realsense t265. In *2019 European conference on mobile robots (ECMR)*.

Bottero, G. A., Luis, C. E., Vinogradska, J., Berkenkamp, F., & Peters, J. (2022). Information-theoretic safe exploration with Gaussian processes. *Neural Information Processing Systems (NeurIPS), 35*, 30707–30719.

Budd, M., Duckworth, P., Hawes, N., & Lacerda, B. (2022). Bayesian reinforcement learning for single episode missions in partially unknown environments. In *6th annual conference on robot learning*.

Cao, C., Zhu, H., Choset, H., & Zhang, J. (2021). Exploring large and complex environments fast and efficiently. In*2021 IEEE international conference on robotics and automation (ICRA)* (pp. 7781–7787).

Dang, T., Tranzatto, M., Khattak, S., Mascarich, F., Alexis, K., & Hutter, M. (2020). Graph-based subterranean exploration path planning using aerial and legged robots. *Journal of Field Robotics, 37*, 13631388.

Elfes, A. (1989). Using occupancy grids for mobile robot perception and navigation. *Computer, 22*(6), 46–57.

Feyzabadi, S., & Carpin, S. (2017). Planning using hierarchical constrained Markov decision processes. *Autonomous Robots, 41*(8), 1589–1607.

Flaspohler, G., Preston, V., Michel, A. P. M., Girdhar, Y., & Roy, N. (2019). Information-guided robotic maximum seek-and-sample in partially observable continuous environments. *IEEE RAL, 4*(4), 3782–3789.

Freda, L., & Oriolo, G. (2005). Frontier-based probabilistic strategies for sensor-based exploration (vol. 2005, pp. 3881–3887). https://doi.org/10.1109/ROBOT.2005.1570713

Gopalan, N., Littman, M. L., MacGlashan, J., Squire, S., Tellex, S., Winder, J. (2017). Planning with abstract Markov decision processes. In *ICAPS*.

GPy (2012). GPy: A Gaussian process framework in Python. http://github.com/SheffieldML/GPy

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., & Burgard, W. (2013). Octomap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots, 34*(3), 189–206.

Hudson, N., Talbot, F., Cox, M., Williams, J., Hines, T., Pitt, A., & Arkin, R. C. (2022). Heterogeneous ground and air platforms, homogeneous sensing: Team CSIRO Data61's approach to the DARPA subterranean challenge. *Journal of Field Robotics, 2*, 595–636.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *AIJ, 101*(12), 99–134.

Kemeny, J., Snell, J., & Knapp, A. (1976). *Denumerable Markov chains* (2nd ed.). Springer.

Khuwaileh, B. A., & Metwally, W. A. (2020). Gaussian process approach for dose mapping in radiation fields. *Nuclear Engineering and Technology, 52*(8), 1807–1816.

Kim, S. K., Bouman, A., Salhotra, G., Fan, D. D., Otsu, K., Burdick, J., & Aghamohammadi, A. A. (2021). PLGRIM: Hierarchical value learning for largescale exploration in unknown environments. In *Proceedings of the international conference on automated planning and scheduling* (vol. 31, pp. 652–662).

Knoll, G. F. (2010). *Radiation detection and measurement*. Wiley.

Koenig, N., & Howard, A. (2004). Design and use paradigms for Gazebo, an opensource multirobot simulator. In*2004 IEEE/RSJ international conference on intelligent robots and systems (IROS) (IEEE Cat. No. 04CH37566)* (vol. 3, pp. 2149–2154).

Kwiatkowska, M., Norman, G., & Parker, D. (2011). PRISM 4.0: Verification of probabilistic real-time systems. In *Proceedings 23rd international conference on computer aided verification (CAV'11)*.

Lacerda, B., Faruq, F., Parker, D., & Hawes, N. (2019). Probabilistic planning with formal performance guarantees for mobile service robots. *IJRR, 38*(9), 1098–1123.

Lauri, M., Pajarinen, J., & Peters, J. (2019). Information gathering in decentralized POMDPs by policy graph improvement. In *AAMAS*.

Marchant, R., Ramos, F., & Sanner, S. (2014). Sequential Bayesian optimisation for spatial-temporal monitoring. In: *UAI*.

McHutchon, A., & Rasmussen, C. (2011). Gaussian process training with input noise. In*Advances in neural information processing systems (NeurIPS)* (vol. 24).

Moldovan, T. M., & Abbeel, P. (2012). Safe exploration in Markov decision processes. In *ICML*.

Morere, P., Marchant, R., & Ramos, F. (2017). Sequential Bayesian optimization as a POMDP for environment monitoring with UAVs. In *ICRA*.
Morrell, B., Thakker, R., Santamaria Navarro, À., Bouman, A., Lei, X., Edlund, J., & Burdick, J. (2022). Nebula: Team costar's robotic autonomy solution that won phase ii of DARPA subterranean challenge. *Journal of Field Robotics, 2*, 1432–1506.
Ong, S. C. W., Png, S. W., Hsu, D., & Lee, W. S. (2010). Planning under uncertainty for robotic tasks with mixed observability. *IJRR, 29*(8), 1053–1068. https://doi.org/10.1177/0278364910369861
Osborne, M. A., Roberts, S. J., Rogers, A., Ramchurn, S. D., & Jennings, N. R. (2008). Towards realtime information processing of sensor network data using computationally efficient multioutput Gaussian processes. In *International conference on information processing in sensor networks (ISPN)*.
Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J. (2009). Ros: An opensource robot operating system. In *ICRA workshop on open source software* (vol. 3, pp. 5).
Rasmussen, C., & Williams, C. (2006). *Gaussian processes for machine learning*. MIT Press.
Rouček, T., Pecka, M., Cızek, P., Petř'ıček, T., Bayer, J., Šalansk'y, V., & Krajník, T. (2022). System for multi-robotic exploration of underground environments CTUCRASNORLAB in the DARPA subterranean challenge. *Journal of Field Robotics, 2*, 1779–1818.
Rutherford, A., Duckworth, P., Hawes, N., & Lacerda, B. (2021). Motion planning in uncertain environments with rapidly-exploring random Markov decision processes. In *2021 European conference on mobile robots (ECMR)*.
Scherer, S., Agrawal, V., Best, G., Cao, C., Cujic, K., Darnley, R., & Travers, M. (2022). Resilient and modular subterranean exploration with a team of roving and flying robots. *Journal of Field Robotics, 2*, 678–734.
Silveira, P. R., Naiff, Dd. F., Pereira, C. M., & Schirru, R. (2018). Reconstruction of radiation dose rate profiles by autonomous robot with active learning and gaussian process regression. *Annals of Nuclear Energy, 112*, 876–886.
Snelson, E., Ghahramani, Z., & Rasmussen, C. (2003). Warped Gaussian processes. In *Advances in neural information processing systems (NeurIPS)* (vol. 16).
Spaan, M. T., Veiga, T. S., & Lima, P. U. (2015). Decision-theoretic planning under uncertainty with information rewards for active cooperative perception. *JAAMAS, 29*(6), 1157–1185.
Sui, Y., Gotovos, A., Burdick, J. W.,& Krause, A. (2015). Safe exploration for optimization with Gaussian processes. In *ICML*.
Sui, Y., Zhuang, V., Burdick, J., & Yue, Y. (2018). Stage-wise safe Bayesian optimization with Gaussian processes. In *ICML*.
TeichteilKönigsbuch, F. (2012). Stochastic safest and shortest path problems. In *AAAI*.
Tranzatto, M., Mascarich, F., Bernreiter, L., Godinho, C., Camurri, M., Khattak, S., & Alexis, K. (2022). CERBERUS: Autonomous legged and aerial robotic exploration in the tunnel and urban circuits of the DARPA subterranean challenge. *Journal of Field Robotics, 2*, 274–324.
Turchetta, M., Berkenkamp, F., & Krause, A. (2016). Safe exploration in finite Markov decision processes with Gaussian processes. In *NeurIPS*.
Wachi, A., Sui, Y., Yue, Y., & Ono, M. (2018). Safe exploration and optimization of constrained MDPs using Gaussian processes. In *AAAI*.
West, A., Tsitsimpelis, I., Licata, M., Jazbec, A., Snoj, L., Joyce, M. J., & Lennox, B. (2021). Use of Gaussian process regression for radiation mapping of a nuclear reactor with a mobile robot. *Scientific Reports, 11*(1), 1–11.
Williams, C., & Barber, D. (1998). Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 20*(12), 1342–1351.
Williams, J., Jiang, S., O'Brien, M., Wagner, G., Hernandez, E., Cox, M., Hudson, N. (2020). Online 3D frontier-based UGV and UAV exploration using direct point cloud visibility. In *IEEE international conference on multi-sensor fusion and integration for intelligent systems (MFI)* (pp. 263–270).
Wisth, D., Camurri, M., & Fallon, M. (2022). Vilens: Visual, inertial, lidar, and leg odometry for all-terrain legged robots. *IEEE Transactions on Robotics., 39*(1), 309–326.
Wright, T., West, A., Licata, M., Hawes, N., & Lennox, B. (2021). Simulating ionising radiation in gazebo for robotic nuclear inspection challenges. *Robotics, 10*(3), 86.
Yamauchi, B. (1998). Frontier-based exploration using multiple robots. In *Proceedings of the second international conference on Autonomous agents* (pp. 47–53).
Yamauchi, B., Schultz, A., & Adams, W. (1998). *Mobile robot exploration and map-building with continuous localization* (Vol. 4, pp. 3715–3720). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ROBOT.1998.681416
Zhu, Z., Bıyık, E., & Sadigh, D. (2020). Multiagent safe planning with Gaussian processes. In *2020 IEEE/RSJ International conference on intelligent robots and systems (IROS)* (pp. 6260–6267).

Alex Stephens is a fourth-year PhD student in the Goal-Oriented Long-Lived Systems (GOALS) lab at the Oxford Robotics Institute, and member of the AIMS Centre for Doctoral Training at the University of Oxford. He was previously an intern at the NASA Jet Propulsion Laboratory, where he worked on SLAM in challenging large-scale environments for the DARPA Subterranean Challenge. His research focuses planning over long horizons for robot monitoring missions, with particular interest in hierarchical planning and modelling continuous environments from sparse observational data.

Matthew Budd is a fourth-year PhD candidate at the Oxford Robotics Institute, working in the Goal-Oriented Long-Lived Systems lab. He earned his M.Eng. in Engineering from the University of Oxford in 2020, supported in part by an Institute of Engineering and Technology scholarship. Matthew's research focuses on decision-making under uncertainty, with a particular emphasis on capability-aware metareasoning and Bayesian reinforcement learning. He has also conducted applied robotics research in various sectors, including the commercial, space, and wireless telecommunications industries.

Michal Staniaszek is a PhD student at the Oxford Robotics Institute. He received a BSc in Computer Science in 2013 from the University of Birmingham, UK and an MSc in Systems, Control and Robotics in 2015 from the KTH Royal Institute of Technology, Sweden. In 2019 he joined the Oxford Robotics institute as a robotics software engineer, where he helped develop and deploy autonomy software for industrial environments. His research interests lie in the practical application of automated planning and scheduling techniques to robotics, with a particular focus on temporal methods and long-term autonomy.

Benoit Casseau received his MSc in computer science and applied mathematics in 2013 from École nationale supérieure d'informatique et de mathématiques appliquées, Grenoble, France. He is currently working as a robotics software engineer at the Oxford Robotics Institute, Oxford, UK.

Paul Duckworth is a Research Scientist at InstaDeep UK. He received a PhD from the University of Leeds in machine learning and mobile robotics in 2017, and held post-doctoral research positions at the Machine Learning Research Group and the Oxford Robotics Institute, both at the University of Oxford between 2017 and 2022. In 2022 he joined the Research Team at InstaDeep where his focus is sequential decision making, reinforcement learning and planning.

Maurice Fallon (IEEE, Senior Member) is an Associate Professor and a Royal Society University Research Fellow at the Oxford Robotics Institute, University of Oxford, UK. Dr. Fallon studied Electronic Engineering at University College Dublin. His PhD research in the field of acoustic source tracking was carried out in the Engineering Department of the University of Cambridge (awarded 2008). Immediately after his PhD he moved to Massachusetts Institute of Technology as a post-doc and later research scientist in the Marine Robotics Group (2008-2012). From 2012-2015 he was the perception lead of MIT's team in the DARPA Robotics Challenge - a multi-year competition developing technologies for semi-autonomous humanoid exploration and manipulation in disaster situations. At Oxford, he leads the Dynamic Robot Systems Group which focuses on probabilistic methods for localization and mapping. He has also made research contributions to state estimation for legged robots with a particular focus on achieving robustness by leveraging sensor fusion.

Nick Hawes is a Professor of AI and Robotics in the Department of Engineering Science at the University of Oxford. He directs the Oxford Robotics Institute (ORI), a federation of seven research groups spanning the breadth of robotics research. He is also a Tutorial Fellow at Pembroke College, and a Group Leader for AI/Robotics at the Alan Turing Institute, the UK's national institute for AI and data science. Within the ORI he leads the Goal-Oriented Autonomous Long-Lived Systems (GOALS) group which performs research into sequential decision-making for autonomous systems, including robots and heterogenous multi-agent teams. From 2013 to 2017 he led the EU STRANDS project which achieved breakthroughs in long-term autonomy for mobile service robots in everyday environments. More recent highlights include the deployment of an autonomous mission planning stack on a quadruped at an active nuclear site, and on an autonomous underwater vehicle harvesting data from a sensor network. He is a passionate believer in public engagement, and an Academic Champion for Public Engagement with Research at the University of Oxford.

Bruno Lacerda is a Senior Researcher at the Oxford Robotics Institute, University of Oxford, UK. He received his Ph.D. in Electrical and Computing Engineering from the Instituto Superior Técnico, University of Lisbon, Portugal, in 2013. Between 2013 and 2017, he was a Research Fellow at the School of Computer Science, University of Birmingham, UK. His research focuses on the intersection of decision-making under uncertainty, formal methods, and mobile robotics. In particular, he is interested in using a combination of techniques from learning, planning and model checking to synthesise intelligent, robust and verifiable behaviour, for both single and multi-robot systems.

4.1 Additional Discussion

Model Checking-Based Replanning

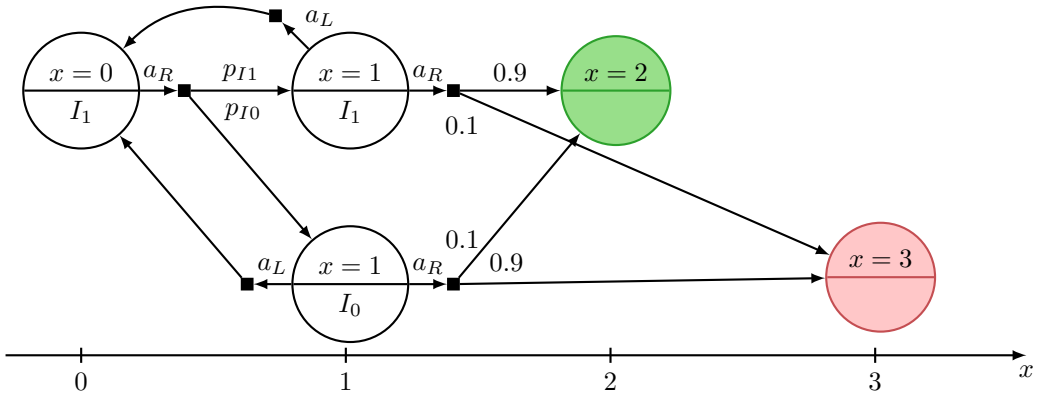


Figure 4.1: An example of an EstMDP representing an epistemically uncertain transition function. The spatial environment is 1-dimensional, and the initial state is the left hand state at $x = 0$. There are two intervals of epistemic values, represented by I_0 and I_1. The outcomes of action a_R from state $x = 1$ depend on the interval value. The green state at $x = 2$ is a goal state, and the red state at $x = 3$ is an unsafe state. For states at $x = \{0, 2, 3\}$, the two interval values are combined into one circle for clarity.

In this section, we discuss why model checking is necessary to detect when policy execution should stop and a new policy should be synthesised. In the previous chapter (Section 3.1), we discussed policy execution in a simple MDP where uncertainty that is epistemic in nature is approximated by aleatoric stochastic outcomes. Figure 4.1 shows an example of an EstMDP with similar properties to the MDP in Figure 3.1a. In this example, the outcomes of the a_R action from state $x = 1$ depend on whether the environment feature values at state $x = 1$ are within interval I_0 or I_1. Probabilities p_{I0} and p_{I1} represent the initial belief over the environment features at state $x = 1$, calculated as described in the paper. In this EstMDP, it is possible to reach the goal state $x = 2$ while avoiding the unsafe state $x = 3$ with probability 1, as long as $p_{I1} > 0$. However, the environment dynamics have ground-truth values: in reality, one interval will always be experienced at each state. In an environment where the true dynamics at $x = 1$ are I_0, executing the policy generated using the EstMDP will result in the agent looping between states $x = 0$ and $x = 1$. This potential issue is exacerbated by the lexicographic optimisation of

the constrained reachability problem (Problem 1 in the paper). Because path safety is always prioritised over cost, policies will be generated that will retry transitions many times if the outcomes include even a very small probability of leading to a safer path to the goal (i.e. $p_{I0} \gg p_{I1}$).[1]

With our replanning method, the EstMDP is updated with a new measurement at $x = 1$ when $x = 1$ is first visited. By model checking the policy, we can determine that the policy is not able to reach the goal in the new model. This triggers generation of a new policy, which will select a different goal state or a different path to reach $x = 2$. As well as accounting for the impact of information gain on the policy, model checking-based replanning also detects when stochastic outcomes lead to no longer being able to safely reach the goal state with high enough probability. This allows for recovery from stochastic outcomes that were considered unlikely when the policy was generated. Overall, the replanning method is an improvement over that used by Budd et al. (2020), which only works in cases where the agent's transition function is independent of the environment dynamics.

Finally, note that the EstMDP model makes a rectangularity assumption (Iyengar, 2005) about the environment features at each state. Environment feature values at each state are assumed to be independent of each other. This is a key assumption of the EstMDP model, and is a simplification of the true environment dynamics. Despite this assumption, we are still able to achieve good safety and cost performance in the experiments.

GP and Kernel Choice

The choice of a Gaussian process as the environment belief, and the kernel function used, entails making assumptions about the environment dynamics. The experiments in this work used the radial basis function (RBF, also known as the *squared exponential*) kernel, which is a common choice for GP-based safe exploration methods (Wachi et al., 2018; Bottero et al., 2022) alongside the Matérn 5/2

[1]This "retrying" of transitions will, however, lead to high expected cost of reaching the goal state. This makes it a less attractive goal state according to the goal state scoring function, so makes it less likely to be selected as a goal state in the first place.

kernel (Turchetta et al., 2016). These kernels are stationary and isotropic, making them straightforward to use in practice and frequently used in the literature for modelling static spatial phenomena. RBF and Matérn kernels are the most common choices for modelling radiation distributions (West et al., 2021; Khuwaileh and Metwally, 2020), for example. Khuwaileh and Metwally evaluate several kernels and find that the RBF kernel resulted in the lowest error when mapping three real spatial radiation distributions. Despite its common use, the RBF kernel is generally considered overly smooth for many real-world phenomena, as it assumes that the function is infinitely differentiable (Rasmussen and Williams, 2006).

The use of an RBF kernel is less justified in the underwater current mapping experiments, as real water current fields are neither stationary nor isotropic. A standard GP also does not enforce the divergence-free condition that is required for a valid current field. The RBF kernel worked well enough in practice to demonstrate the method in the relatively small test areas, but it is likely that a better suited kernel would enable the method to safely explore a larger proportion of the ground-truth safe state space.

Placement within the Safe RL Field

Within the broad field of Safe RL, this method falls under the category of model-based safe RL methods (Gu et al., 2024). Within the taxonomy of that survey paper, our method is primarily a Gaussian process-based method, but uses formal methods to attain probabilistic guarantees of safety. Our method also has some similarities to GP-based safe RL methods which use Gaussian Process Dynamical Models (Wang et al., 2005). Safe PILCO (Polymenakos et al., 2019) is one such method, which safely controls uncertain dynamical systems (Deisenroth and Rasmussen, 2011) using a similar state-based safety definition to ours. Also similarly, it ensures safe behaviour during exploration, rather than solely after training as many Safe RL methods do (Gu et al., 2024).

SafeEstMDP shares some similarities with the QMDP method (Littman et al., 1995), in that both transform POMDPs into fully observable MDPs to make them

more tractable. However, applying QMDP to this problem would require additional modifications to provide the probabilistic safety guarantees that SafeEstMDP achieves. The QMDP algorithm, as we discuss further in Section 5.1, would also require significantly more computation due to the need to fully solve the POMDP's underlying MDP.

4.2 Limitations and Future Work

As described in the previous section, the EstMDP model approximates the distribution of models consistent with the belief over environment dynamics. Its key approximations are (a) the assumption of rectangularity for environment features at each state, and (b) the approximation of epistemic uncertainty as aleatoric. Although the replanning method is able to detect and *react* to the impact of new information on the current policy, it is not able to *proactively* plan for this eventuality. Replacing the EstMDP in the replanning loop with a more accurate model could potentially lead to safer or faster exploration performance. However, maintaining computational tractability while improving the accuracy of the model is a challenge. The next chapter describes a fully online method that is able to lift the rectangularity assumption, though its setting is minimum cost goal-reaching rather than MDP safe exploration.

The current method performs lexicographic optimisation on the constrained reachability problem. It identifies the minimum-cost policy among those that can, with a minimum probability, reach the goal state without entering unsafe states. Approaching the problem in a multi-objective manner, where cost and safety are optimised to give a Pareto front of policies (Forejt et al., 2012; Lacerda et al., 2017), could lead to better goal choices and more easily tuned exploration behaviour.

Although we also evaluated the method with a Matérn 3/2 kernel and saw similar exploration performance, we did not investigate how well different kernels and hyperparameter choices modelled the ground-truth environment dynamics. Similarly to other safe exploration works (Turchetta et al., 2016; Wachi et al., 2018; Bottero et al., 2022), our focus was on how different hyperparameter values affected

exploration performance and safety, rather than how well the GP modelled the environment dynamics. Also in common with previous works, we did not maintain and marginalise over hyperparameter belief distributions, instead using a single set of hyperparameters for each experiment. Maintaining beliefs over uncertain hyperparameter values can result in significantly better-calibrated predictive uncertainty (Svensson et al., 2015; Simpson et al., 2021), and would not significantly increase the computational complexity of the method as most computational effort is spent on EstMDP model checking rather than GP inference (paper, Fig. 18). We did however conduct additional experiments using type II maximum likelihood estimation (Rasmussen and Williams, 2006) to tune the GP hyperparameters at each replanning iteration, given the priors and the data collected so far. However, the safe exploration setting breaks the standard independent and identically distributed (i.i.d.) assumption, as the dataset will only contain data from the safe region. As the dataset grows, GP hyperparameters are tuned to model the safe region well, and do not generalise well to unsafe regions. This safe region overfitting continues until the GP, via the PKSMDP, incorrectly predicts that a path through an unsafe area is sufficiently safe, causing the agent to fail the safety specification. This effect can be slowed by using narrower priors over hyperparameters, but this reduces the utility of hyperparameter tuning. Further research could investigate methods for hyperparameter tuning with biased datasets, allowing the agent to perform model structure improvements during exploration and enabling marginalisation over hyperparameters for better GP uncertainty estimates.

There are some real-world environment effects that are difficult to capture with a single GP model. For example, in the radiation mapping case, certain materials can cause radiation shadowing and reflection. The greatest effect on safety would be caused by a sudden change in radiation levels, as the agent moves from behind a shielding object to a clear line of sight to a strong radiation source. This very sharp spatial gradient would be difficult to capture with a GP without overfitting and affecting the model's ability to generalise. Similar shadowing problems are tackled in the wireless communication modelling literature (Hähnel and Fox, 2006;

Fink and Kumar, 2010), although these methods typically require reasoning over the locations of signal/radiation *sources*, rather than the signal/radiation level spatial distribution. These methods also commonly use a GP prior mean function conditioned on signal propagation models, and/or multiple GPs for different areas or signal sources. Inspired by these methods, to solve the radiation shadowing problem, we could use a prior mean function based on the 2D or 3D map built by the agent, pessimistically assuming that radiation levels are higher in corners and behind obstacles. Further research is needed to determine whether this can be carried out in a principled and consistent manner.

Finally, recent work in a control-theoretic setting (Prajapat et al., 2025) is able to carry out safe exploration of deterministic non-linear systems. Similarly to other MDP safe exploration methods, only a scalar constraint function is considered rather than agent dynamics being altered by environment dynamics as in our work. However, their method applies to continuous state and action spaces, where ours is limited to discrete state and action spaces. The next chapter, which uses a GP model of the environment dynamics in a similar manner to this chapter, discusses the challenges of extending our methods from discrete to continuous state and action spaces.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Planning under uncertainty for safe robot exploration using Gaussian process prediction
Publication Status	☑ Published ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and unsubmitted work written in a manuscript style
Publication Details	Alex Stephens, **Matthew Budd**, Michal Staniaszek, Benoit Casseau, Paul Duckworth, Maurice Fallon, Nick Hawes, and Bruno Lacerda. Planning under uncertainty for safe robot exploration using Gaussian process prediction. *Autonomous Robots*, 48(7):18, 2024.

Student Confirmation

Student Name:	Matthew Budd		
Contribution to the Paper	• I led the designed and implemented the SafeEstMDP-Process algorithm. • I designed and carried out simulation experiments for SafeEstMDP-Process. I designed and carried out real-world experiments for the system in collaboration with Alex Stephens. • I wrote all SafeEstMDP-Process-related sections of the paper (including Sections 5, 6, 8.1, 8.2, and 8.3) with review and input from other authors. • Introduction, related work, prelims and conclusions were jointly written.		
Signature		Date	April 24, 2025

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title:	Professor Nick Hawes		
Supervisor comments	I confirm that Matthew made a substantial contribution to the publication, and that the description above is accurate.		
Signature		Date	April 25, 2025

This completed form should be included in the thesis, at the end of the relevant chapter.

5 Online Adaptation to Uncertain Environments

	Chapter 3	Chapter 4	Chapter 5	Chapter 6	Chapter 7
Epistemic uncertainty	Environment dynamics	Environment dynamics	Environment dynamics	Planner dynamics	Planner dynamics
Mode	Offline	Replanning	Online	Offline	Online
Model	Timed MDP	EstMDP	BAMDP	Metalevel MDP	Metalevel MDP
Solution method	VI (max reward)	VI (reachability)	MCTS	Deep RL	Deep RL
Data regime	Small/ medium	Small	Small	Large	Large

Table 1.1: Settings, methods and features for each chapter.

We now move on to the *zero-shot adaptation* setting: maximising performance in the specific environment instantiation encountered by the agent during execution. We approach this problem using online decision-making, given measurements of the environment dynamics taken so far. Similarly to the previous chapter, a single environment instantiation is encountered at test time, and the agent has only one opportunity to operate in it. Compared to common settings in e.g. episodic reinforcement learning, the agent will not repeatedly encounter the same environment instantiation. Given the wide distribution over possible environments, it is not possible to pre-plan for (or, equivalently, learn a deep RL policy for) every possible environment instantiation before the mission.

We make use of the same factored state space and transition function formulation as the previous chapter, given its ability to represent a clean separation between agent and environment dynamics. The previous chapter made use of a rectangularity

assumption for scalability, effectively assuming no correlation of independent environment dynamics across the state space. In this chapter, we leverage methods from Bayesian RL to make planning with *consistent* environment instantiations tractable. By *consistent*, we mean that each draw from the environment belief distribution is a coherent, valid possible environment instantiation. This allows the agent to consider correlated environment dynamics across the possible agent states when making decisions.

In this work, the PKSMDP is replaced with a U-MDP, a more abstract model that does not assume that the agent operates on a topological map composed of waypoints and edges. The PKSMDP is a special case of the U-MDP, but the state space factorisation and transition function structure are the same. The differences in notation are shown in Table 5.1.

PKSMDP $\mathcal{M}^O = \langle S^O, \overline{s^O}, A^O, T^O, C^O \rangle$	**U-MDP** $\mathcal{M}^o = \langle S^o, s_0^o, A^o, T^o, C^o, G^o \rangle$
$S^O = V \times O$	$S^o = S_k \times S_e$
$\overline{s^O} = (\overline{v}, f(\overline{v}))$	$s_0^o = \big(s_{k,0}, o(s_{k,0})\big)$
$A^O = V \times V$	A^o
$T^O : (V \times O) \times A^O \times V \to [0, 1]$	$T^o : (S_k \times S_e) \times A^o \times S_k \to [0, 1]$
$C^O : S^O \times A^O \to \mathbb{R}_{\geq 0}$	$C^o : S^o \times A^o \to \mathbb{R}_{\geq 0}$

Table 5.1: Formulation comparison, PKSMDP (Chapter 4) vs. U-MDP (Chapter 5). The PKSMDP assumes a topological map with waypoints V and actions defined between waypoints, whereas the U-MDP does not assume this. The two state space component factors are renamed, and the unknown dynamics mapping function f is renamed to o.

For the same reasons as in the previous chapter, we use a Gaussian Process (GP) to model the environment dynamics. From the Bayesian RL viewpoint, this GP is the agent's belief about the environment dynamics. The GP prior, equivalent to a prior belief over possible transition functions, provides a principled way to include prior information about possible environment dynamics. Basic information about the environment dynamics can be encoded in the GP kernel: in the underwater navigation example, priors can inform the agent about typical ranges of water current magnitude and spatial variation found in the ocean.

The most common problem setting where GPs are used for online decision-making in environments with unknown dynamics is informative path planning (IPP) (Morere et al., 2017; Flaspohler et al., 2019; Blanchard and Sapsis, 2022). Their objective is to maximise the information gain about the environment dynamics, rather than minimise the cost to achieve a goal specification. These methods must carry out similar multi-step decision-making with GP beliefs, but differ in that they generally assume that agent transitions are deterministic and do not depend on the environment dynamics.

We make the following contributions: (i) reformulation of the U-MDP/PKSMDP model into a Bayesian reinforcement learning framework, and (ii) using techniques from the Bayesian RL literature, adapted to our model, to achieve adaptive behaviour. Compared to previous IPP and GP-based decision-making methods, our method achieves decision-making that is faster by one or two orders of magnitude. This allows our method to make better decisions given the same time budget, incurring lower cumulative cost in experiments in the radiation exposure and underwater navigation domains.

Citation:

Matthew Budd, Paul Duckworth, Nick Hawes, and Bruno Lacerda. Bayesian reinforcement learning for single-episode missions in partially unknown environments. In *Conference on Robot Learning (CoRL)*. PMLR, 2023. URL `https://proceedings.mlr.press/v205/budd23a/budd23a.pdf`

Please note that the version of the paper and its appendix included in this thesis have small changes to the published version:

- In the introduction, updated the comparison to RL methods to refer to a contemporaneous work (Cao et al., 2023) that applies to a similar problem setting.
- Clarified why it is useful to sample functions from the GP posterior, and the complexity of doing so.
- In the appendix, added a section on GP kernel function choice and hyperparameter priors, and fully described the bias kernel used in the experiments.

Bayesian Reinforcement Learning for Single-Episode Missions in Partially Unknown Environments

Matthew Budd **Paul Duckworth** **Nick Hawes** **Bruno Lacerda**
Oxford Robotics Institute, University of Oxford
{mbudd, pduckworth, nickh, bruno}@robots.ox.ac.uk

Abstract: We consider planning for mobile robots conducting missions in real-world domains where *a priori* unknown dynamics affect the robot's costs and transitions. We study single-episode missions where it is crucial that the robot appropriately trades off exploration and exploitation, such that the learning of the environment dynamics is *just enough* to effectively complete the mission. Thus, we propose modelling unknown dynamics using Gaussian processes, which provide a principled Bayesian framework for incorporating online observations made by the robot, and using them to predict the dynamics in unexplored areas. We then formulate the problem of mission planning in Markov decision processes under Gaussian process predictions as Bayesian model-based reinforcement learning. This allows us to employ solution techniques that plan more efficiently than previous Gaussian process planning methods are able to. We empirically evaluate the benefits of our formulation in an underwater autonomous vehicle navigation task and robot mission planning in a realistic simulation of a nuclear environment.

Keywords: Planning under Uncertainty, Gaussian Processes, Single-Episode Bayesian Reinforcement Learning

1 Introduction

Real-world mobile robots rarely have complete knowledge of their environment dynamics. When operating under uncertainty, they need to be able to incorporate their online *observations* of uncertain environment features into their plans. In this paper, we consider the single-episode setting where a robot must carry out a mission in an environment for which the dynamics are not fully known at deployment time. The mission is specified by a goal state(s) that the agent must eventually reach while minimising incurred cost under an environment feature that has *a priori unknown dynamics*. The robot's information gathering capabilities are limited, as the environment features are only observable at the robot's current state. An example would be a Geiger counter-equipped robot minimising its cumulative radiation exposure in an environment with an unknown radiation distribution. In this setting, it is infeasible to pre-plan for every possible environment that might be encountered. Training an RL agent on such a wide distribution of possible environment dynamics may take significant time or be infeasible, depending on the range of true environment dynamics that may be encountered [1].

To model continuous environment features, we follow previous works [2, 3, 4, 5] and use a Gaussian process (GP) [6] to predict unknown dynamics away from the agent's current location. GPs are well-suited for modelling spatio-temporal distributions by incorporating online measurements into the posterior distribution, along with a measure of predictive uncertainty. GP prior hyperparameters can be estimated from physical intuition or from a small dataset of similar environments. We discuss this further in Section 3 of the appendix. Similarly to [2, 3, 4, 5], our GP maintains a *belief* over the underlying dynamics of the environment. Rather than ensuring safe environment exploration or maximising information collected, we extend these works by formulating a unified Bayes-optimal framework for efficient online mission planning.

We therefore pose our task as online, model-based Bayesian RL (BRL) with a GP belief over the transition function. As all environmental uncertainty is then encapsulated within the transition function, we assume that the cost function is known given an instance of the environment dynamics. Continuing the previous example, the robot does not need to learn online that it incurs more damage

from higher levels of radiation. Our problem statement also assumes a discrete state space (except for unknown environment dynamics, which are continuous) and full observability of the current state.

The BRL formulation provides several advantages. First, it is known to optimally trade off exploration and exploitation [7], which is crucial in the single-mission setting we address: over-exploration may hinder mission performance or increase the risk of failure. Second, it allows us to intuitively encode the agent's transition function and local observability limitations, and reflects the existence of fixed (but *a priori* unknown) true underlying environment dynamics; third, it enables the use of efficient Monte-Carlo planning approaches developed for BRL.

Our contributions are to 1. formulate goal-driven planning in unknown environments as model-based BRL; 2. adapt the Bayes-adaptive Monte-Carlo planning (BAMCP) algorithm [7] for our GP-based goal-driven BRL problem formulation; and 3. use this formulation and algorithm to improve on previous GP planning approaches, both in expressibility and computational efficiency. In particular, we exploit techniques that allow us to efficiently sample possible environment dynamics from the GP to use during planning. To the best of our knowledge, we are the first to apply BRL with GP belief models to goal-driven planning in partially unknown environments.

2 Related Work

Non-myopic decision-making with an unknown transition function requires reasoning over possible observation (i.e. function evaluation) sequences. This task has been investigated from the perspective of several fields, including sequential Bayesian optimisation (BO) of unknown functions. An example BO objective could be to stay within a computational evaluation budget while improving a GP model of the unknown function. Some recent methods are able to pose multiple-step look-ahead in GPs as a single joint optimisation problem [8]. However, we focus on physical systems such as mobile robots which are required to physically move and observe any queried location: observations cannot be made in parallel or at freely specified locations. This implies reachability, observability and cost limitations that are not usually considered when using BO.

Considering some of these aspects, recent literature [9, 4] performs non-myopic "informative path planning" for environmental monitoring. Similar to our approach, these methods perform Monte-Carlo tree search (MCTS) [10] in belief space with a GP belief. However, they assume robot actions have deterministic outcomes, and do not consider the case where the unknown environment features can affect robot transition dynamics, as we do. This recent literature uses a partially observable Markov decision process (POMDP) [11] problem formulation, and plan with MCTS trees of computationally expensive GP-represented beliefs due to their BO objective. Our Bayes-adaptive MDP (BAMDP) [12] formulation more appropriately represents the existence of fixed but *a priori* unknown environment dynamics. In the context of a goal-based planning objective, the two formulations and solution methods are equivalent, as we demonstrate in this paper. However, we use model-based BRL techniques to *root sample* the GP environment belief. This avoids computationally expensive belief updates, enabling the construction of larger MCTS trees within the same computational budget.

Monte-Carlo tree search in GP-modelled unknown environments has also been carried out in [13], where the environment features affect only transition durations in a semi-MDP. A model-based BRL method with a Gaussian process dynamical model (GPDM) belief representation was presented in [14]. Their method must make restrictive maximum likelihood transition/observation assumptions for tractability, due to their unfocused Monte-Carlo planning algorithm. Alternatively, some RL techniques such as Gaussian Process temporal difference [15] use GPs to directly model an MDP value function. We argue that GP modelling the real-world environmental phenomena, rather than the value function, lets us provide physically principled and interpretable prior knowledge.

3 Preliminaries

Markov Decision Processes. We consider stochastic shortest path (SSP) MDP problems [16], as they are well-suited for specifying single episode goal-driven missions. An SSP MDP is defined as a tuple $\mathcal{M} = \langle S, s_0, A, T, C, G \rangle$, where S is a finite set of states; $s_0 \in S$ is the initial state; A is a finite set of actions; $T : S \times A \times S \to [0, 1]$ is a probabilistic transition function $T(s, a, s') = p(s' \mid s, a)$; $C : S \times A \to \mathbb{R}_{\geq 0}$ is a cost function; and $G \subset S$ is a set of absorbing, zero-cost goal states. A history h of an MDP $\mathcal{M}$ is a state-action sequence $s_0 a_0 s_1 a_1 \cdots a_{t-1} s_t$ such that $T(s_i, a_i, s_{i+1}) > 0$

for all $i \in \{0, \cdots, t-1\}$. We denote the set of all histories of $\mathcal{M}$ as $\mathcal{H}^{\mathcal{M}}$. A stationary, deterministic policy is a mapping $\pi : S \to A$ that defines the action to take at each state. A policy is proper in a state s if it reaches a goal state $s_g \in G$ when starting from s with probability 1. In an SSP MDP there must exist a policy that is *proper* in all states, and all improper policies must incur infinite cost. Under these assumptions, there exists a cost-optimal proper policy [17].

Bayesian RL. In BRL, an agent uses Bayesian inference to maintain a posterior distribution, or *belief*, over the true dynamics of the underlying model given some prior distribution. For an MDP, either or both of the transition function T and cost function C could be a priori unknown. We focus on the case where only T is unknown. C can be assumed to be known in our setting, as it represents the known effect of a *given instance* of environment dynamics on the robot. For example, the time cost of travelling against given water current vectors can be calculated given the value of the vectors and the vehicle's known dynamics. Given a history $h = s_0 a_0 s_1 a_1 \cdots a_{t-1} s_t$, it is possible to generate the posterior belief over the transition function T given h. This can be carried out with successive applications of Bayes' rule $p(T \mid h_i) \propto p(h_i \mid T)p(T)$ from the initial history $h_0 = s_0$ up to the full history $h_t = h$. This allows for the definition of a Bayes-adaptive MDP (BAMDP) [12], which achieves Bayes optimality by adding histories to its state representation, and encoding uncertainty over T in its transition function. Let $\mathcal{M} = \langle S, s_0, A, T, C, G \rangle$ be an MDP with a prior belief $p(T)$ over the true transition function T. The corresponding BAMDP is an MDP $\mathcal{M}^+ = \langle S^+, s_0^+, A, T^+, C^+, G^+ \rangle$, where $S^+ = S \times \mathcal{H}^{\mathcal{M}}$; $s_0^+ = (s_0, h_0)$; $C^+((s,h),a) = C(s,a)$; $G^+ = \{(s,h) \in S^+ \mid s \in G\}$; and

$$T^+((s,h),a,(s',has')) = \int_T T(s,a,s')p(T \mid h)dT. \tag{1}$$

Although the state-history pairs in S^+ are redundant because the current state can be extracted from the history, we use the (s,h) notation for clarity as in [7]. A policy in a BAMDP is a mapping $\pi : S \times \mathcal{H}^{\mathcal{M}} \to A$. The optimal policy π^* minimises the expected cumulative cost to reach G^+, given the prior over T. This policy is stationary in S^+ but is history-dependent in the original MDP. π^* considers the posterior $p(T \mid h)$ and adapts its action selection to account for the conditional distribution of T given the observed h.

Gaussian Processes. A GP is a collection of random variables, any finite number of which have a joint Gaussian distribution [6]. A GP regression is of the form $o(s) \sim \mathcal{GP}(m(s), k(s,s'))$, giving a probability distribution over functions fully specified by the mean $m(s)$ and kernel $k(s,s')$ functions. We can let $m(s) = 0$ without loss of generality. Given a dataset of n noisy observations $\mathcal{D} = \{(s_i, o(s_i) + \epsilon_i)\}_{i=1}^n$ for locations s_i and where $\epsilon_i \sim \mathcal{N}(0, \sigma_\eta^2)$ is Gaussian observation noise, GP regression predicts unknown environment feature values at all inputs s_*. The kernel function k is parameterised by hyperparameters $\boldsymbol{\theta}$. Given hyperparameter priors $p_0(\boldsymbol{\theta})$, their values are commonly optimised by maximising the log marginal likelihood for the model given the dataset. The resulting Gaussian posterior, conditioned on the observations $\mathbf{o} = [o(s_1) + \epsilon_1, \ldots, o(s_n) + \epsilon_n]^\mathsf{T}$, is a multivariate normal $p^{\mathcal{GP}}\big(o(s_*) \mid s_*, \mathcal{D})\big) \sim \mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$, where $\boldsymbol{\mu}_* = \mathbf{K}_*^\mathsf{T}(\mathbf{K}_n + \sigma_\eta^2 \mathbf{I})^{-1}\mathbf{o}$, and $\boldsymbol{\Sigma}_* = \mathbf{K}_{**} - \mathbf{K}_*^\mathsf{T}(\mathbf{K}_n + \sigma_\eta^2 \mathbf{I})^{-1}\mathbf{K}_*$.

The positive semi-definite kernel matrix $\mathbf{K}_n = [k(s,s')]_{s,s' \in \mathbf{s}_n}$, $\mathbf{K}_* = [k(s,s')]_{s \in \mathbf{s}_n, s' \in \mathbf{s}_*}$, $\mathbf{K}_{**} = [k(s,s')]_{s,s' \in \mathbf{s}_*}$, and $\mathbf{I} \in \mathbb{R}^{n \times n}$ is the identity matrix. We wish to sample possible, coherent environment dynamics using the GP, to enable MCTS-based decision making in Section 4.3. Sampling from the Gaussian process posterior at a finite set of m test points requires computing the joint covariance structure between the training and test points, incurring $O((n+m)^3)$ computational cost. Once this has been computed, drawing additional function samples has complexity $O(m^2)$.

4 Approach

4.1 Problem Formulation

In order to clearly separate known system transition dynamics from the unknown environment dynamics, we represent the unknown environment and its effect on the agent as an *MDP with Unknown Feature Values (U-MDP)* [5].

Let S_k be a set of state features with discrete, known values (e.g. pose of a robot in a grid map) and S_e a set of state features with unknown values in $\mathbb{R}$ (e.g. the water current vector at a pose).

Let $o : S_k \to S_e$ be an *a priori* unknown mapping that specifies the values $o(s_k) \in S_e$ observed at locations $s_k \in S_k$. An SSP U-MDP is a tuple $\mathcal{M}^o = \langle S^o, s_0^o, A^o, T^o, C^o, G^o \rangle$ where: $S^o = S_k \times S_e$; s_0^o is the initial state $s_0^o = \big(s_{k,0}, o(s_{k,0})\big)$; A^o is a finite set of actions; T^o is the U-MDP transition function $T^o : (S_k \times S_e) \times A \times S_k \to [0, 1]$. As the state of the U-MDP is uniquely defined by the value of the known state feature(s) $s_k \in S_k$, the transition function of the U-MDP only represents the change in the known state feature(s); $C^o : S^o \times A \to \mathbb{R}_{\geq 0}$ is the cost function; and $G^o \subset S_k$ is the set of goal states, defined only across known value state features as $o(s_g)$ is not known for all $s_g \in G^o$. The problem addressed in this paper is formalised as an SSP U-MDP. The objective is to find a policy that minimises the expected cost to a reach state $(s_k, o(s_k)) \in S^o$ such that $s_k \in G^o$.

4.2 From U-MDPs to GP-BAMDPs

For notational simplicity, in the following we assume a single state feature with unknown values. The approach presented below can easily be extended to cases with more than one unknown value state feature, either using a multi-output GP [18] or multiple single-output GPs. The former assumes non-independent feature dynamics, where learning about one could improve predictions of another.

To estimate the unknown mapping o, we propose that the agent maintains a GP model created by adding a new observation of o at each timestep. Specifically, after observing history $h = (s_{k,0}, s_{e,0})a_0(s_{k,1}, s_{e,1})a_1 ... a_{t-1}(s_{k,t}, s_{e,t})$, we define $\mathcal{D}_h = \{(s_{k,i}, s_{e,i}) \mid i \in \{0, \dots, t\}\}$. Then, the GP model is denoted as $\mathcal{GP}_{\mathcal{D}_h}$ and the GP posterior over $o(s_k)$ is given by $p^{\mathcal{GP}}\big(s_e \mid s_k, \mathcal{D}_h\big)$. We assume that observations of o have negligible noise, which corresponds to full observability of the current state. Finally, note that by modelling unknown environment features with a GP we are implicitly making regularity and Lipschitz continuity modelling assumptions on the environment feature functions [6].

We now formulate SSP U-MDP as a BAMDP with GP belief over the transition function.

Proposition 1. *Let $\mathcal{M}^o = \langle S^o, s_0^o, A^o, T^o, C^o, G^o \rangle$ be a U-MDP. The GP-BAMDP for $\mathcal{M}^o$ is a BAMDP $\mathcal{M}^{o+} = \langle S^{o+}, s_0^{o+}, A^o, T^{o+}, C^{o+}, G^{o+} \rangle$ where the transition function incorporates the GP posterior. Formally, for $s = (s_k, s_e)$, $s' = (s'_k, s'_e) \in S^o$; $a \in A$; and $h = (s_{k,0}, s_{e,0})a_0(s_{k,1}, s_{e,1})a_1 ... a_{t-1}(s_{k,t}, s_{e,t}) \in \mathcal{H}^{\mathcal{M}^o}$ for $(s_{k,t}, s_{e,t}) = (s_k, s_e)$:*

$$T^{o+}((s, h), a, (s', has')) = \quad T^o\big((s_k, s_e), a, s'_k\big) \cdot p^{\mathcal{GP}}\big(s'_e \mid s'_k, \mathcal{D}_h\big). \tag{2}$$

Proof. We start by noting that the integral in Equation (1) is the product of two components. The first component is a specific possible transition function. As the single unknown component of T^o is the mapping o, the value of $T^o(s, a, s')$ given knowledge of o is defined according to the U-MDP transition function, ensuring that the unknown state feature dynamics are consistent with o:

$$T^o((s_k, s_e), a, (s'_k, s'_e) \mid o) = T^o\big((s_k, s_e), a, s'_k\big)\mathbb{I}[s'_e = o(s'_k)], \tag{3}$$

where $\mathbb{I}[.]$ is the indicator function. The second component is the posterior distribution over possible transition functions given a history h, which in our case is the GP posterior $p^{\mathcal{GP}}$:

$$p(o(s'_k) \mid h) = p^{\mathcal{GP}}\big(o(s'_k) \mid s'_k, \mathcal{D}_h\big). \tag{4}$$

As the unknown state feature dynamics are deterministic given o, the integral over T from Equation (1) only has value when the indicator function in Equation (3) is 1. This leads to Equation (2), where all uncertainty in T is captured in the GP posterior over o, and T^{o+} represents the combination of the U-MDP transition function and the GP belief over the values of the unknown state features. □

4.3 Solving U-MDPs with BAMCP

GP-BAMCP. Having framed the U-MDP mission planning problem as a BAMDP, we can exploit MCTS planning frameworks that were developed in this context. Specifically, we base our algorithm on BAMCP [7]. This algorithm plans in belief space[1], but builds search trees of action-observation histories rather than of belief states. The search tree consists of alternating state and action nodes and is constructed over the course of Monte-Carlo trials starting from the root node and sampling action outcomes from a generative model. BAMCP adapts the concept of *root sampling* from POMCP [19],

[1] The distribution over the transition function T in the case of BAMCP.

Algorithm 1 GP-BAMCP: PLAN

1: **procedure** PLAN($\mathcal{GP}_{\mathcal{D}_h}$, history h, start state (s_k, s_e), goal state set G^o)
2: **repeat**
3: $\hat{o} \leftarrow$ sample from $\mathcal{GP}_{\mathcal{D}_h}$
4: $\hat{T}((s_k, s_e), a, (s'_k, s'_e)) \leftarrow T^o((s_k, s_e), a, (s'_k, s'_e) \mid \hat{o})$
5: SIMULATE$((s_k, s_e), h, \hat{T}, G^o)$
6: **until** TIMEOUT( )
7: **return** $\arg\min_a Q(h, a)$
8: **end procedure**

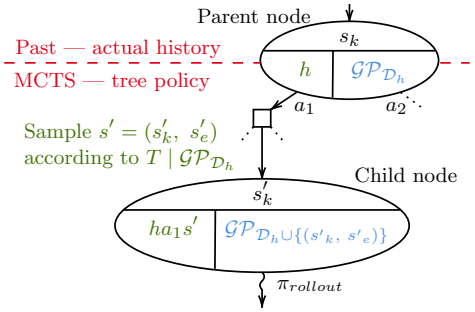


Figure 1: Example BAMDP MCTS search-tree, with search nodes as ellipses. Green shows search node generation/contents with a root sampling approach, and blue shows full belief planning.

where for each MCTS trial a transition function T is sampled from the root belief node and used throughout the trial. Actions are chosen inside the search tree using a *tree policy*, most commonly UCT [10]. New leaf nodes' values are estimated heuristically by continuing the trial trajectory from the leaf node using a *rollout policy*.

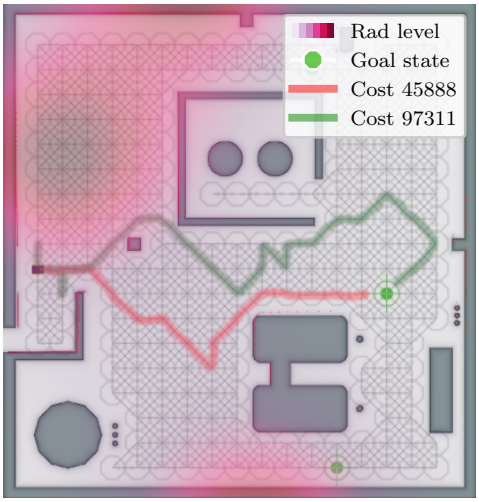
Our online MCTS planning algorithm, a modified version of BAMCP for GP beliefs, is described in Algorithm 1. To simplify the presentation, we assume that the rollout policy is able to reach the goal set with probability 1. Our algorithm replaces BAMCP's depth d and reward R parameters with the goal set G and costs C, respectively, to reflect the SSP mission setting of our work. In our case root sampling of T is performed by sampling from the current GP posterior, as described in lines 3 and 4. Concretely, we take a sample $\hat{o} \sim \mathcal{GP}_{\mathcal{D}_h}$ where $\hat{o} : S_k \to S_e$ is a possible mapping from known states to values of the unknown state features. Sampling $\hat{o}$ from the root belief node corresponds to sampling a possible environment that is consistent with the current GP environment model, i.e. trained only on real-world observations. Finally, in line 7, the agent greedily selects a single real-world action by running MCTS trials up to a computational budget.

Treatment of continuous s_e. Equation (2) represents a BAMDP with both discrete (known value) and continuous (unknown value) state features, and therefore also a combination of a discrete transition function, T^o, and a continuous transition function given by the GP belief over o. This presents challenges for MCTS methods, since the probability of transitioning to the same state twice is 0. To enable generalisation between histories and allow the search tree to reach depths greater than 1, our algorithm aggregates similar outcomes in s_e into the same child search node. Each search node has an associated "mean" value $\overline{s_e}$, where all outcomes $\|s'_e - \overline{s_e}\|_1 < \epsilon$ will be associated with that node. When two MCTS trials sample $s = (s_k, s_e)$ and $s' = (s_k, s'_e)$ from the same start state and action, if $\|s_e - \overline{s_e}\|_1 < \epsilon$ and $\|s'_e - \overline{s_e}\|_1 < \epsilon$, the two histories will be associated with the same child node. Note that this only associates histories to nodes and does not discretise the computation of reward or transition probabilities. The ϵ parameter therefore controls the search-tree branching factor, similarly to ϵ in [13] and the two branching factor parameters in progressive widening MCTS [20].

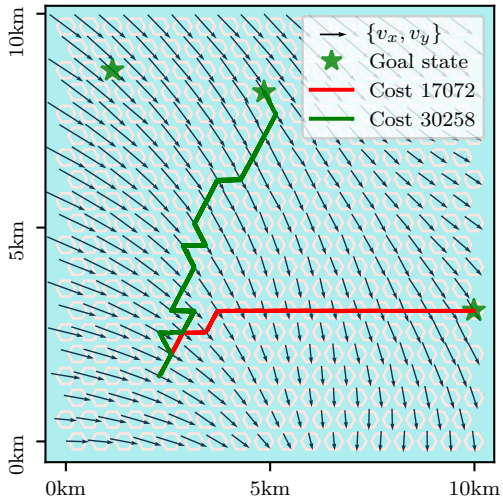
4.4 Theoretical Analysis

Equivalence to Partially Observable MDP Belief Planning Methods. Figure 1 depicts an example MCTS search-tree mid-mission. Several real actions have been taken in the environment, corresponding to the history h in the root node. When carrying out BAMCP root sampling, a new leaf node is added to the MCTS tree by appending the parent node history with a new state. This new state $s \sim T(s, a, \cdot)$ is sampled from the MDP transition function which was itself sampled from the root for this MCTS simulation, in lines 3 and 4 in Algorithm 1. This is depicted in green in Figure 1.

Several previous approaches [9, 4] that integrate GPs and MDPs for decision-making in uncertain environments use a partially observable MDP (POMDP) [11]. We call these *POMDP belief MCTS* approaches. These methods incorporate a GP environment belief into the POMDP state, and use MCTS to plan in belief space. Here, new belief state leaf nodes are generated using a *belief update* from the parent node. Belief update in this context is carried out by sampling a hypothesised data



(a) Example radiation domain visualisation, with a robot trajectory generated by GP-SSP-BAMCP (red) and by GP belief MCTS (green) algorithms.



(b) Example ocean currents domain visualisation, with an AUV trajectory generated by GP-SSP-BAMCP (red) and by GP belief MCTS (green) algorithms.

Figure 2: Experiment domains. Randomly selected multi-goal problem instances are shown.

point $\tilde{o} \sim p^{\mathcal{GP}}\big(\cdot \mid s_k, \mathcal{D}_{parent}\big)$ from the parent node GP belief posterior and augmenting the child belief node's GP dataset with this new point: $\mathcal{D}_{parent} \cup \{(s_k, \tilde{o})\}$. This is shown in blue in Figure 1, where the GP in the child node can be explicitly generated by a belief update from the parent node GP. GP belief updates require adding a single new data point to the model, the complexity of which can be reduced from $O(N^3)$ to $O(N^2)$ where N is the number of data and sample points [18]. Even with this reduction, the belief update is still computationally expensive as the dataset of real and hypothesised observations grows. Furthermore, the belief updates are required sequentially, once per MCTS simulation to generate a new leaf node. This greatly slows the MCTS procedure.

In contrast, our method does not explicitly generate these hypothesised GPs and plans only using histories. The child node in Figure 1 still represents the same belief as the GP shown in blue, but only contains the history obtained using a root sampled transition function. We prove the history-GP equivalence, and hence the validity of root sampling in our setting, by showing that the probability of generating a history from the BAMDP is the same for root sampling as it is for maintaining and updating full GP belief at each belief node.

Proposition 2. *Let $\mathcal{P}_\pi^{h_t}(h_{t+\tau})$ be the probability of a history $h_{t+\tau}$ in the BAMDP, starting at history h_t under policy π, when carrying out individual GP belief updates at every stage; and $\tilde{\mathcal{P}}_\pi^{h_t}(h_{t+\tau})$ be the probability of $h_{t+\tau}$ when carrying out GP root sampling. Then, $\mathcal{P}_\pi^{h_t}(h_{t+\tau}) = \tilde{\mathcal{P}}_\pi^{h_t}(h_{t+\tau})$ for all policies π and all histories $h_{t+\tau}$.*

The proof is given in the appendix and is a direct adaptation of Lemma 1 in [21], accounting for the GP posterior over the transition function, and applies to a general stochastic policy $\pi : \mathcal{H}^{\mathcal{M}} \times A \to [0, 1]$.

Once any new real observations have been added to the root node GP, one can draw an arbitrary number of root samples to plan with at little additional computational cost. This means we can run more trials with the same computational budget, thus building a larger MCTS search tree, as we demonstrate in Section 5. Finally, we note that, due to the BO setting of [9, 4], where the objective is directly related to the uncertainty in the function being predicted by the GP, it is not enough to ensure the same distribution over histories. Thus, they must maintain full GP beliefs in their search nodes.

5 Experiments

Domains Description. We experimentally evaluate the proposed method in two simulated domains.

1) Radiation domain: a robot must navigate to a goal location (single-goal variant), or one of three goal locations (multi-goal variant), in a 20m $\times$ 20m reactor room with an unknown distribution of radiation level, while minimising its cumulative exposure. The GP model is log-warped [22] to constrain the predictions to be strictly positive and better model order-of-magnitude variation in radiation level caused by $1/r^2$ "solid angle" radiation physics. Goal locations and radiation distributions are randomly generated and described in full in the appendix. The map (Figure 2a) is discretised into an 8-connected grid with side length 1.0m. The robot pose therefore comprises the known value U-MDP state features: S_k is a finite set of (x, y) locations $\{x, y\} \subseteq S_k$. The radiation level is a single unknown value state feature $S_e = \mathbb{R}_{\geq 0}$ where $rad\_exp \in S_e$ is the level at a location. The reactor room world is from [23] and is used with Gazebo [24] and ROS [25].

2) Ocean currents domain: an autonomous underwater vehicle (AUV) must navigate underwater to one of a set of two to three goal locations (multi-goal variant) or a single goal location (single-goal variant) across a 10km $\times$ 10km map, under the influence of currents. These are drawn from a real-world ocean current dataset and modelled online by a multi-output coregionalised GP [26]. The AUV is simulated by a kinematics, guidance, navigation and control (GNC) model of a small AUV. The robot pose comprises the known value U-MDP state features: S_k is a finite set of (x, y) locations $\{x, y\} \subseteq S_k$ on a 18×20 hexagonal grid of states, giving approximately 500m spacing between states. An example state grid with ground-truth currents is shown in Figure 2b. The unknown value state features $S_e = \mathbb{R}^2$ represent the current x and y velocities $\{v_x, v_y\} \in S_e$. The cost function encodes the expected traversal time between states given the AUV's water-relative velocity and the current vector values. Additional domain details and parameters are given in the appendix.

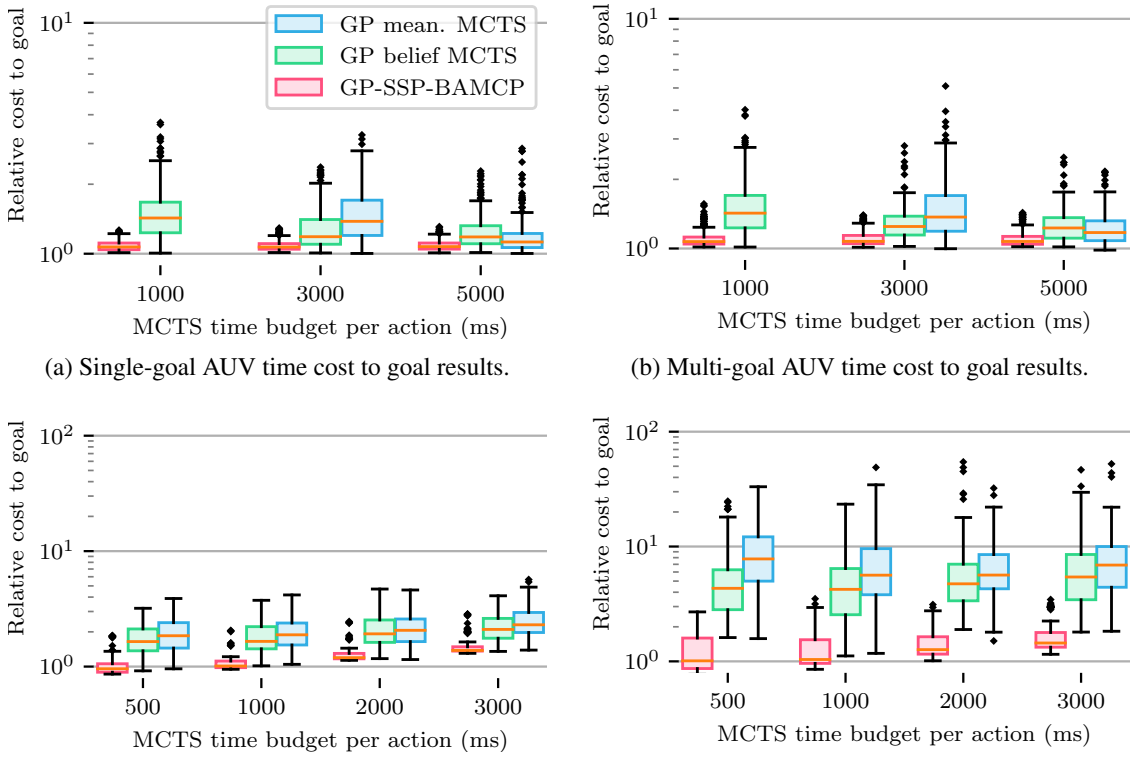
Algorithms. Our online BAMCP variant as described in Algorithm 1 is evaluated against two other GP belief planning algorithm baselines. The performance of sampling-based methods is dependent on the assigned compute budget to select each action, therefore we vary compute budget in each experiment. The *GP mean belief MCTS* algorithm represents a full GP belief planning approach that makes maximum likelihood assumptions. This is similar to [14], but with a GP rather than GPDM model and replacing the Monte Carlo action selection search with MCTS due to the complexity of the search problem. The *GP belief MCTS* algorithm represents the other full GP belief planning approaches [9, 4] which plan with GP beliefs inside the MCTS search tree, but sample a fixed environment dynamics instance for the MCTS rollout. This avoids carrying out belief updates during the rollout, leading to a speed up in MCTS trials per second. For all algorithms the rollout policy is to choose the action that minimises the travel distance from the next state to the closest goal state.

Results. The plots in Figure 3 are from multiple randomly generated problem instances: 10 for the currents domain, and 5 for the radiation domain. Each algorithm/MCTS time budget combination is given 25 repeats. Values are normalised by the expected minimum achievable cost for the corresponding randomly generated problem, calculated using an exact method and full knowledge of the environment. Due to stochasticity in the simulated robot and environment, it is possible for some runs to achieve costs below this value. Experiments are run on a 3.20GHz i7 / 64GB RAM machine.

In the ocean currents domain, Figures 3a and 3b, GP-SSP-BAMCP significantly outperforms the other algorithms. It is capable of achieving close to optimal cost with only 1 second of computation budget. Given an increased computational budget, each algorithm achieves lower cost-to-goal mean and variance. At lower computational budgets, the GP belief MCTS algorithm consistently outperforms the GP mean belief MCTS algorithm, which must carry out a belief update for each step it takes towards a goal state during a rollout. The 1000ms / GP mean belief MCTS combination is not shown, as the algorithm is not capable of building a meaningful plan within that time limit.

GP-SSP-BAMCP also significantly outperforms the baselines in the radiation domain, shown in Figures 3c and 3d. In this case, note that the increase in cost-to-goal with additional computation time is caused by the robot spending more time planning while being exposed to radiation, while the normalisation denominator is kept constant. For all methods additional computation time increases plan quality, but this can be offset by the additional stationary time spent planning. The 3000ms time budget case corresponds to the robot spending $\sim 50\%$ of its runtime planning rather than moving.

As a summary, GP-SSP-BAMCP's higher performance is due to its ability to carry out far more MCTS trials than the two full belief planning methods. On average across all sampled problem scenarios, GP-SSP-BAMCP carries out $\sim 100\times$ more MCTS trials per second than GP mean belief MCTS, and $\sim 40\times$ more than GP belief MCTS.



(a) Single-goal AUV time cost to goal results.

(b) Multi-goal AUV time cost to goal results.

(c) Single-goal radiation exposure cost to goal results.

(d) Multi-goal radiation exposure cost to goal results.

Figure 3: Single- and multi-goal experiment results. Plots consist of 25 simulated experiments for each algorithm/MCTS time budget combination in each randomly generated problem instance for that domain.

6 Conclusion

We have proposed a unified Bayesian RL framework for single-mission robot planning in GP-modelled unknown environments, and demonstrated that we are able to plan more effectively in representative real-world environments than previous approaches are able to. One potential avenue for future work is to apply progressive widening [20] or function approximation [21] techniques to our BAMCP search tree, to determine whether these produce better plans given the continuous GP-BAMDP state-space. Relaxing the assumption of negligible measurement noise would require introducing partial observability alongside transition function uncertainty. The reformulation would transform the BAMDP into a more complex Bayes-adaptive POMDP [27] with continuous GP belief. This would allow us to address settings with very high localisation uncertainty and sensor noise.

Limitations. As with any method using exact GP regression, there is a limit to the size of environment that can practically be modelled. Although root sampling limits the computational cost of GP planning, exact GP regression scales with $O((n+m)^3)$ where n is the number of data points and m is the number of sample points. In future work we aim to replace exact GP predictions with approximate GP posterior sampling [28] to reduce the GP computational burden and improve scalability.

As our evaluation is in simulation there may exist a sim-to-real transfer gap when using the method with a real robot. This concern should be partly alleviated by our use of realistic Gazebo [24] simulation and use of real-world currents data with a complex kinematic AUV simulation. The proposed method is also a higher-level planning approach, meaning that the gap should be less wide than with low-level control methods, which are more sensitive to small sim-to-real changes.

Finally, for some environments or problem settings there is less inherent value in carrying out online planning. For example, with only a single goal state and few feasible routes to that state, the best approach for the radiation domain may be to navigate as quickly as possible to that state without incurring the radiation exposure costs of stopping to plan.

Acknowledgments

This work received EPSRC funding via the "From Sensing to Collaboration" programme grant [EP/V000748/1]. Matthew Budd was supported by an Amazon Web Services Lighthouse scholarship. Paul Duckworth was supported by the Cancer Research UK Radnet Oxford Centre grant (CRUK A28736).

References

[1] Y. Cao, Y. Wang, A. Vashisth, H. Fan, and G. A. Sartoretti. CAtNIPP: Context-aware attention-based network for informative path planning. In *Conference on Robot Learning (CoRL)*, pages 1928–1937. PMLR, 2023.

[2] M. Turchetta, F. Berkenkamp, and A. Krause. Safe exploration in finite Markov decision processes with Gaussian processes. *Advances in Neural Information Processing Systems (NeurIPS)*, 29, 2016.

[3] A. Wachi, Y. Sui, Y. Yue, and M. Ono. Safe exploration and optimization of constrained MDPs using Gaussian processes. In *AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[4] G. Flaspohler, V. Preston, A. P. M. Michel, Y. Girdhar, and N. Roy. Information-guided robotic maximum seek-and-sample in partially observable continuous environments. *IEEE Robotics and Automation Letters (IEEE RA-L)*, 4(4), 2019.

[5] M. Budd, B. Lacerda, P. Duckworth, A. West, B. Lennox, and N. Hawes. Markov decision processes with unknown state feature values for safe exploration using gaussian processes. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7344–7350. IEEE, 2020.

[6] C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[7] A. Guez, D. Silver, and P. Dayan. Scalable and efficient Bayes-adaptive reinforcement learning based on Monte-Carlo tree search. *Journal of Artificial Intelligence Research (JAIR)*, 48, 2013.

[8] S. Jiang, D. Jiang, M. Balandat, B. Karrer, J. Gardner, and R. Garnett. Efficient nonmyopic Bayesian optimization via one-shot multi-step trees. *Advances in Neural Information Processing Systems (NeurIPS)*, 33, 2020.

[9] P. Morere, R. Marchant, and F. Ramos. Sequential Bayesian optimization as a POMDP for environment monitoring with UAVs. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017.

[10] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning (ECML)*, pages 282–293. Springer, 2006.

[11] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[12] M. O. Duff. *Optimal learning: Computational procedures for Bayes-adaptive Markov decision processes.* PhD thesis, Dept. of Computer Science, University of Massachusetts Amherst, 2003.

[13] P. Duckworth, B. Lacerda, and N. Hawes. Time-bounded mission planning in time-varying domains with semi-MDPs and Gaussian processes. In *Conference on Robot Learning (CoRL)*. Journal of Machine Learning Research, 2021.

[14] P. Dallaire, C. Besse, S. Ross, and B. Chaib-draa. Bayesian reinforcement learning in continuous POMDPs with Gaussian processes. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2604–2609. IEEE, 2009.

[15] Y. Engel, S. Mannor, and R. Meir. Reinforcement learning with Gaussian processes. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)*, pages 201–208, 2005.

[16] D. P. Bertsekas and J. N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3), 1991.

[17] Mausam and A. Kolobov. *Planning with Markov decision processes: An AI perspective*. Morgan & Claypool Publishers, 2012.

[18] M. A. Osborne, S. J. Roberts, A. Rogers, S. D. Ramchurn, and N. R. Jennings. Towards real-time information processing of sensor network data using computationally efficient multi-output Gaussian processes. In *International Conference on Information Processing in Sensor Networks (ISPN)*. IEEE, 2008.

[19] D. Silver and J. Veness. Monte-Carlo planning in large POMDPs. *Advances in Neural Information Processing Systems (NeurIPS)*, 23, 2010.

[20] A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard. Continuous upper confidence trees. In *International Conference on Learning and Intelligent Optimization*, pages 433–445. Springer, 2011.

[21] A. Guez, N. Heess, D. Silver, and P. Dayan. Bayes-adaptive simulation-based search with value function approximation. *Advances in Neural Information Processing Systems (NeurIPS)*, 27, 2014.

[22] E. Snelson, Z. Ghahramani, and C. Rasmussen. Warped Gaussian processes. *Advances in Neural Information Processing Systems (NeurIPS)*, 16, 2003.

[23] T. Wright, A. West, M. Licata, N. Hawes, and B. Lennox. Simulating ionising radiation in Gazebo for robotic nuclear inspection challenges. *Robotics*, 10(3):86, 2021.

[24] N. Koenig and A. Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2149–2154. IEEE, 2004.

[25] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, et al. ROS: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, 2009.

[26] C. Williams, E. V. Bonilla, and K. M. Chai. Multi-task Gaussian process prediction. *Advances in Neural Information Processing Systems (NeurIPS)*, 20, 2007.

[27] S. Katt, F. A. Oliehoek, and C. Amato. Learning in POMDPs with Monte Carlo tree search. In *International Conference on Machine Learning (ICML)*, pages 1819–1827. PMLR, 2017.

[28] J. Wilson, V. Borovitskiy, A. Terenin, P. Mostowsky, and M. Deisenroth. Efficiently sampling functions from Gaussian process posteriors. In *International Conference on Machine Learning (ICML)*. PMLR, 2020.

[29] M. C. Kennedy and A. O'Hagan. Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(3):425–464, 2001.

[30] G.-A. Fuglstad, D. Simpson, F. Lindgren, and H. Rue. Constructing priors that penalize the complexity of gaussian random fields. *Journal of the American Statistical Association*, 114 (525):445–452, 2019.

[31] S. Marmin and M. Filippone. Deep gaussian processes for calibration of computer models (with discussion). *Bayesian Analysis*, 17(4):1301–1350, 2022.

[32] N. Cressie and G. Johannesson. Fixed rank kriging for very large spatial data sets. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 70(1):209–226, 2008.

[33] D. Nguyen-Tuong and J. Peters. Using model knowledge for learning inverse dynamics. In *2010 IEEE international conference on robotics and automation*, pages 2677–2682. IEEE, 2010.

[34] B. Bischoff, D. Nguyen-Tuong, H. van Hoof, A. McHutchon, C. E. Rasmussen, A. Knoll, J. Peters, and M. P. Deisenroth. Policy search for learning robot control using sparse data. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3882–3887. IEEE, 2014.

[35] P. Tighineanu, K. Skubch, P. Baireuther, A. Reiss, F. Berkenkamp, and J. Vinogradska. Transfer learning with gaussian processes for bayesian optimization. In *International conference on artificial intelligence and statistics*, pages 6152–6181. PMLR, 2022.

[36] Z. Wang, G. E. Dahl, K. Swersky, C. Lee, Z. Nado, J. Gilmer, J. Snoek, and Z. Ghahramani. Pre-trained gaussian processes for bayesian optimization. *Journal of Machine Learning Research*, 25(212):1–83, 2024.

5.1 Additional Discussion

Similar considerations about the choice of GP kernel and hyperparameter priors apply to this chapter as in the previous chapter, as discussed in Section 4.1. We can, however, define and update GP hyperparameter priors, since we do not face the same non-i.i.d. data issue arising from safety constraints as in the previous chapter.

A contemporaneous work (Cao et al., 2023) uses deep RL as a core component of an IPP method. Similarly to other IPP methods, their focus is not on environment dynamics which affect a situated agent, but on information gain about the environment. Their method is able to avoid GP belief updates in a search tree, which is a significant computational bottleneck in MCTS-based IPP methods such as those we compare to. However, the method requires extensive training on specific GP hyperparameters. Changes to hyperparameters require retraining the model, at significant computational cost. Our method is able to work with broad GP hyperparameter priors, making it better able to adapt to a wider range of environments. In addition to the offline training requirement, their method still relies on online computation, making it similar to ours in that respect.

Because our method uses sample-based simulation for planning, it shares similarities with the concept of hindsight optimisation. At a high level, hindsight optimisation techniques sample multiple simplified versions of a problem, solve these, and combine the solutions to generate a good solution to the original problem. A well-known example is probabilistic planning via hindsight determinisation, where multiple possible determinised versions of the original probabilistic problem are sampled and solved with a deterministic planner (Yoon et al., 2008).

A related algorithm is QMDP (Littman et al., 1995), which solves POMDPs and therefore could straightforwardly be adapted to solving BAMDPs. QMDP solves the "underlying" MDP of the POMDP, assuming full observability, and defines the Q-value of a POMDP belief as the expected value of the belief-weighted MDP Q-values. The approach therefore assumes that all state uncertainty will be resolved after a single action, making it overly optimistic and unable to consider the value

of information-gathering actions. Applying QMDP to BAMDPs would require solving the underlying MDP for all possible transition functions, which is likely to be infeasible for practical problems such as those we consider in this chapter.

Similar ideas are employed by more scalable algorithms, such as Thompson sampling (Russo et al., 2018) and posterior sampling (Osband and Van Roy, 2017). These methods sample one or several transition functions from the posterior distribution, solve the resulting MDPs (or use them to train an RL agent, in the RL context), and select the best overall action across the samples. Posterior sampling achieves state-of-the-art regret bounds (Osband and Van Roy, 2017) where the setting allows multiple episodes of interaction with the same environment, but in the few- or single-episode setting, Bayesian RL methods such as BAMCP tend to perform better (Guez et al., 2014b).

MCTS-based planning methods, such as BAMCP, offer additional advantages over posterior sampling planning methods in our setting. The MCTS search tree focuses search effort on the most promising parts of the state space, and generally avoids constructing overly deep trees that extend beyond the generalisation horizon of the GP belief. Our method samples a large number of possible transition functions from the GP posterior, and uses each for a single Monte-Carlo simulation. In contrast, posterior sampling methods fully solve a small number of sampled MDPs, which can lead to overfitting to the sampled transition functions and to wasting computational effort on planning in regions of the state space with highly uncertain dynamics. This is particularly true for Thompson sampling, which only samples a single transition function from the posterior.

5.2 Limitations and Future Work

The PKSMDP/U-MDP model is referred to as *partially known* (or, equivalently, partially unknown) because although the environment dynamics are uncertain, the possible agent states where the agent experiences the dynamics are pre-specified.[1]

[1]In the safe exploration work, this set of locations is expanded as the agent explores the environment, but is fixed during the new goal selection routine.

In the previous chapter, possible agent states were defined by the waypoints in a topological map. In this chapter, we have grids of agent states defined over the robot's operating area. This pre-specification of the sample points for the GP is what allows us to efficiently root sample from the GP posterior.

The current approach uses standard Cholesky decomposition for GP inference, which incurs cubic complexity in the number of data/sample points (Rasmussen and Williams, 2006). The number of sample points (and therefore the size of the agent state space) is therefore practically limited to a few thousand points. Sparse GPs (Snelson and Ghahramani, 2005), or other GP approximations discussed in Section 2.2, could be used to improve scalability. In particular, if agent states are defined in a grid, Toeplitz structure could be exploited to significantly reduce the complexity of the GP posterior computation (Wilson et al., 2015). Neural process models (Garnelo et al., 2018) could also offer a more scalable alternative to GPs, but have not been proven in the adaptive decision-making context.

Along similar lines, the requirement to pre-specify sample points to enable root sampling makes extending to the continuous state/action space setting difficult. The simplest approach would be to define a grid of sample points and linearly interpolate between them. Issues would occur if the sample points are not sufficiently dense compared to the environment dynamics lengthscales, or if the environment dynamics are high dimensional. There exist methods for fast sampling from the posterior GP distribution, such as (Wilson et al., 2020), which may be able to augment or replace the root sampling method. As well as GP sampling considerations, the continuous state/action space setting would require a continuous action space MCTS method, which are well studied in the literature (Sunberg and Kochenderfer, 2018; Lim et al., 2021).

Lifting the assumption of negligible observation noise would make the method more practically applicable to real-world problems. The GP model already incorporates observation noise in the GP likelihood, but the decision-making method would need to move from a BAMDP method to a Bayes-adaptive POMDP (BA-POMDP) method (Ross et al., 2007; Katt et al., 2017). BA-POMDPs are more complex

than BAMDPs, due to the additional need to reason about partial observability. However, for practical sensors with relatively low noise, this would not greatly affect the computational tractability of our method.

Some domains have a time-varying aspect, which would require a spatiotemporal GP to model. Our method is straightforwardly applicable to this setting, by including time as an additional state feature, but we do not carry out experiments in time-varying problem settings. The limited timescale of one mission means it is difficult to learn a GP that captures the time-varying dynamics of the environment. More prior information on temporal dynamics of the problem would be needed to make this tractable.

Finally, it is worth noting that although our method is a Bayesian model-based RL method, we do not use the GP belief in a truly Bayesian manner. A true Bayesian treatment with respect to hyperparameter priors would require marginalising over the hyperparameters, rather than taking a Maximum a Posteriori (MAP) estimate as we do, and may result in significantly better-calibrated GP posteriors (Svensson et al., 2015). Methods exist for carrying out this marginalisation, such as sequential Monte Carlo (Svensson et al., 2015) and nested sampling (Simpson et al., 2021). However, these methods would lose some of the computational benefit of root sampling, as the posterior GP would need to be recomputed for each hyperparameter sample. The MAP estimate is a very commonly used approximation (Rasmussen and Williams, 2006), so we do not consider this a significant limitation.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Bayesian Reinforcement Learning for Single-Episode Missions in Partially Unknown Environments
Publication Status	☑ Published ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and unsubmitted work written in a manuscript style
Publication Details	**Matthew Budd**, Paul Duckworth, Nick Hawes, and Bruno Lacerda. Bayesian reinforcement learning for single-episode missions in partially unknown environments. In *Conference on Robot Learning*. PMLR, 2023. URL https://proceedings.mlr.press/v205/budd23a/budd23a.pdf.

Student Confirmation

Student Name:	Matthew Budd		
Contribution to the Paper	• I developed the approach proposed in the paper. • I implemented the method and tested the system in simulation. • I wrote the paper with review and input from Bruno Lacerda and Nick Hawes.		
Signature		Date	April 24, 2025

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title:	Professor Nick Hawes		
Supervisor comments	I confirm that Matthew made a substantial contribution to the publication, and that the description above is accurate.		
Signature		Date	April 25, 2025

This completed form should be included in the thesis, at the end of the relevant chapter.

6 Metareasoning for Offline Decision-Making

	Chapter 3	Chapter 4	Chapter 5	Chapter 6	Chapter 7
Epistemic uncertainty	Environment dynamics	Environment dynamics	Environment dynamics	Planner dynamics	Planner dynamics
Mode	Offline	Replanning	Online	Offline	Online
Model	Timed MDP	EstMDP	BAMDP	Metalevel MDP	Metalevel MDP
Solution method	VI (max reward)	VI (reachability)	MCTS	Deep RL	Deep RL
Data regime	Small/ medium	Small	Small	Large	Large

Table 1.1: Settings, methods and features for each chapter.

The previous three chapters described decision-making methods for situated agents in epistemically uncertain environments. These methods reason about possible outcomes of the agent taking actions in the environment. Their behaviour and performance are often dependent on algorithm hyperparameters, and the most appropriate hyperparameter values depend on the specific problem instance the agent is operating in. However, most commonly these hyperparameters are fixed at deployment time, potentially based on a hyperparameter tuning process.

We now turn our attention to the problem of controlling the decision-making process itself. Rather than using a fixed decision-making approach, we are interested in how to select the best decision-making approach for a given problem instance. This chapter and the next chapter develop novel metareasoning methods for controlling decision-making processes in the context of situated agents. The metalevel controller agent manages decision-making by selecting and switching

hyperparameters of the object-level algorithm, and deciding when to execute the object-level algorithm's best solution.

We present a metalevel control method for offline decision-making algorithms. In this setting, a single period of decision-making is followed by a period of execution of the planned actions, as shown in Figure 1.1a. Examples of offline decision-making algorithms include TMIT* (Thomason et al., 2022) for task and motion planning (TAMP), algorithms for motion planning under uncertainty (Bry and Roy, 2011) and the pre-planning method in Chapter 3. Metareasoning methods for the offline setting exist for the TAMP problem (Sung et al., 2021), temporal planning with durative actions (Cashmore et al., 2018), and heuristic search (Burns et al., 2013). However, these methods apply only to deterministic search algorithms, and correspondingly assume that the quality of the current object-level solution is always known. This is not the case when policies are deployed in uncertain environments. Even when the model is known, calculating the quality of the solution using policy evaluation is computationally expensive.

In the offline setting of Chapter 3, the decision-making process takes place before the agent is deployed in the environment. However, the combination of the environment, agent and planning computer still forms a decision-making and execution system. The entire system exhibits a trade-off between the time spent generating plans and the time spent executing those plans. Using more complex models and sampling more possible epistemic parameters leads to better-performing policies at the cost of longer computation times. The user must therefore choose between better-performing policies that spend overall less time in the water, and simpler policies that can be deployed to the AUV more quickly.

We relax a key assumption from previous metareasoning works, making our method applicable to probabilistic decision-making methods without requiring strong simplifying assumptions that limit performance. This makes our method the first non-myopic metareasoning method for probabilistic decision-making algorithms.

The method is agnostic to the object-level algorithm, and learns from the decision-making traces of the object-level algorithm. Some previous metareasoning methods

need to be trained on full solution traces, requiring the object-level algorithm to be run to full convergence (Hansen and Zilberstein, 2001; Sung et al., 2021). By training an RL policy in the loop with the object-level algorithm, our method avoids running the object-level algorithms past the point of diminishing returns for the metareasoning objective. By using an RL policy for the metalevel agent, the added computational overhead of carrying out metareasoning is minimal.

Experimental evaluation shows the trained metalevel agent outperforming baseline methods in performance, and acting rationally in the face of resource constraints. When the cost of decision-making is high relative to problem complexity, the metalevel policy learns to use hyperparameters that emphasise fast, lower-quality solutions, then quickly act on that solution. When the cost of decision-making is low relative to problem complexity, the metalevel policy reasons for longer before execution and uses appropriate hyperparameters for high-quality solutions. Which hyperparameter values are appropriate for a given problem instance is learned from the decision-making traces of the object-level algorithm on the training problem distribution.

Citation:

Matthew Budd, Bruno Lacerda, and Nick Hawes. Stop! Planner Time: metareasoning for probabilistic planning using learned performance profiles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20053–20060, 2024

The Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI-24)

Stop! Planner Time: Metareasoning for Probabilistic Planning Using Learned Performance Profiles

Matthew Budd, Bruno Lacerda, Nick Hawes

Oxford Robotics Institute, University of Oxford
{mbudd, bruno, nickh}@robots.ox.ac.uk

Abstract

The metareasoning framework aims to enable autonomous agents to factor in planning costs when making decisions. In this work, we develop the first non-myopic metareasoning algorithm for planning with Markov decision processes. Our method learns the behaviour of anytime probabilistic planning algorithms from performance data. Specifically, we propose a novel model for metareasoning, based on contextual performance profiles that predict the value of the planner's current solution given the time spent planning, the state of the planning algorithm's internal parameters, and the difficulty of the planning problem being solved. This model removes the need to assume that the current solution quality is always known, broadening the class of metareasoning problems that can be addressed. We then employ deep reinforcement learning to learn a policy that decides, at each timestep, whether to continue planning or start executing the current plan, and how to set hyperparameters of the planner to enhance its performance. We demonstrate our algorithm's ability to perform effective metareasoning in two domains.

Introduction

Rational agents should reason about the consequences of their actions. However, this reasoning process itself has its own consequences: thinking is a physical process that requires time and energy. In many systems this issue can be ignored, in particular where the costs of any computation are negligible compared to the costs of actions. In other systems, the real-world effects of reasoning cannot be discounted. As an example, for mobile robots thinking and acting are often closely coupled. An autonomous vehicle can use its finite battery capacity on its on-board computer or its motors. To what extent is the energy saved by executing a better motion plan offset by the energy cost of additional computation? This scenario is illustrated in Figure 1. Of course, this assumes that the agent is reasoning *online*. If it is able to pre-plan all of its tasks offline, planning effort is negligible at runtime and no metareasoning tradeoff exists.

Over the last 40 years, the field of rational metareasoning has developed bounded optimal algorithms for reasoning about these tradeoffs (Russell and Wefald 1991; Cox and Raja 2011; Hay et al. 2012). Recent problem settings have included motion planning and heuristic search (Burns, Ruml, and Do 2013; Sung, Kaelbling, and Lozano-Pérez 2021; Bhatia et al. 2022). Optimal exact metareasoning is polynomially harder than reasoning (Lin et al. 2015), so all practical algorithms carry out approximate metareasoning. A common simplification is to only consider the immediate effects of a single reasoning step: this is known as the *meta-greedy* or *meta-myopic* assumption (Russell and Wefald 1989, 1991).

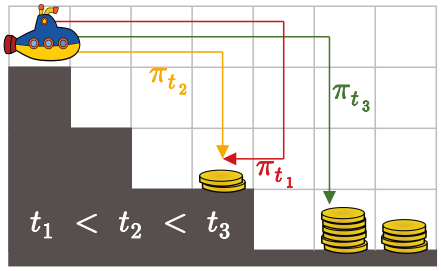


Figure 1: A deep sea treasure problem. Each policy represents the results of planning for a longer duration, and achieves reward indicated by the number of coins. π_{t_2} would be optimal overall if the extra reward gained by π_{t_3} is less than the cost (duration) of the additional planning time.

Under the umbrella of metareasoning problems, we focus specifically on metalevel control of anytime algorithms (Cox and Raja 2011). In this setting, reasoning is carried out by an *object-level* process, which is an anytime interruptible algorithm. This class of algorithms provides its current solution at any time during computation, but the longer the algorithm runs the better the solution will be. The object-level process is supervised by a *meta-level* process, which observes the object-level algorithm's state. Metalevel actions consist of allowing object-level computation to continue, or terminating the algorithm and acting using its current solution.

We wish to apply metareasoning to probabilistic planning with Markov decision processes (MDPs). Recent works have used deep reinforcement learning (RL) to learn non-myopic metalevel policies (Sung, Kaelbling, and Lozano-Pérez 2021; Bhatia et al. 2022). However, these works apply only to object-level algorithms where the current solution quality is known exactly, for example the current minimum path length in RRT*. Relaxing the solution quality observability

The Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI-24)

assumption is one of the main contributions of this paper and is crucial for the class of probabilistic planning algorithms which we consider, since they often maintain only an approximation of the solution quality. We go on to develop a novel contextual metalevel MDP formulation which enables an RL agent to learn to infer solution quality from limited information about the planner and problem instance.

Compared to existing MDP planning metareasoning works, we avoid meta-greedy approximations and instead directly learn the value function of the metalevel MDP. Our method is agnostic to the choice of object-level algorithm, rather than relying on properties of one specific algorithm as Lin et al. (2015) do. Finally, we use the hyperparameter tuning method proposed by Bhatia et al. (2022) to give the metalevel agent more control over the object-level algorithm. This enables switching between parameters that emphasise fast but low-quality solutions vs slow and high-quality solutions, alongside deciding when to execute the current solution.

Related Work

The metalevel control problem is commonly posed as a metalevel Markov decision process (Hay et al. 2012). Closest to our setting, Lin et al. (2015) formulate online stochastic shortest path (SSP) MDP planning metareasoning as a metalevel MDP. In common with many metareasoning algorithms, they are unable to exactly solve this model and must use a meta-greedy approximation (Russell and Wefald 1991). This is the simplifying assumption that the reasoning agent can only choose between executing now, or executing after a single additional reasoning step. The approximation ignores the value of further computation, but can be shown to be optimal in the case of diminishing returns and non-decreasing computation cost (Callaway et al. 2018). However, solution quality plots from real algorithms and problems rarely show smoothly diminishing returns. One example from motion planning is the sharp change caused by a change of homotopy class.

One way to achieve non-myopic metalevel behaviour is to build a probabilistic model of the algorithm's solution quality evolution from a dataset, and use this *performance profile* to determine the optimal execution point (Hansen and Zilberstein 2001). As long-horizon model-based planning is computationally expensive, the approach is limited to short horizons to avoid excessive metareasoning overhead. Recent works improve on this approach by learning the performance profiles, and metalevel control policies, from experience (Bhatia et al. 2022; Sung, Kaelbling, and Lozano-Pérez 2021). These works focus on deterministic search problems, and make metalevel decisions based on a fixed utility function that combines the object-level solution quality and computation time. This contrasts with the setting of Lin et al. (2015), where the effects and costs of reasoning depend on the current object-level MDP state. By defining utility as a function of object-level solution quality, these methods assume that the current solution quality is always known. In this work we relax that assumption in order to apply deep RL-based metareasoning to probabilistic planning.

Callaway et al. (2018) present a method that estimates the value of computation using a weighted combination of myopic and full-knowledge value of information (VOI) features. These weights are learned from an object-level problem distribution using Bayesian optimisation. Their algorithm outperforms those that depend only on myopic VOI features. However, the method assumes perfect knowledge of the metalevel MDP dynamics, which is infeasible for a practical planning algorithm. It also scales poorly with increasing complexity of the object-level problem, due to the online calculations required to compute the values of VOI features.

Indeed, Callaway et al. note that metareasoning is only useful when object-level reasoning is significantly more expensive than metareasoning. It is counterproductive to spend more time deciding whether to think than actually thinking. Other work which explicitly analyses the overhead of metareasoning (Milli, Lieder, and Griffiths 2017) concurs with this point. This explains the strong simplifying assumptions commonly used in metareasoning algorithms, and justifies using deep RL-trained policies for these problems: the cost of querying actions from a trained policy is minimal.

A different but related problem setting is situated temporal planning, which aims to maximise the probability of finding a valid deterministic plan within a deadline (Cashmore et al. 2018). Rather than a metalevel agent deciding when to execute, metareasoning takes place on the level of deciding which nodes to expand in a search tree. Some more recent extensions also attempt to minimise the cost-to-goal of the chosen plan (Shperberg et al. 2020). However, these methods do not directly minimise the combination of planning cost and execution cost as ours does.

Preliminaries

Markov Decision Processes. A stochastic shortest path (SSP) MDP is a tuple $\mathcal{M} = \langle S, \mathit{init}, A, T, C, G \rangle$, where S is a finite set of states; $\mathit{init} : S \to [0, 1]$ is an initial state distribution; A is a finite set of actions; $T : S \times A \times S \to [0, 1]$ is a probabilistic transition function; $C : S \times A \to \mathbb{R}_{\geq 0}$ is a cost function; and $G \subset S$ is a set of absorbing, zero-cost goal states.

When the initial state is deterministic, we will replace *init* with s_0 in the definition of $\mathcal{M}$. Actions are selected using a stationary policy $\pi : S \to A$. A policy is *proper* in state s if the probability of reaching a goal state when starting from s is 1. For an SSP MDP there must exist a proper policy from the initial state. All improper policies must have infinite expected cumulative cost from states that they are improper in. It can be shown that there exists a minimum cost proper policy (Mausam and Kolobov 2012). A policy is *complete* if it provides an action for all states $s \in S$. Alternatively, a *partial* policy π has a domain $S_\pi \subset S$ and is complete in S_π. For a proper policy π, let $V^\pi_{\mathcal{M}} = \mathbb{E}_{\mathcal{M}, s_0 \sim \mathit{init}} [\sum_{i=0}^{\infty} C(s_i, \pi(s_i))]$, where s_i is a random variable representing the state visited at the i-th timestep, denote the expected cumulative cost of executing policy π.

Problem Formulation

In this section, we pose our problem as minimising the expected cost of a metalevel SSP MDP $\mathcal{M}^M$ that observes and controls the behaviour of an object-level probabilistic planner. Similarly to previous work (Milli, Lieder, and Griffiths 2017;

The Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI-24)

Sung, Kaelbling, and Lozano-Pérez 2021; Bhatia et al. 2022), we assume that object-level problems are drawn from a distribution $p(\mathcal{M})$ over a set of decision problems D. Specifically, the planner operates on an object-level SSP MDP instance $\mathcal{M} = \langle S, s_0, A, T, C, G\rangle$, where $\mathcal{M}$ is sampled according to $p(\mathcal{M})$. The object-level planner has a set of hyperparameters $\{\Delta_i\}_{i=1}^{N_\Delta}$, which alter its behaviour. Each hyperparameter has a set of valid values $\Delta_n = \{\delta_1, \delta_2, \ldots, \delta_{k_n}\}$.

The metalevel agent has no direct control over the actions taken in the object-level MDP. At each metalevel timestep, the metalevel agent observes the object-level planner's configuration (its internal state) χ and chooses a metalevel action. Let π_χ be the object-level planner's current best solution, represented within its current configuration χ. Metalevel actions are to a) continue planning for another timestep, altering hyperparameter values if desired, or b) stop planning and execute the current best solution π_χ. Continuing planning for one timestep runs the planner for time τ (which may correspond to many object-level planning iterations), and incurs a fixed instance-dependent planning cost $\lambda(\mathcal{M})$.

The object-level planner's current best policy π_χ may not be complete for all states reachable under that policy from s_0. We assume that π_χ is combined with a *default policy* which is complete and proper in state s_0. This new complete and proper policy, $\tilde{\pi}_\chi$, follows π_χ when it is defined, and the default policy when outside of π_χ's support.

We now define our metalevel reasoning model.

Definition 1 *Given a stochastic shortest path object-level MDP $\mathcal{M} = \langle S, s_0, A, T, C, G\rangle$, the metalevel MDP is an SSP MDP $\mathcal{M}^M = \langle S^M, s_0^M, A^M, T^M, C^M, G^M\rangle$, where:*

- $S^M = X \cup \{\text{DONE}\}$ *where X is the set of all possible configurations of the object-level planner and* DONE *is a terminal state;*
- $s_0^M = \chi_0$ *where χ_0 is the initial configuration of the object-level planner;*
- $A^M = (\{\text{PLAN}\} \times \Delta_1 \times \ldots \times \Delta_{N_\Delta}) \cup \{\text{EXEC}\}$*, i.e. the agent may choose to let the planner run for another metalevel timestep (τ time in planner time) using the specified hyperparameter values, or execute the current best solution;*
- $T^M : S^M \times A^M \times S^M \to [0, 1]$ *is defined as follows:*

$$T^M(\chi, a, \chi') = \begin{cases} 1 & \text{if } \chi \in X, \\ & \chi' = \text{DONE and} \\ & a = \text{EXEC} \\ p(\chi' \mid \mathcal{M}, \chi, a) & \text{if } \chi, \chi' \in X \text{ and} \\ & a \neq \text{EXEC} \\ 0 & \text{otherwise}, \end{cases}$$

where $p(\chi' \mid \mathcal{M}, \chi, a)$ is the probability of the object-level planner transitioning to configuration χ', given that it was in configuration χ and ran on the object-level MDP $\mathcal{M}$ for one metalevel timestep, using the hyperparameters specified by a;

- $C^M : S \times A \to \mathbb{R}_{\geq 0}$ *is defined as follows:*

$$C^M(\chi, a) = \begin{cases} \lambda(\mathcal{M}) & \text{if } a = \text{PLAN} \\ V_{\mathcal{M}}^{\tilde{\pi}_\chi} & \text{if } a = \text{EXEC}, \end{cases}$$

where $\lambda(\mathcal{M})$ is the cost of planning for one metalevel timestep in MDP $\mathcal{M}$ and π_χ is the object-level planner's current policy; and

- $G^M = \{\text{DONE}\}$.

A cost-minimising metalevel policy $\pi^M : S^M \to A^M$ for $\mathcal{M}^M$ minimises the sum of expected total planning cost and expected object-level policy execution cumulative cost. In the definition of $\mathcal{M}^M$, we assume that X contains all the internal features necessary to build Markovian internal state dynamics. These dynamics might still be probabilistic because the object-level algorithm itself will typically have a probabilistic nature (e.g. successor sampling in trial-based search). Furthermore, to ensure that the metalevel cost minimisation objective is meaningful, we assume that the cost of thinking $\lambda(\mathcal{M})$ uses the same unit as the object-level MDP cost. For example, the metalevel problem might be minimising total (planning + execution) time or energy spent.

Note that $V_{\mathcal{M}}^{\tilde{\pi}_\chi}$ can be interpreted as the *quality profile* of the anytime planner, i.e., given a planner configuration, it provides a measure of the quality of the solution. With full knowledge of $p(\chi' \mid \mathcal{M}, \chi, a)$ and $V_{\mathcal{M}}^{\tilde{\pi}_\chi}$, the metalevel MDP could be solved to find an optimal metalevel policy. However, two key factors make this infeasible.

Firstly, for practical planners, the configuration space X is prohibitively large, making offline solution impractical. Online solution raises the spectre of metareasoning overhead. Even more critically, the configuration dynamics depend on the MDP instance $\mathcal{M}$. Evaluating the transition dynamics of X on $\mathcal{M}$ is at least as hard as solving $\mathcal{M}$ itself, negating the value of performing metareasoning (Lin et al. 2015).

Secondly, although the configuration includes a representation of the current policy $\tilde{\pi}_\chi$, the expected cost $V_{\mathcal{M}}^{\tilde{\pi}_\chi}$ of this solution is not necessarily readily accessible. Many MDP solution algorithms simultaneously carry out policy improvement and policy value estimation. They will only precisely calculate the expected cost of the optimal policy at convergence. Of course, the current policy can be evaluated on the current MDP instance, but this must be done sparingly to avoid metareasoning overhead. Next, we propose an abstraction of the metalevel MDP which addresses these issues.

Method

Abstracting the Metalevel MDP

Inspired by Bhatia et al. (2022), we begin by abstracting X to Ω, where $\Phi_X : X \to \Omega$ provides a smaller representation of *algorithm features*. Features $\Phi_X(\chi)$ are hand-designed and algorithm specific, and aim to act as surrogates for the full configuration χ. Examples include the total number of trials or state expansions carried out, the depth of a Monte-Carlo search tree, statistical properties of the lengths of sampled trajectories, or value function estimates.

To learn a policy that is good in expectation across the space of all MDPs in the domain D of object-level problems, we extend the state-representation of the abstracted metalevel MDP to include a *context vector* representing the object-level instance $\mathcal{M}$ being solved. By learning the correlations between context vector values and algorithm performance, we

The Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI-24)

can approximate the effects of the object-level MDP instance on $p(\chi' \mid \mathcal{M}, \chi, a)$. The information contained in the context vector is based on what the system could be reasonably expected to know about the problem it is solving, and thus is problem-dependent. For example, in our race track domain experiments, the context vector contains the maximum speed and probability of acceleration failure for the agent's car. We denote the context set as Ψ, and define the function $\Phi_D : D \to \Psi$ which maps an MDP $\mathcal{M}$ to its context vector $\Phi_D(\mathcal{M})$. The cost of thinking $\lambda(\mathcal{M})$ is also included in the context vector, as it is also a constant property of the object-level MDP instance. The context vector can be treated as a part of the state of the abstract metalevel MDP which is set in the initial state according to $p(\mathcal{M})$, and remains fixed during execution.

We can now define our abstract metalevel MDP:

Definition 2 *The abstract metalevel MDP is defined as* $\hat{\mathcal{M}}^M = \langle \hat{S}^M, \hat{init}_0^M, A^M, \hat{T}^M, \hat{C}^M, G^M \rangle$*, where:*

- $\hat{S}^M = \Omega \times \Psi \cup \{\text{DONE}\}$*, i.e., a state is either of the form* (ω, ψ) *where* ω *is the current value of the algorithm features and* ψ *is a context vector representing the object-level MDP being considered; or is the final state* DONE *which represents sending the current object-level policy for execution;*
- *For* $s = (\omega, \psi) \in \hat{S}^M$*:*

$$\hat{init}_0(s) = \begin{cases} \int_{\{\mathcal{M} \mid \Phi_D(\mathcal{M}) = \psi\}} p(\mathcal{M}) d\mathcal{M} & \text{if } \omega = \omega_0 \\ 0 & \text{otherwise,} \end{cases}$$

  where ω_0 *are the planner initial feature values (i.e. the initial context vector value distribution is calculated via marginalisation over* $p(\mathcal{M})$*);*
- *As with the metalevel MDP,* $\hat{T}^M$ *moves* $\mathcal{M}$ *to state* DONE *with probability 1 when action* EXEC *is selected. For* $s = (\omega, \psi), s' = (\omega', \psi) \in \hat{S}$ *and* $a \in A^M$*,* $\hat{T}^M(s, a, s')$ *is the probability of the planner moving to a configuration* χ' *such that* $\Phi_X(\chi') = \omega'$*, given that its algorithm features were* ω *and it ran for one metalevel timestep, using the hyperparameters specified by* a*;*
- *The cost function* $\hat{C}^M$ *is only action-dependent and defined as* C^M*.*

Note that the action space and goal state of the abstract metalevel MDP are the same as the metalevel MDP in Def. 1. Its behaviour is similar to the metalevel MDP, but it (i) operates over a small set of algorithm features rather than its full internal configuration, to achieve scalability; and (ii) considers all possible MDPs in D using a context vector which is fixed during execution and distributed in the initial state according to $p(\mathcal{M})$. We do not have access to a closed-form definition of $\hat{T}^M$. Hence, next we propose a general deep RL algorithm that learns the value function and optimal policy for $\hat{\mathcal{M}}$. This approach assumes that the transition function remains Markovian when defined over a state-space based on abstraction Φ_X. In future work, we will investigate how to relax this assumption by allowing the agent to reason using a history of its past actions and observations, e.g., using a recurrent, LSTM or transformer architecture.

Algorithm 1: General deep RL on the abstract metalevel MDP

Input: Problem distribution $p(\mathcal{M})$, object-level algorithm PLANNERALG, default policy $\tilde{\pi}$, number of training episodes M_e, object-level planning time per metalevel timestep τ
Output: Trained policy π_θ^M.

```
Initialise π_θ^M randomly
for episode = 1, ..., M_e do
    Sample M from D according to p(M)
    ψ ← calculate MDP context Φ_D(M)
    χ ← initial configuration of PLANNERALG for M
    ω ← calculate features Φ_X(χ)
    ŝ ← (ω, ψ)
    repeat {run episode}
        â ← select action using π_θ^M(ŝ)
        if â = EXEC then
            π̃_χ ← best solution from current configuration
                  χ , completed by π̃
            ĉ ← estimate V_M^{π̃_χ}
                using Monte-Carlo simulation
            ŝ' ← DONE
        else {â = (PLAN, δ_1, ..., δ_{N_Δ})}
            χ ← update hyperparameters to δ_1, ..., δ_{N_Δ}
            χ ← run planner for time τ
            ĉ ← λ(M)
            ω ← calculate new features Φ_X(χ)
            ŝ' ← (ω, ψ)
        end if
        Store (ŝ, â, ĉ, ŝ')
        ŝ ← ŝ'
    until ŝ = DONE
    Update parameters of π_θ^M using stored episode data
end for
return π_θ^M
```

Unlike previous metalevel MDP formulations (Sung, Kaelbling, and Lozano-Pérez 2021; Bhatia et al. 2022), the state of our abstract metalevel MDP does not include the performance of the current policy. Thus, we cannot use a closed-form cost function based on a utility function that combines the current time and solution quality, as those methods do. Instead, we must learn the cost function from experience, using evaluations of the final policy cost at execution. This means that we are implicitly learning contextual and feature-based performance profiles for the planner algorithm.

Deep RL for the Abstract Metalevel MDP

An algorithm for training the RL metareasoning agent is given in Algorithm 1. To simplify the presentation, we abstract away the specifics of the exact RL algorithm used, although the algorithm structure is based on that proposed by Bhatia et al. (2022), which is similar to DQN (Mnih et al. 2015). Adapting our approach to an actor-critic framework is straightforward. Specifically, replacing DQN with an algorithm that supports a hybrid action space, e.g. Hybrid SAC (Delalleau et al. 2019), would enable continuous hyperparameter tuning alongside the discrete plan/act actions.

In Algorithm 1, each training episode corresponds to solv-

The Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI-24)

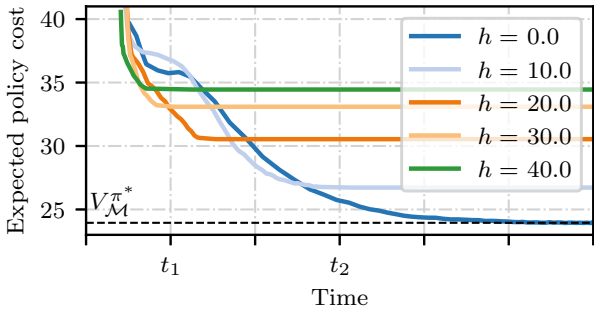


Figure 2: The effect of differing heuristic weights on solution convergence for WBRTDP, on a racetrack problem instance.

ing and then executing a policy in a single problem instance sampled from the problem distribution $p(\mathcal{M})$ (line 3). A new object-level planner is initialised for each problem instance (line 5). In line 7 the initial state contains the initial values of the planner features plus the context vector $\Phi_D(\mathcal{M})$ for the sampled problem, which does not change during the episode. At each timestep, the metalevel agent can choose to execute the current best object-level policy (completed by the default policy) $\tilde{\pi}_\chi$, (line 10) or to continue planning (line 14).

If the agent chooses to execute, we estimate the expected cost of the completed policy $\tilde{\pi}_\chi$, by sampling N trajectories of $\mathcal{M}$ under $\tilde{\pi}_\chi$, and taking their average cumulative cost (line 12). With this final cost, the metalevel MDP transitions to its final absorbing state DONE (line 13) and the episode ends (line 23). If the agent chooses to continue planning, the action specifies the object-level planner hyperparameters to use for the next timestep (line 15). These are used by the object-level planner to plan for a single metalevel timestep (line 16) which, we recall, can correspond to many iterations of the object-level planner. The thinking step costs $\lambda(\mathcal{M})$ (line 17), and causes the features $\Phi_X(\chi)$ to evolve (line 18). This is reflected in the updated state (line 19).

Experienced state transitions are used to update the learned policy (line 24), depending on the specific RL algorithm. As the RL agent chooses when to terminate deliberation, the metalevel episode length is not fixed. In practice, we set a maximum number of timesteps per episode, and force the action choice to be EXEC after that number. The policy-dependent episode length has the advantage that the algorithm performance data collected is only that which is relevant to the metareasoning task. By comparison, sequence prediction methods such as that proposed by Sung, Kaelbling, and Lozano-Pérez (2021) require training on performance profiles that are run to full convergence. This can take a long time due to diminishing returns as the algorithm converges.

Experiments

Object-Level Algorithm: Weighted BRTDP

Automatic hyperparameter tuning is generally useful to improve algorithms' performance (Falkner, Klein, and Hutter 2018), and one would expect a hyperparameter-tuning learning agent to learn the optimal hyperparameters to use for a problem distribution $p(\mathcal{M})$. However, we wish to demonstrate the *metareasoning-specific benefits* of hyperparameter tuning. To do so, we introduce a hyperparameterised version of BRTDP (McMahan, Likhachev, and Gordon 2005), which we call Weighted BRTDP (WBRTDP). By allowing the metalevel agent to change the weighting hyperparameter, we can give it more control over the object-level algorithm's behaviour. Specifically, we expect it to learn to use weights that lead to fast, imperfect solutions when the cost of planning λ is high relative to the cost of object-level policy execution.

The algorithm is inspired by variants of A* and AO* (Hansen and Zhou 2007; Bonet and Geffner 2012) which use inadmissible heuristics to bias the algorithms' convergence behaviour. WBRTDP allows switching between different lower-bound heuristics during the search process, with minimal overhead. Figure 2 shows WBRTDP running on a single instance of the racetrack problem (described below), using differing fixed heuristic values for each line. With the admissible ($h_l = 0.0$) fixed heuristic for all states, the algorithm converges to the true optimal policy some time after time t_2. However, early in the search process (e.g. at t_1), the inadmissible heuristic values enable much faster convergence to a suboptimal policy. We detail WBRTDP and the features we used for metareasoning in the supplementary material.

Deep RL Algorithm

In our experiments we use DQN (Mnih et al. 2015), a frequently-used model-free deep RL algorithm. The action space for the algorithm is the EXEC action or choosing to set the heuristic weight to a value $h_l \in \{0.0, 10.0, 20.0, 30.0\}$. Further details on the implementation of DQN for our method can be found in the supplementary material.

Experiment Domains

We generate object-level MDP problems from problem distributions in two domains.

The **deep sea treasure (DST) domain** is similar to that in Figure 1, but larger. Object-level solution improvement in this domain can result from the planner optimising its path to a given treasure, or finding a better treasure. Finding a better treasure will generally cause a larger improvement in the solution expected cost. This makes the domain interesting for metareasoning problems: performance profiles will typically not show diminishing returns behaviour.

The domain was originally proposed as a deterministic multi-objective episodic MDP by Vamplew et al. (2011). We convert it to a probabilistic single-objective SSP. The agent starts in the top left corner of a grid world with randomly sampled dimensions, with actions that probabilistically accelerate it in the eight cardinal directions. With probability P_{fail}, the acceleration action fails. Each action (including no acceleration) has a fixed time cost of 1. Final rewards for gathering treasure are converted to costs. The cost of a treasure-collection action is the maximum value of treasure generated in this problem class minus the value of the treasure collected. The default policy is to proceed directly downwards from the current state. The shape of the sea floor, treasure locations, treasure values and $P_{\text{fail}} \sim \text{Uniform}(0, 0.3)$ are randomly sampled. Deeper states correlate with higher average treasure values. P_{fail}, the x and y dimensions of the problem instance,

The Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI-24)

and max velocity $v_{\max} \in \{1, 2\}$ are provided as context to the metalevel agent.

The **race track (RT) domain** is a grid world originally described by Barto, Bradtke, and Singh (1995), and used by McMahan, Likhachev, and Gordon (2005) to evaluate their BRTDP algorithm. The state space $(x, y, v_x, v_y) \in S$ is 4 dimensional and consists of 2D position and velocity. Actions are to accelerate in any of the 8 cardinal directions, or to do nothing. Actions fail to have any effect with probability P_{fail}. Colliding with a wall sets the velocity to zero. The initial state is a state on the start line and goal states are the racetrack finish line. The default policy is to proceed at a constant speed of 1 in the direction along the track.

Problem instances for this domain are randomly generated 28×21 racetrack layouts, combined with randomly sampled maximum velocities and action failure probabilities. The maximum allowed velocity of the agent in each axis is $v_{\max} \in \{3, 4\}$ with equal probability, and $P_{\text{fail}} \sim \text{Uniform}(0, 0.3)$. These two values are provided to the metalevel agent as context. Challenges for metareasoning in this domain are the relatively large 4-dimensional state space, and the object-level planner's uninformative heuristic function. Unlike the DST domain, goal states are located some distance from the agent's start state. This can lead to planning algorithms taking a long time to generate an initial solution.

For the experiments we define time in units of object-level planner state visits. For WBRTDP, a state visit consists of evaluating transition probabilities and carrying out a backup at a state, which can be assumed to take constant time if the number of actions and transitions available is similar across all states. Using this instead of elapsed real time aids repeatability by negating external effects, such as varying processor task load impacting processing time. The maximum time for a metareasoning episode is 10K state visits for the DST domain or 100K state visits for the RT domain. This is divided evenly into 20 metalevel timesteps. For each problem instance, the thinking cost per metalevel timestep is sampled from $\text{Uniform}(0.0, 10.0)$. For both domains, illustrations and additional details are given in the supplementary material.

For each method, we evaluate across 8 DQN agents trained from differing random seeds. Agents are trained with 300K steps (RT) or 600K steps (DST) of experience, and their best performing checkpoint (evaluated on a non-overlapping evaluation set) is used in the experiments. Experiments are run using a held-out set of 1000 problems sampled from $p(\mathcal{M})$.

Results

We compare our method to two alternative ways of addressing the lack of solution quality knowledge. These methods use the known-solution-quality cost function from previous works (Sung, Kaelbling, and Lozano-Pérez 2021; Bhatia et al. 2022), but differ in how they estimate the current solution quality. The first estimation method, *MidBound*, uses the midpoint of the BRTDP bounds as an estimate of the expected cost of the solution. This is readily available and cheap to compute, but is likely inaccurate. The second method, *PolicyEval*, evaluates the current object-level policy on the object-level MDP at each metareasoning step. This gives an accurate estimate of the current solution quality, but is expensive to compute. The additional reasoning increases the cost of thinking incurred by the metalevel agent by a constant factor, which we measure empirically as described in the supplementary material.

Additionally, we compare performance with ablations of our method that disable certain components. The *NoFeatures* ablation removes algorithm features from the RL agent state, the *NoContext* ablation similarly removes the problem context, and the *NoTuning* ablation disables hyperparameter tuning so carries out only stopping time optimisation.

We do not compare with the metamyopic agent proposed by Lin et al. (2015) as their setting is online (rather than single-shot) metareasoning. Furthermore their method relies on predicting BRTDP bounds behaviour from the last step's bounds behaviour. In almost all cases in our experiments, it takes more than one metalevel timestep for the planner to generate an initial solution. This would result in the metamyopic agent assuming zero possibility of solution improvement and always executing after a single thinking step.

Figure 3 shows the combined thinking and acting cost for each method in the DST domain. Performance is normalised for each problem instance, where 0.0 is the expected cost of the optimal object-level solution (equivalent to the optimal metalevel cost assuming zero thinking cost) and 1.0 is the expected cost of the default policy. It is not possible for the metalevel agent to achieve a cost lower than the optimal object-level solution, as $\lambda \geq 0$. This is a similar concept to the metareasoning gap presented by Lin et al. (2015).

In the DST domain, our method achieves the lowest cost on average. The NoContext ablation performs the worst of the ablations, as this learning agent does not know the cost of thinking for the current problem. At best it can only optimise its behaviour in expectation for the cost of thinking distribution in the problem distribution. All ablations show a range of deliberation time, demonstrating that they are carrying out metareasoning based on the problem at hand rather than converging on a fixed metalevel episode length that suits the problem distribution. In fact all ablations have a termination time ranging from the earliest timestep to the latest timestep.

The two methods that estimate solution quality, MidBound and PolicyEval, have a much narrower range of deliberation time and incur more cost on average. For MidBound, the agent's estimate of the current solution quality is particularly inaccurate at the start of the episode, where the WBRTDP bounds are very loose. This leads to myopic behaviour and early termination. On the other hand, PolicyEval incurs an increase in thinking cost by having to evaluate its current policy at every metalevel decision step. For this problem setting and object-level implementation, the increase in thinking cost varied from around 50% to 100%. This is too high a cost to be offset by the improved solution quality estimate. Given the high cost of thinking, it converges to always terminating very early in the episode.

For the RT domain (Figure 4), the NoFeatures ablation performs almost identically in cost and metalevel episode length to the full method, although it does have higher standard deviation in cost. This suggests that for this domain the algorithm features are not particularly useful, and good performance can be achieved based on other state features.

The Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI-24)

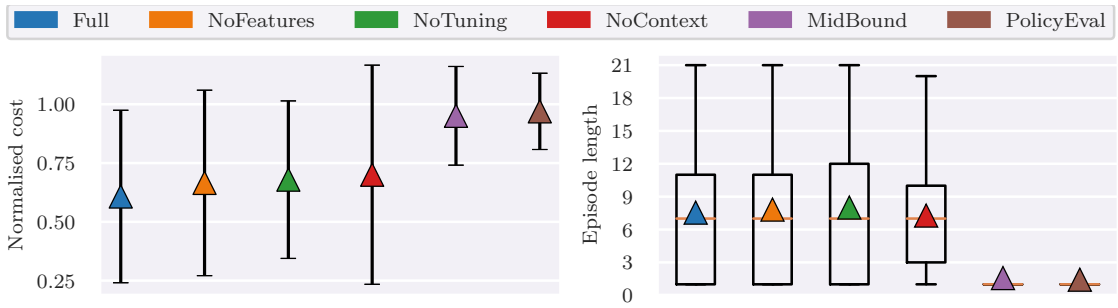


Figure 3: Normalised total thinking and acting cost and metalevel episode length on the held-out problem set in the DST domain. The left hand side plot shows the mean and standard deviation of the incurred cost for each method or ablation, and the right shows the distribution over number of thinking steps before executing. Triangles show mean values, orange lines show medians.

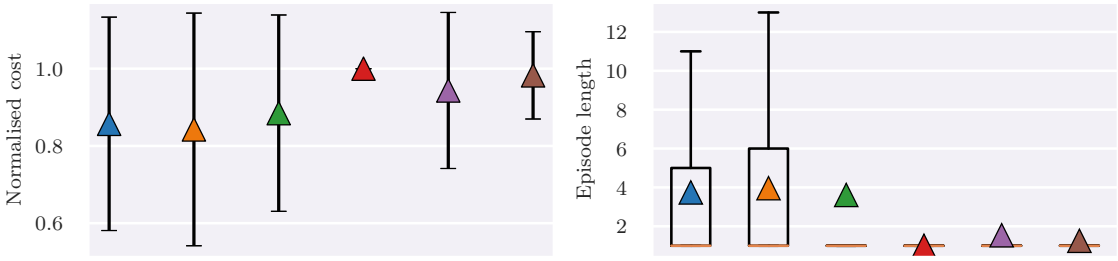


Figure 4: Normalised total thinking and acting cost and metalevel episode length on the held-out problem set in the RT domain.

The mean metalevel episode length for the NoTuning ablation significantly differs from the median and quartiles. This could result from a small proportion of long episodes and a large proportion of immediate execution. This makes sense for an ablation which can only perform the slowest, most optimal planning, and chooses to do this only when the cost of thinking is low. Finally, the MidBound estimation method is more successful in this domain than in the DST domain, but still performs poorly compared to our method.

We analyse the statistical significance of these results using the one-sided Mann-Whitney U test. Our method is significantly better ($p = 0.01$) than all ablations and alternative solution quality estimation methods in both domains, except for the NoFeatures ablation in the RT domain.

In order to demonstrate the metareasoning behaviour of our method, we study metalevel episode length in more detail in the DST domain. For the held-out problem set, Figure 5 shows the relationship between the cost of thinking for an episode $\lambda(\mathcal{M})$ and the timestep at which the metalevel agent chooses the EXEC action. There is a strong negative correlation between the two, indicating that the agent is more likely to terminate early when the cost of thinking is high. The figure also shows the agent's hyperparameter optimisation behaviour. Rather than learning a fixed optimal hyperparameter value for the problem distribution, the agent is clearly adapting its hyperparameter values to the cost of thinking in the current problem. Higher costs of thinking correlate with higher WBRTDP heuristic weights. Higher heuristic weights decrease the time taken for the planner to generate an initial imperfect solution, as illustrated in Figure 2.

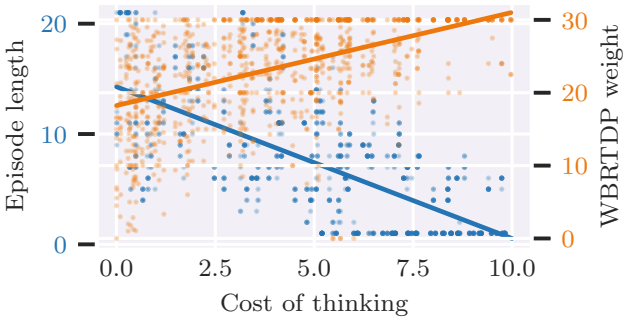


Figure 5: Correlation between the cost of thinking value in an episode and two algorithm behaviour metrics: metalevel episode length and average hyperparameter values chosen by the metalevel agent. Lines are linear regression fits.

Conclusion

This paper has presented a learning-based method to achieve non-myopic metalevel control of probabilistic planning algorithms. Our abstraction of algorithm configuration to algorithm features may lead to a non-Markov state transition function, which could be addressed in future work using a recurrent, LSTM or transformer architecture. We also aim to extend this method to the online planning rather than single-shot planning setting, widening its applicability. Finally, the assumption of a default object-level policy could be lifted by reasoning about both the goal-reaching success probability and the expected cost of the current policy.

The Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI-24)

Acknowledgements

This work received EPSRC funding via the "From Sensing to Collaboration" programme grant [EP/V000748/1]. Matthew Budd was supported by an Amazon Web Services Lighthouse Scholarship.

References

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1-2): 81–138.

Bhatia, A.; Svegliato, J.; Nashed, S. B.; and Zilberstein, S. 2022. Tuning the hyperparameters of anytime planning: A metareasoning approach with deep reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, 556–564.

Bonet, B.; and Geffner, H. 2012. Action selection for MDPs: Anytime AO* versus UCT. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, 1749–1755.

Burns, E.; Ruml, W.; and Do, M. B. 2013. Heuristic search when time matters. *Journal of Artificial Intelligence Research*, 47: 697–740.

Callaway, F.; Gul, S.; Krueger, P.; Griffiths, T.; and Lieder, F. 2018. Learning to select computations. In *34th Conference on Uncertainty in Artificial Intelligence (UAI 2018)*, 776–785.

Cashmore, M.; Coles, A.; Cserna, B.; Karpas, E.; Magazzeni, D.; and Ruml, W. 2018. Temporal planning while the clock ticks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, 39–46.

Cox, M. T.; and Raja, A. 2011. *Metareasoning: Thinking about thinking*. MIT Press.

Delalleau, O.; Peter, M.; Alonso, E.; and Logut, A. 2019. Discrete and continuous action representation for practical RL in video games. *arXiv preprint arXiv:1912.11077*.

Falkner, S.; Klein, A.; and Hutter, F. 2018. BOHB: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, 1437–1446. PMLR.

Hansen, E. A.; and Zhou, R. 2007. Anytime heuristic search. *Journal of Artificial Intelligence Research*, 28: 267–297.

Hansen, E. A.; and Zilberstein, S. 2001. Monitoring and control of anytime algorithms: A dynamic programming approach. *Artificial Intelligence*, 126(1-2): 139–157.

Hay, N.; Russell, S.; Tolpin, D.; and Shimony, S. E. 2012. Selecting computations: theory and applications. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, 346–355.

Lin, C. H.; Kolobov, A.; Kamar, E.; and Horvitz, E. 2015. Metareasoning for Planning Under Uncertainty. In *IJCAI'15 Proceedings of the 24th International Conference on Artificial Intelligence*, 1601–1609.

Mausam; and Kolobov, A. 2012. *Planning with Markov decision processes: An AI perspective*. Morgan & Claypool Publishers.

McMahan, H. B.; Likhachev, M.; and Gordon, G. J. 2005. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *Proceedings of the 22nd International Conference on Machine learning*, 569–576.

Milli, S.; Lieder, F.; and Griffiths, T. 2017. When does bounded-optimal metareasoning favor few cognitive systems? In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533.

Russell, S.; and Wefald, E. 1989. On Optimal Game-Tree Search using Rational Meta-Reasoning. In *IJCAI*, 334–340.

Russell, S.; and Wefald, E. 1991. Principles of metareasoning. *Artificial Intelligence*, 49(1-3): 361–395.

Shperberg, S. S.; Coles, A.; Karpas, E.; Shimony, S. E.; and Ruml, W. 2020. Trading plan cost for timeliness in situated temporal planning. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*.

Sung, Y.; Kaelbling, L. P.; and Lozano-Pérez, T. 2021. Learning when to quit: meta-reasoning for motion planning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 4692–4699. IEEE.

Vamplew, P.; Dazeley, R.; Berry, A.; Issabekov, R.; and Dekker, E. 2011. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine Learning*, 84: 51–80.

6.1 Limitations and Future Work

As an offline metareasoning method, this work is only able to control object-level algorithms which calculate a solution and then act on that solution with no further reasoning. The reasoning system is not "strongly" situated in the environment: the only connection is via the cost of reasoning for the agent in the problem instance. Furthermore, our experimental domains only cover cases where the object-level algorithm has a complete and accurate model of the environment. In principle, the method is applicable to offline reasoning algorithms which are resilient or adaptive to epistemic uncertainty. However, algorithms which reason about epistemic uncertainty are more complex than those that do not, and the metalevel policy would likely need to be trained on a much larger dataset of decision-making traces to learn to perform well. Additionally, more complex object-level algorithms lead to more computation being required to generate each metalevel data point, leading to even longer training times.

Decision-making traces from the method in Chapter 3 are roughly two orders of magnitude more expensive to generate than those from the object-level problem domains studied in this work. Additionally, the method in Chapter 3 is not an anytime algorithm. However, it is not difficult to imagine an anytime version of the method in Chapter 3 which starts with fewer epistemic parameter samples and then refines the solution over time. Metalevel control can be applied to this type of contract algorithm, which must be given a commitment to a computation budget before they can begin reasoning (Zilberstein, 1995). This would typically turn the metalevel control problem into a regression, rather than sequential decision-making, problem. However, metalevel control of contract algorithms removes the flexibility of monitoring an anytime algorithm's progress to better react to reasoning dynamics.

The method assumes that the environment and agent state in the environment are static during the reasoning process. In the following chapter, we extend the method to online decision-making algorithms, which includes the metareasoning component more closely in the situated decision-making process. This allows for

the agent's state in the environment to change during reasoning, and for reasoning costs to vary based on the agent's state within the environment. Non-stationary problem settings, with object-level reasoning problems represented as non-stationary MDPs (Cheung et al., 2020), are an interesting avenue for future work.

We make use of off-policy model-free RL, giving higher sample efficiency than on-policy methods (Haarnoja et al., 2018). However, the method still requires a large number of samples to learn a metalevel policy. The learned metalevel policy is specific to the object-level algorithm, training problem distribution, and the cost of planning for the agent in the environment. Future work could investigate the generalisability of the metalevel policy across problem distributions and planning costs, such as starting from a pre-trained metalevel policy and fine-tuning it on a new problem distribution. This may also identify interesting common features between different metareasoning problems. Learning from datasets of reasoning traces, in an offline RL manner (Levine et al., 2020), could allow for training metalevel policies using only logged solution traces. This would significantly reduce the computational cost of training a metalevel policy purely through interaction. Improvements to training efficiency would help scale the method to problem settings and object-level algorithms which operate under epistemic uncertainty.

The metalevel control formulation, where a separate metalevel agent controls an object-level agent, has advantages and disadvantages. Designing the metalevel agent as a separate component allows for algorithm agnosticism, and for interpretable interfaces between the two agents. It allows for easier control of metareasoning overhead, as the metalevel agent can be designed to be as lightweight as possible. Several metareasoning works extend existing object-level reasoning algorithms to include metareasoning components, effectively combining the two levels of reasoning (Burns et al., 2013; O'Ceallaigh and Ruml, 2015; Sezener and Dayan, 2020). These works are able to exert more fine-grained control over the object-level algorithm, such as selecting specific nodes to expand in a search tree based on metalevel considerations. This type of integration is difficult to achieve with hyperparameter-like control interfaces as defined in our work. The metalevel control

separation of components can lead to suboptimal behaviour in cases where the object-level agent is unaware of the metalevel agent's objectives, or the metalevel agent can potentially break assumptions of the object-level agent. This is explored in more detail in the following chapter.

There is wide scope for more extensive and flexible control of the object-level reasoning process than simply selecting hyperparameters. The metalevel agent could select or switch between different reasoning algorithms or variants on-the-fly, for example switching between upper confidence bound (Kocsis and Szepesvári, 2006) and Boltzmann (Painter et al., 2023) exploration in MCTS. Similarly, further research could identify which object-level algorithm features are most useful for metalevel control, and investigate automatic discovery of these features.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Stop! Planner Time: Metareasoning for Probabilistic Planning Using Learned Performance Profiles
Publication Status	☑ Published ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and unsubmitted work written in a manuscript style
Publication Details	**Matthew Budd**, Bruno Lacerda, and Nick Hawes. Stop! Planner Time: metareasoning for probabilistic planning using learned performance profiles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20053–20060, 2024.

Student Confirmation

Student Name:	Matthew Budd		
Contribution to the Paper	• I developed the approach proposed in the paper. • I implemented the method and tested the system in simulation. • I wrote the paper with review and input from Bruno Lacerda and Nick Hawes.		
Signature		Date	April 24, 2025

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title:	Professor Nick Hawes		
Supervisor comments	I confirm that Matthew made a substantial contribution to the publication, and that the description above is accurate.		
Signature		Date	April 25, 2025

This completed form should be included in the thesis, at the end of the relevant chapter.

7 Metareasoning for Online Decision-Making

	Chapter 3	Chapter 4	Chapter 5	Chapter 6	Chapter 7
Epistemic uncertainty	Environment dynamics	Environment dynamics	Environment dynamics	Planner dynamics	Planner dynamics
Mode	Offline	Replanning	Online	Offline	Online
Model	Timed MDP	EstMDP	BAMDP	Metalevel MDP	Metalevel MDP
Solution method	VI (max reward)	VI (reachability)	MCTS	Deep RL	Deep RL
Data regime	Small/ medium	Small	Small	Large	Large

Table 1.1: Settings, methods and features for each chapter.

In this chapter, we extend the offline metareasoning framework from the previous chapter to a fully online setting. At each timestep in the environment, the metalevel controller decides whether to execute the object-level action specified by the policy from the object-level algorithm, or execute a reasoning action to improve this policy. The cost of executing reasoning actions may vary depending on the agent's state in the environment, and may even result in a change of state according to the environment dynamics.

The online metalevel control framework is applicable to a wider range of decision-making problems, and aims to solve the practical reasoning-related challenges encountered in previous chapters. In Chapter 4, the exploration algorithm explores the environment in a cost-effective way by balancing the expected informativeness of goal states with the estimated cost and safety of reaching them. In order to optimise for exploration rate, the PKSMDP cost structure would represent

execution time for each action. However, the algorithm has no ability to reason about the time required to replan, which is significant and dependent on several hyperparameters. One such hyperparameter is the number of potential goal states to evaluate. Evaluating more possible goal states increases the likelihood of finding well-performing goal states, but each goal state evaluation takes time. An online metalevel controller could dynamically adjust the number of goal states to evaluate based on exploration progress and the scores of goal states already evaluated, in order to optimise exploration rate.

The online adaptive method in Chapter 5 by default carries out a fixed number of MCTS trials per decision. Figure 3 in that work shows a result of the agent's reasoning process being situated in the environment: in the radiation domain, spending longer reasoning can increase the amount of radiation cost incurred, as radiation cost is incurred while reasoning. This hyperparameter could be tuned to try to achieve better performance on average across deployment environments. However, it is possible to achieve improvements beyond simply optimising the number of reasoning trials across the environment distribution. An online metalevel control agent could choose to carry out in-depth planning when the agent is in a low-cost region of the environment, and carry out less or no planning computation when the agent is in a high-cost region.

Our metalevel control framework is fully applicable to a replanning-based approach, where the metalevel controller can choose to replan at any time, potentially depending on the agent's state. For clarity, we focus on a fully online approach where a single policy is maintained and improved at the object level, using an anytime planning algorithm.

Compared to online heuristic search metareasoning methods (O'Ceallaigh and Ruml, 2015; Mitchell et al., 2019), our method applies to stochastic environments. To our knowledge, a single existing practical method exists for metalevel control of online probabilistic planning (Lin et al., 2015). However, it applies only to one specific object-level algorithm, and makes a limiting meta-myopic simplification for tractability. It also lacks the hyperparameter control aspect of our work, which

allows the metalevel controller to tune behaviour of the object-level algorithm's reasoning process for the problem at hand. We show that our method can learn to outperform this existing method, the offline metareasoning approach from the previous chapter, and common fixed reasoning strategies.

Please note that this paper has not yet been submitted, or accepted for publication in its current form.

Think Fast! Learning to Control Online Reasoning in Stochastic Environments

Matthew Budd, Bruno Lacerda, Nick Hawes
Oxford Robotics Institute
University of Oxford
{mbudd, bruno, nickh}@robots.ox.ac.uk

Abstract

When an autonomous agent's decision-making has resource costs or incurs potential real-world consequences, its performance can be improved by reasoning about its own decision-making process. This is known as *metareasoning*, and is a key capability of rational agents. However, existing metareasoning methods have significant limitations. Most apply only to the *offline* setting, controlling only how long the agent should think before executing its current best solution. Few methods exist for *online* metareasoning, where the agent can interleave thinking and acting, and these make strong simplifying assumptions that limit their performance. It is rarer still for methods to be applicable to stochastic problems, or to consider the effects of the environment on the agent's planning process.

In this work we extend a learning-based metareasoning method for probabilistic planning to the online setting. The framework enables the agent to learn *when*, *where* and *how* to think in order to make better decisions in stochastic environments. We demonstrate our method outperforming several baselines across two domain distributions, each highlighting different benefits of online metareasoning.

1 Introduction

Autonomous agents often leverage online computation to surpass the performance attainable through offline pre-training alone [1, 2]. For sequential decision-making, this is implemented using the *decision-time planning* paradigm, where the agent performs planning computation before each action [3, 4]. However, optimally balancing the time spent planning and acting is a challenging problem. In practice, this parameter is usually set to a fixed value by the designer, based on their knowledge of the agent's capabilities and the types of problems the agent will encounter. In this work, we aim to enable agents to learn how best to balance planning and acting, given the current state of the agent and the environment.

We pose the problem of balancing planning and acting as a *metalevel control* problem [5]. In this problem formulation, a *metalevel* agent supervises the *object-level* algorithm carrying out the decision-making process. The metalevel algorithm may reason about when to allow the object-level algorithm to continue planning, and when to act using the current solution. It may also change *how* the object-level algorithm reasons, by selecting hyperparameters or even different algorithms that are better suited to the current problem or situation [6, 7]. *Anytime* [8] object-level algorithms are ideal in this setting, as they can be queried for their current solution at any point in time.

Metalevel control is particularly useful when real-world decision-making is subject to costs and constraints on both acting and planning, often drawing from shared resources. Planning and acting both take time and energy: this is particularly important for mobile robots, where a finite battery reserve must be shared between planning and acting, and computation constraints make planning time a meaningful consideration.

Preprint. Under review.

The costs of reasoning may also depend on the agent's state within an environment: Figure 1 illustrates an example setting inspired by Budd et al. [9] where a mobile robot incurs varying rates of radiation exposure while operating in a radioactive environment. In this setting, it can be difficult to specify an optimal planning duration that is long enough for good decision-making but short enough to avoid excessive radiation exposure while planning [9]. The cost-optimal metalevel behaviour in this setting is to pause to plan only in low radiation areas in order to take advantage of lower planning costs. In other domains, such as underwater robotics in strong current areas, pausing to plan may even cause the agent's state to change due to environmental dynamics.

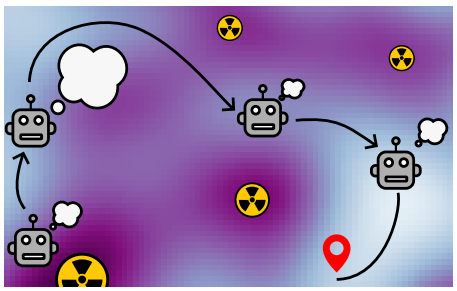


Figure 1: A mobile robot plans and executes a path across a radioactive environment. Using online metareasoning, it can pause to plan for longer (larger thought bubbles) in safer areas so that it can move continuously through hazardous zones without planning in them.

The simplest metalevel control problem is to decide the length of time to spend planning, before proceeding to fully execute the current best solution. We refer to this as *offline* metareasoning, as no planning takes place during the execution period. Offline metareasoning methods aim to solve the *how to plan* problem, by determining the optimal duration of the planning period and potentially which hyperparameters to use for the planning process [6]. In this work we study the more general case of *online* metareasoning, where the agent can interleave planning and acting. As well as broadening the applicability of metareasoning to controlling a wider class of object-level algorithms, we also wish to demonstrate several metareasoning-specific advantages of online metareasoning.

First, an online metalevel agent can decide *when* best to plan, based on its current knowledge of the environment. Decision-time planning improves performance by focusing search effort on the current state [4], and by adapting behavior to the actual environment parameters encountered at test time, thereby better addressing epistemic (i.e. model) uncertainty [9]. Similarly, online metareasoning enables adapting *reasoning behaviour* to the environment encountered: rather than planning for all possible situations, the agent can reduce decision-making costs by deferring in-depth planning until it has more information about the environment. For example, consider a robot navigating a building with internal doors it cannot open: if possible, it is more efficient to observe whether the door is open before planning a policy that includes paths passing through the door.

Second, online metareasoning enables optimisation of *where* to plan, rather than being limited to planning at the initial state. Planning may incur more cost at some states than others, as illustrated by the radiation exposure example. In some states planning may be unsafe or not be possible at all: for example, autonomous underwater vehicles often offload their planning to a remote server via satellite, so can only plan while at the water surface.

To the best of our knowledge, the method we present is the first to jointly consider the *when*-, *where*- and *how*-to-plan aspects of metalevel control within a single framework. We achieve this by extending a learning-based metareasoning method for probabilistic planning to the online setting. By learning the behaviour of object-level algorithms from data, the framework applies to any object-level algorithm and problem distribution. We demonstrate our method's performance in domains which highlight the different benefits of online metareasoning. Our method outperforms existing online metalevel controllers, both adaptive and fixed, and a state-of-the-art offline metareasoning method [7].

2 Related Work

Due to the intractability of exact metareasoning [10], no generally applicable exact algorithm exists. Metalevel control methods therefore differ by what type of object-level algorithm they supervise and the assumptions they make. The difficulty of metareasoning means that algorithms must be cognisant of the overhead of metareasoning [11, 12]. Effort spent on metareasoning should be less than the effort saved as a consequence of using metareasoning, or the net effect is negative.

Our focus is on high-level control over an agent's reasoning process during interaction with a stochastic environment. We therefore take a metalevel control approach, and model an arbitrary object-level algorithm as a parameterised black box. In contrast, some approaches integrate metareasoning

components into a specific search algorithm [13, 14], enabling decision-making about which branches of a search tree to expand or execute. The integrated approach is more suitable for problems with timing constraints, where some search branches may become impossible to execute due to deadlines.

Offline metareasoning. Callaway et al. [15] carry out approximate metareasoning by defining three estimated Value of Computation (VoC) features, which are designed to be upper or lower bounds on the value of additional reasoning. These VoC features are combined, using weights learned by Bayesian optimisation for the specific problem distribution, to estimate the true value of continuing computation. However, their method incurs significant metareasoning overhead from calculating VoC features and does not scale past the small deterministic MDPs they evaluate their method on. They assume an idealised object-level reasoning process, rather than learning the behaviour of real object-level algorithms, as we do.

One recent work carries out offline metareasoning for probabilistic planning [7], building on a reinforcement learning-based framework that Bhatia et al. [6] designed for metalevel control of deterministic search problems. One key advantage of deep RL-based metalevel control is that the cost of querying the trained metalevel policy is small compared to other methods that require online calculations to determine the benefit of further planning [11, 15]. As these methods are data-driven, they are able to learn the optimal stopping point and hyperparameter control for a given object-level problem distribution and arbitrary object-level algorithm.

Online metareasoning. To the best of our knowledge, the only practical online metareasoning algorithm for probabilistic planning is presented by Lin et al. [11]. Their method adds a metalevel control layer to a specific object-level algorithm, BRTDP [16]. It does not require training on a problem distribution, but its performance is limited by several simplifying assumptions. The most significant of these is the *meta-myopic* assumption, which means that the metalevel agent only considers the immediate benefit of a single additional planning step. Though their formulation supports reasoning about the cost of planning in the agent's current state, the meta-myopic assumption means that the agent is unable to reason about future states potentially having lower costs of planning or more information available. They also do not consider hyperparameter control of the object-level algorithm, so their metalevel agent cannot control *how* the object-level algorithm reasons.

Ho et al. [17] analyse online metareasoning by defining an information processing Bellman objective that jointly optimises both reward and planning cost. They can therefore infer how much planning effort should be spent in each state of the MDP, in a similar manner to our work. As their objective is to study human cognition rather than a practical metareasoning algorithm, their gradient-based algorithm scales very poorly with the state space size and is impractical for real-world problems.

3 Preliminaries

Object-level SSP MDP. The object-level problem we consider is a stochastic shortest path (SSP) Markov Decision Process (MDP) [18], and we also formulate the metalevel control problem as an SSP MDP. An SSP MDP is defined as a 6-tuple $\mathcal{M} = \langle S, \mathit{init}, A, T, C, G \rangle$ where S is a finite set of states; $\mathit{init} : S \to [0, 1]$ is an initial state distribution; A is a finite set of actions; $T : S \times A \times S \to [0, 1]$ is a probabilistic transition function; $C : S \times A \to \mathbb{R}_{\geq 0}$ is a cost function; and $G \subset S$ is a set of absorbing, zero-cost goal states. When referring to the structure of a specific SSP MDP instance $\mathcal{M}$, we add a subscript for clarity, e.g. $T_{\mathcal{M}}$. The two widely studied problems of infinite- and finite-horizon reward maximisation can be compiled into an equivalent SSP MDP [19].

The object-level algorithm aims to produce a policy π that minimises the expected cumulative cost of reaching the goal from the initial state. We assume that the policy is stationary and deterministic (i.e., $\pi : S \to A$), which is known to be sufficient for SSP MDPs [19]. A policy is *proper* in state s if, when starting from s, it reaches a goal state $s_g \in G$ with probability 1. For an SSP MDP there must exist a policy that is proper in all initial states. All improper policies must have infinite expected cumulative cost from states they are improper in. Under these assumptions, a minimum cost proper policy is known to exist [19].

In the same manner as in [11], the SSP MDP is assumed to include an action NOP which represents the agent planning for a fixed duration. The cost of the NOP action is the cost of planning in that state, and the probabilistic outcomes of the NOP action define possible state transitions that may occur.

4 Problem Formulation

The object-level algorithm, which is a probabilistic planner, operates on SSP MDP instances $\mathcal{M}$ drawn from a problem distribution $p(\mathcal{M})$ which is a probability distribution over possible MDPs the agent may encounter. This distribution has domain $D_{\mathcal{M}}$, and MDP instances from this domain may have different state spaces, action spaces, transition functions, and cost functions. However, they will likely share some common structure, such as the factorisation of the state space [20]. The internal state of the planner is represented by its configuration $\chi \in X$. The configuration is the full internal state of the planner, including all search trees, value function tables or approximators, and any other internal state variables. A configuration induces a policy π_χ which is the best policy found by the planner so far given the current configuration. The objective of the planner is to find a policy π_χ that minimises the expected cumulative cost of reaching a goal state from the initial state in the object-level MDP $\mathcal{M}$.

The planner is parameterised by hyperparameters $\boldsymbol{\Delta} = \{\Delta_i\}_{i=1}^{N_\Delta}$, where each hyperparameter Δ_i has a finite set of valid values $\Delta_i = \{\delta_1, \delta_2, \ldots, \delta_{k_i}\}$ or a range of valid values $\Delta_i = [\delta_{\min,i}, \delta_{\max,i}]$. The planner's configuration evolution over time is dependent on the hyperparameter values or sequence of values it has been run with. To model the configuration evolution of the planner, we assume a constant timestep τ which is the minimum time unit the metalevel controller can run the planner for. This could correspond to many steps or trials of the object-level algorithm, and is not equivalent to the timestep in the object-level MDP. The evolution of the planner's configuration is dependent on the problem instance $\mathcal{M}$ and the current configuration χ. When planning is carried out using hyperparameter values $\boldsymbol{\delta} = \{\delta_i\}_{i=1}^{N_\Delta}$, the configuration transition function is defined as $T^\chi_{\mathcal{M}}(\chi' \mid \chi, \boldsymbol{\delta})$. After an action is executed by the agent, including when the planner runs and the NOP action is therefore executed, the planner configuration is updated to the new state s' and the corresponding configuration transition function is defined as $T^\chi_{\mathcal{M}}(\chi' \mid \chi, s')$. These transition functions may be stochastic if the planner has a stochastic component, such as a random sampling component.

We are now ready to define the metalevel MDP, which represents the operation of the object-level planner on a single problem instance $\mathcal{M}$.

Definition 1 ***Metalevel MDP****: For a SSP MDP instance $\mathcal{M} = \langle S, init, A, T, C, G\rangle$, the metalevel MDP is an SSP MDP $\mathcal{M}^M = \langle S^M, init^M, A^M, T^M, C^M, G^M\rangle$ where:*

- $S^M = S \times X$*, combining the object-level MDP state space with the planner configuration space,*
- $init^M = init \times \delta^{Dirac}(\chi_0)$*, where $\delta^{Dirac}(\chi_0)$ is a Dirac delta at the initial configuration χ_0,*
- $A^M = (\{\text{PLAN}\} \times \Delta_1 \times \ldots \times \Delta_{N_\Delta}) \cup \{\text{ACT}\}$*, i.e. the agent may run the planner for one metalevel timestep τ using the specified hyperparameter values, or execute a single action (specified by the current policy) in the object-level MDP,*
- $T^M((s,\chi), a, (s',\chi')) \to [0,1]$ *is defined as:*

$$T^M((s,\chi), a, (s',\chi')) = \begin{cases} T^\chi_{\mathcal{M}}(\chi'|\chi,\boldsymbol{\delta}) \cdot T(s, \text{NOP}, s') \cdot T^\chi_{\mathcal{M}}(\chi'|\chi, s') & \text{if } a = (\text{PLAN}, \boldsymbol{\delta}) \\ T(s, \pi_\chi(s), s') \cdot T^\chi_{\mathcal{M}}(\chi'|\chi, s') & \text{if } a = \text{ACT}. \end{cases} \quad (1)$$

  The product terms indicate the transition order: for PLAN*, the planner runs for one metalevel timestep τ with hyperparameters $\boldsymbol{\delta}$, the* NOP *outcome at the object level is observed, and the planner configuration is updated given the outcome state s'. For* ACT*, the object-level algorithm's policy π_χ selects the object-level action, and the planner configuration is updated given s',*

- $C^M((s,\chi), a)$ *is defined by the object-level cost function, using the object-level action taken:*

$$C^M((s,\chi), a) = \begin{cases} C(s, \pi_\chi(s)) & \text{if } a = \text{ACT} \\ C(s, \text{NOP}) & \text{otherwise,} \end{cases} \quad (2)$$

- $G^M = G \times X$*, representing goal states in the object-level MDP with any algorithm configuration.*

Given an object-level SSP MDP $\mathcal{M}$, our objective is to derive a metalevel policy $\pi^M : S^M \to A^M$ that is optimal for the metalevel MDP $\mathcal{M}^M$. A cost-optimal solution to the metalevel MDP implies a policy that optimally switches between running the object-level planner and acting on the planner's

current policy, given the cost of planning in different states in the MDP and using the optimal hyperparameters at each timestep.

As previously discussed, it is infeasible to solve this metalevel MDP whenever the agent is provided with a new problem instance [11, 7]. Even the simplest object-level algorithms have large configuration spaces, and as the size of the metalevel MDP state space is larger than the object-level MDP state space, metareasoning overhead is prohibitive. Online metareasoning overhead could be avoided by pre-solving the metalevel MDP for every problem instance in the support of $p(\mathcal{M})$, but this is even more infeasible. To build a metalevel agent that can learn to solve metalevel MDPs for problem instances sampled from $p(\mathcal{M})$, we define a new metalevel MDP that is an abstraction of the metalevel MDP for a single problem instance, and contains features that are practical for learning.

5 Method

In this section, we present our approach to learning a metalevel policy for the metalevel MDP that is generalisable to new problem instances, in a similar manner as Bhatia et al. [6] and Budd et al. [7].

5.1 Abstracting the Metalevel MDP

Firstly, we reduce the state space of the metalevel MDP by abstracting the configuration space X to a much smaller space of algorithm features Ω. The mapping function $\Phi_X : X \to \Omega$ maps configurations to features. Φ_X and Ω are designed specifically for the object-level algorithm, and are chosen to be informative summary statistics of the configuration space. As we will be using deep function approximation to learn value functions, features will be automatically weighted and there is no practical drawback to defining a large number of features. Features are often readily identifiable from an algorithm's description, such as estimated value functions, search tree size/depth, the number of iterations or samples simulated, or the size of the latest update in an optimisation algorithm.

Secondly, we abstract the metalevel MDP so that it represents the operation of the planner on problems from the problem distribution $p(\mathcal{M})$, not a single problem instance. This consists of augmenting the state with a context vector $\psi \in \Psi$, where $\Phi_D : D_{\mathcal{M}} \to \Psi$. The context vector $\psi = \Phi_D(\mathcal{M})$ is a compact representation of the current problem instance $\mathcal{M}$, and allows for generalisation between problem instances with similar attributes. Φ_D and Ψ are designed to provide informative summary statistics describing the MDP problem instance, and are problem-dependent. Producing these features should not require extensive analysis of $\mathcal{M}$, in order to avoid metareasoning overhead. As the object-level MDP state representation is also a component of the metalevel MDP state, using a consistent state factor representation across MDPs in the problem distribution will likely also improve generalisation. Note that, as well as encoding a metalevel control problem, the abstract metalevel MDP encodes a meta-RL problem [21] as we represent operating under a distribution of MDPs as a single MDP. Also, as with all MDP state abstraction methods, the transition function may become non-Markovian due to the chosen abstraction of the configuration space [22].

Definition 2 ***Abstract Metalevel MDP****: For a problem distribution $p(\mathcal{M})$ over problem instances $\mathcal{M}$, the abstract metalevel MDP is an SSP MDP $\hat{\mathcal{M}}^M = \langle \hat{S}^M, \hat{init}^M, \hat{A}^M, \hat{T}^M, \hat{C}^M, \hat{G}^M \rangle$ where:*

- $\hat{S}^M = S^M \times \Omega \times \Psi$*, i.e. states are* (s, ω, ψ)*,*
- $\hat{init}^M\left(s^M = (s, \omega, \psi)\right) = \int_{\{\mathcal{M} \mid \Phi_D(\mathcal{M}) = \psi\}} init_{\mathcal{M}}(s)\; p(\mathcal{M})\; d\mathcal{M} \times \delta^{Dirac}(\Phi_X(\chi_0))$*,*
  where $init_{\mathcal{M}}$ is the initial state distribution of problem instance $\mathcal{M}$, and $\delta^{Dirac}(\Phi_X(\chi_0))$ is a Dirac delta at the algorithm initial features $\Phi_X(\chi_0)$,
- $\hat{A}^M = A^M$*, i.e. the action space is unchanged,*
- $\hat{T}^M : \hat{S}^M \times \hat{A}^M \times \hat{S}^M \to [0, 1]$ *is composed of three components T_ψ, T^ω, and π_ω, which are abstracted versions of those in Equation 1:*

$$\hat{T}^M((s, \omega, \psi), a, (s', \omega', \psi)) = \begin{cases} T^\omega(\omega' \mid \omega, \boldsymbol{\delta}, \psi) \cdot T_\psi(s, \text{NOP}, s') \cdot T^\omega(\omega' \mid \omega, s', \psi) & \text{if } a = (\text{PLAN}, \boldsymbol{\delta}) \\ T_\psi(s, \pi_\omega(s), s') \cdot T^\omega(\omega' \mid \omega, s', \psi) & \text{if } a = \text{ACT} \end{cases}$$

Algorithm 1 General deep RL on the abstract metalevel MDP

Input: Problem distribution $p(\mathcal{M})$, object-level algorithm, number of training episodes M_e, object-level planning time per metalevel timestep τ
Output: Trained policy π_θ^M.

1: Initialise π_θ^M randomly
2: **for** episode = $1, \ldots, M_e$ **do**
3: Sample $\mathcal{M}$ from D according to $p(\mathcal{M})$
4: $\psi \leftarrow \Phi_D(\mathcal{M})$ {calculate MDP context}
5: $\chi \leftarrow \chi_0$; $\omega \leftarrow \Phi_X(\chi)$ {instantiate planner, calculate initial features}
6: $s \sim \mathit{init}_\mathcal{M}$ {sample initial object-level state}
7: $\hat{s}^M \leftarrow (s, \omega, \psi)$ {construct metalevel state}
8: **repeat** {run episode}
9: $\hat{a}^M \leftarrow$ select action using $\pi_\theta^M(\hat{s}^M)$ {or e.g. ϵ-greedy exploration}
10: **if** $\hat{a}^M = (\text{PLAN}, \boldsymbol{\delta})$ **then**
11: $\chi \sim T_\mathcal{M}^\chi(\cdot \mid \chi, \boldsymbol{\delta})$ {run planner for time τ with hyperparams $\boldsymbol{\delta}$}
12: $a \leftarrow$ NOP
13: **else** $\{\hat{a}^M = \text{ACT}\}$
14: $a \leftarrow \pi_\chi(s)$ {select action using current object-level policy}
15: **end if**
16: $\hat{c}^M \leftarrow C_\mathcal{M}(s, a)$ {incur object-level cost from planning or acting}
17: $s \sim T_\mathcal{M}(s, a, \cdot)$ {object-level state outcome}
18: $\chi \sim T_\mathcal{M}^\chi(\cdot \mid \chi, s)$ {update planner with new current object-level state}
19: $\omega \leftarrow \Phi_X(\chi)$ {update features, including Q-value of NOP action $Q_\chi(s, \text{NOP})$}
20: $\hat{s}^{M\prime} \leftarrow (s, \omega, \psi)$
21: Store $(\hat{s}^M, \hat{a}^M, \hat{c}^M, \hat{s}^{M\prime})$ {e.g. replay buffer}
22: $\hat{s}^M \leftarrow \hat{s}^{M\prime}$
23: **until** $s \in G_\mathcal{M}$
24: Update parameters of π_θ^M using stored episode data
25: **end for**

where T_ψ represents object-level state transitions and T^ω represents algorithm feature transitions caused by planning or updating the object-level algorithm with the new state, as in Definition 1. By abstracting the current object-level MDP instance $\mathcal{M}$ to a context vector ψ, we no longer have a specific transition function $T_\mathcal{M}$. Given the set of MDPs where $\Phi_D(\mathcal{M}) = \psi$, we can construct T_ψ by marginalisation: $T_\psi = \int_{\{\mathcal{M} \mid \Phi_D(\mathcal{M}) = \psi\}} T_\mathcal{M}\, p(\mathcal{M})\, d\mathcal{M}$. Similarly, T^ω is constructed by marginalisation over both the object-level algorithm configuration space and the problem distribution. Finally, π_ω is the policy induced by the features ω, so is an abstraction of π_χ calculated by marginalising over the configuration space X. Note that the context vector ψ is fixed within an episode, as a single object-level problem instance is being solved,

- *$\hat{C}^M$ and $\hat{G}^M$ are defined as in Definition 1, but with π_ω replacing π_χ and Ω replacing X.*

5.2 Learning to Act in the Abstract Metalevel MDP

Algorithm 1 outlines our method for training a deep RL-based policy for the abstract metalevel MDP. We present the approach with a DQN-like structure [23] for clarity, but the metalevel MDP construction and interaction is independent of the learning algorithm used, and any deep RL algorithm could be used to learn a metalevel policy. The RL policy is trained in-the-loop by sampling one object-level problem instance from $p(\mathcal{M})$ at the start of each episode. Additional implementation details are given in the supplementary material, but we highlight two key concepts here.

Budd et al. [7] provide their RL-based offline metalevel controller with an observation of the offline planning cost per timestep, so that it can learn how to act under different planning costs. Along similar lines, we provide the agent with the Q-value $Q_\chi(s, \text{NOP})$. This Q-value is from the object-level algorithm's value function, so captures both immediate planning cost and the estimated value of possible successor states after taking the NOP action. As it is dependent on the object-level algorithm configuration χ, it is incorporated into the features ω in Line 19. A similar Q-value is also used by the myopic metalevel controller in Lin et al. [11].

6 Experiments

6.1 Baseline Algorithms

We compare to three baseline metalevel control methods. The most basic is a fixed decision-time planner (*DTP*), parameterised by the ratio of planning to acting time. The optimal value of this parameter is problem-dependent, and so we evaluate a range of values for each problem instance to find the best-performing ratio. The second baseline, *Bounds*, is the correlated metareasoner proposed by Lin et al. [11]. It uses a meta-myopic approximation based on how the object-level algorithm's value bounds changed over the previous computation step. The third baseline is the offline learned metareasoner from Budd et al. [7] (*OffLearn*), which learns from reasoning data and uses hyperparameter control to optimise the object-level algorithm's performance as our method does, but only supports planning at the initial state. This method relies on access to a default policy to complete its policy outputs, effectively granting it privileged information not available to our approach. We call our method the *OnLearn* metareasoner.

6.2 Object-level Algorithm: Weighted Bounded RTDP (WBRTDP).

Bounds is built on the Bounded Real-Time Dynamic Programming (BRTDP) algorithm [16]. For fair comparison between metareasoning agents, we therefore use a BRTDP-based object-level algorithm for both learning methods. As the learning-based methods support hyperparameter control, we specifically use Weighted BRTDP (WBRTDP) [7] as the object-level algorithm. The weight parameter allows the learning-based algorithms to trade-off the quality of the solution and the speed of computing that solution. Each metalevel MDP timestep corresponds to a fixed number of state Bellman backup operations in the object-level algorithm. This is a more consistent time unit than walltime, which is dependent on hardware capability and load, or BRTDP trials, which vary in length. Additional details, including discussion of the algorithm features abstraction function, are given in Appendix A.2.1.

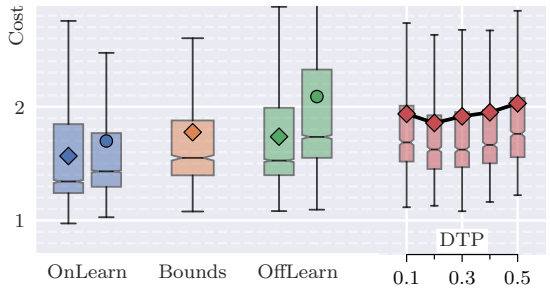
6.3 Deep RL Algorithm

We use DQN [23] as the deep RL algorithm for the metalevel controller. We are able to use DQN as the metalevel controller because the hyperparameter values are discrete, leading to a purely discrete action space. A different deep RL algorithm would be required if the hyperparameters were continuous, which would lead to a mixed discrete-continuous action space [24]. Other learning algorithms are evaluated in Appendix B.2. For RL-based metalevel controllers, we evaluate across 5 DQN agents trained with different random seeds. Details on training and compute resources are given in Appendix A.3.

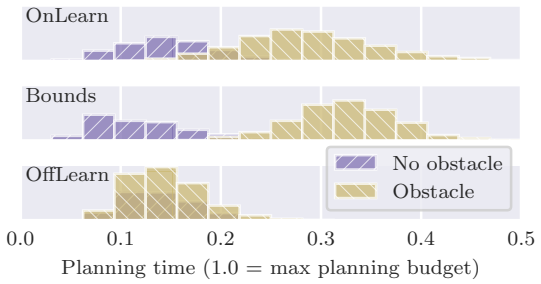
6.4 Experiment Domains

The **ObstacleRacetrack** domain is based on the racetrack domain used by Budd et al. [7] and originally introduced by Barto et al. [25]. It is a 2D gridworld where the agent state is the 4-tuple x, y, v_x, v_y, and actions are to accelerate in an 8-connected direction. Actions have a failure probability that varies by problem instance. The agent's goal is to reach one of two finish lines. However, one finish line is potentially blocked by an obstacle: whether it is blocked or not is unknown to the agent *a priori*. This is represented in the object-level MDP by a state factor with values $\{-1, 0, 1\}$, where -1 indicates the obstacle state is unknown. The agent observes the state of the obstacle by colliding with it if it is present, or passing through that area if it is not. The finish line behind the possible obstacle is closer, so incurs less cost to reach and requires less intensive planning to determine how to reach. This domain is designed to demonstrate the benefits of online metareasoning under epistemic uncertainty: the optimal metalevel behaviour is to generate a simple policy to reach the obstacle, and then to plan a detailed policy to reach the further away goal *only if the obstacle is present*.

The **RadWorld** domain is based on the example in Figure 1. The agent exists in a gridworld with randomly generated obstacles and radiation level distributions. It aims to reach a goal on the opposite side of the map while minimising the total radiation exposure from planning and acting. Actions are to move one cell in an 8-connected manner, with costs geometrically adjusted for diagonal movement. Actions have a radiation-level dependent failure probability, ranging linearly from 0 to 0.75 for the



(a) Normalised cost-to-goal for the method and baselines. Markers (diamond marker for full method, circle for disabled hyperparam. control) show the mean cost.



(b) Distribution of planning time for the method and baselines, separated by the ground-truth obstacle state.

Figure 2: Cost performance and planning time distributions in the ObstacleRacetrack domain.

lowest and highest radiation levels respectively. This domain is designed to demonstrate the benefits of online metareasoning where different states have substantially different costs of planning.

Problem distributions are defined implicitly by procedural generation of object-level MDPs for the agent to act in. Learning methods are trained on 10K object-level MDPs drawn from $p(\mathcal{M})$. All algorithms are evaluated on 1K object-level MDPs also drawn from $p(\mathcal{M})$, with consistent random seeding based on the evaluation problem instance ID. This random seeding ensures that the e.g. the ground truth obstacle state in each ObstacleRacetrack problem instance is the same for all algorithms at evaluation time. Domain visualisations, problem context functions and further details are given in Appendix A.1.

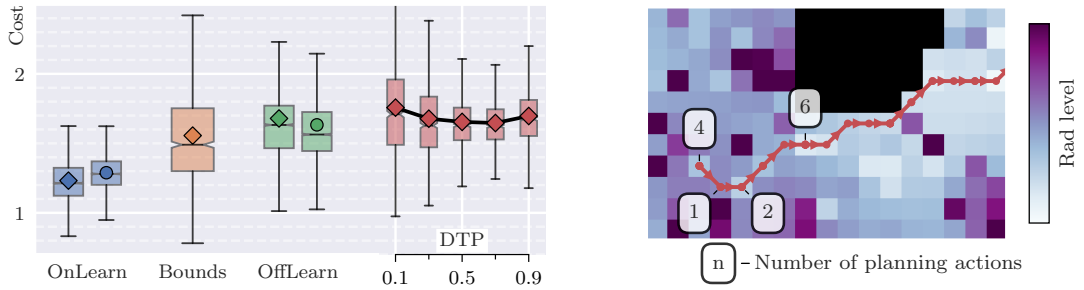
6.5 Quantitative and Qualitative Results

Cost results are normalised by the expected cost of the optimal policy for that problem instance. It is not possible to achieve normalised cost of 1 in expectation when cost is paid for reasoning, so a normalised cost of 1 is as a lower bound on performance for all metareasoning methods. As domains are stochastic, it is still sometimes possible for few runs to achieve normalised cost < 1. When comparing algorithms in the text, we report how much excess cost they incur over the lower bound of 1, rather than the difference between the normalised costs of the algorithms. Statistical significance is calculated using the one-sided Mann-Whitney U test, with significance threshold $p = 0.01$. For the learning-based metalevel controllers, we also train and evaluate in a case where hyperparameter control is disabled, giving the learned controllers access to the same interface as *Bounds*.

In the ObstacleRacetrack domain, *OnLearn* achieves significantly lower normalized cost than both *Bounds* and *OffLearn*, as shown in Figure 2a. Relative to *OnLearn*, *Bounds* incurs a 37% mean excess-cost increase (61% median increase), while *OffLearn* incurs a 30% mean increase (54% median increase). All parameterisations of *DTP* perform worse than the other methods: the best-performing parameterisation (0.2) incurs a 51% mean excess cost (82% median) relative to *OnLearn*. The ablated version of *OnLearn* performs 18% worse than *OnLearn*, but is still significantly better than *Bounds*. The ablated version of *OffLearn* performs significantly worse than all online methods.

Figure 2b shows the distribution of planning time used by three of the metalevel controllers, with the distributions separated by whether the obstacle ground-truth state was present or not present in that evaluation MDP. The two online metalevel controllers (*OnLearn* and *Bounds*) both use significantly more planning time when the obstacle is present, as this is the state where the agent needs to plan a more complex policy. Analysis of planning states in MDPs where the obstacle was present shows planning taking place after the obstacle is reached. Despite varying its planning time, and displaying a larger variation in planning times than *OnLearn*, there is no significant cost performance difference between *OffLearn* and *Bounds*. *OffLearn* has the benefit of hyperparameter control and being provided with a default policy, although the default policy does not check the obstacle state.

A wider cost performance gap is observed in the RadWorld domain, as shown in Figure 3a. For all methods the mean and median performance is similar, indicating that the distribution of costs is more consistent across runs than in ObstacleRacetrack. *OnLearn* achieves significantly lower normalized cost than all baselines. Compared to *OnLearn*, *Bounds* incurs 2.4× as much excess cost and *OffLearn*



(a) Normalised cost-to-goal for the method and baselines. Markers show the mean cost. Note that the value range of DTP planning ratio differs from Figure 2a.

(b) One evaluation trajectory for the online learning method, showing the agent's path (red) and how much time it chooses to spend planning at each state.

Figure 3: Results for the RadWorld domain.

incurs 2.9× as much excess cost. *Bounds* performs significantly better than *OffLearn*, in this domain, and *OffLearn* does not significantly outperform the best *DTP* parameterisation. *Bounds* is able to make myopic metalevel decisions about the cost of planning in its current state, but is not able to reason about the cost of planning being lower in potential future states. *OffLearn* is able to control how long to reason based on the radiation level of the start state, but has no ability to reason at other states. The ablated version of *OnLearn* performs very similarly to *OnLearn*, showing that most of the performance gain is due to non-myopic metalevel reasoning rather than hyperparameter control. Interestingly, in this domain, the ablated version of *OffLearn* performs slightly better than *OffLearn*.

Figure 3b shows the qualitative reasoning behaviour of *OnLearn* in a RadWorld MDP, showing only the states near the start state. It spends 4 metalevel timesteps planning at the start state, but waits until it reaches a low radiation level state to plan for longer. After this, it only performs one more planning step during the 80-length trajectory to the goal.

We carried out an ablation of *OnLearn* by replacing the NOP action Q-value estimate observation (Section 5.2) with the immediate cost of the NOP action in the current state, in a similar manner as Budd et al. [7]. We found that this led to a statistically significant 10% increase in mean excess cost in the ObstacleRacetrack domain, and no significant difference in the RadWorld domain. The lack of effect in RadWorld is likely because the agent remains in the same state while planning, whereas in ObstacleRacetrack the agent may transition state depending on its current speed. As the RL agent has an observation of the object-level algorithm's value estimate of the current state, it is straightforward to reconstruct $Q_\chi(s, \text{NOP})$ from this and the immediate cost observation.

7 Conclusion

We have presented a learning-based metalevel control framework for online metareasoning in probabilistic planning, and shown it to outperform several baselines in two novel environments that are well-suited to evaluating metareasoning methods.

Limitations. As a deep RL-based method, the trade-off for minimal metareasoning overhead is the requirement to train on a representative problem distribution, which we assume is available. The metalevel control formalisation itself, with separate object-level and metalevel agents, also introduces one notable limitation in the online setting: one can construct problems where optimal planning and acting decisions are not aligned, leading to suboptimal metalevel performance. For example, if there exists a state with a low cost of planning which does not contribute to solving the object-level problem, the metalevel agent has no way to instruct the planner to visit that state.

This limitation could be addressed by tighter integration of the metalevel controller and planner, with the drawback of losing the generality of the metalevel controller. Finally, although we consider a simple case of epistemic uncertainty in the environment, we assume the agent has a perfect model of the environment.

References

[1] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of Go without human knowledge. *nature*, 550(7676):354–359, 2017.

[2] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:24824–24837, 2022.

[3] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning and acting*. Cambridge University Press, 2016.

[4] Dimitri Bertsekas. *Lessons from AlphaZero for optimal, model predictive, and adaptive control*. Athena Scientific, 2022.

[5] Nicholas Hay, Stuart Russell, David Tolpin, and Solomon Eyal Shimony. Selecting computations: theory and applications. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 346–355, 2012.

[6] Abhinav Bhatia, Justin Svegliato, Samer B Nashed, and Shlomo Zilberstein. Tuning the hyperparameters of anytime planning: A metareasoning approach with deep reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 32, pages 556–564, 2022.

[7] Matthew Budd, Bruno Lacerda, and Nick Hawes. Stop! Planner Time: Metareasoning for probabilistic planning using learned performance profiles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20053–20060, 2024.

[8] Shlomo Zilberstein and Stuart Russell. Approximate reasoning using anytime algorithms. In *Imprecise and approximate computation*, pages 43–62. Springer, 1995.

[9] Matthew Budd, Paul Duckworth, Nick Hawes, and Bruno Lacerda. Bayesian reinforcement learning for single-episode missions in partially unknown environments. In *Conference on Robot Learning (CoRL)*, pages 1189–1198. PMLR, 2023.

[10] Stuart Russell and Eric Wefald. Principles of metareasoning. *Artificial Intelligence*, 49(1-3): 361–395, 1991.

[11] Christopher H Lin, Andrey Kolobov, Ece Kamar, and Eric Horvitz. Metareasoning for planning under uncertainty. In *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI)*, pages 1601–1609, 2015.

[12] Smitha Milli, Falk Lieder, and Thomas Griffiths. When does bounded-optimal metareasoning favor few cognitive systems? In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[13] Bence Cserna, Wheeler Ruml, and Jeremy Frank. Planning time to think: Metareasoning for on-line planning with durative actions. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 27, pages 56–60, 2017.

[14] Eren Sezener and Peter Dayan. Static and dynamic values of computation in MCTS. In *Proceedings of the Thirty-Sixth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 31–40, 2020.

[15] F Callaway, S Gul, P Krueger, TL Griffiths, and F Lieder. Learning to select computations. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 776–785, 2018.

[16] H Brendan McMahan, Maxim Likhachev, and Geoffrey J Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)*, pages 569–576, 2005.

[17] Mark K Ho, David Abel, Jonathan D Cohen, Michael L Littman, and Thomas L Griffiths. The efficiency of human cognition reflects planned information processing. In *Proceedings of the 34th AAAI conference on artificial intelligence*, 2020.

[18] Martin L Puterman. Markov decision processes: Discrete stochastic dynamic programming, 1994.

[19] Mausam and Andrey Kolobov. *Planning with Markov decision processes: An AI perspective*. Morgan & Claypool Publishers, 2012.

[20] Alexander L Strehl, Carlos Diuk, and Michael L Littman. Efficient structure learning in factored-state mdps. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 7, pages 645–650, 2007.

[21] Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. A survey of meta-reinforcement learning. *arXiv preprint arXiv:2301.08028*, 2023.

[22] Aijun Bai, Siddharth Srivastava, and Stuart Russell. Markovian state and action abstractions for MDPs via hierarchical MCTS. In *Proceedings of the 25th International Conference on Artificial Intelligence (IJCAI)*, pages 3029–3039, 2016.

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[24] Olivier Delalleau, Maxim Peter, Eloi Alonso, and Adrien Logut. Discrete and continuous action representation for practical RL in video games. *arXiv preprint arXiv:1912.11077*, 2019.

[25] Andrew G Barto, Steven J Bradtke, and Satinder P Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1-2):81–138, 1995.

[26] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulao, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

[27] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *The Journal of Machine Learning Research*, 22(1):12348–12355, 2021.

[28] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[29] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PmLR, 2016.

7.1 Limitations and Future Work

As this framework is an extension of the offline metareasoning framework in the previous chapter, it shares some common limitations. Again, we assume a known MDP model for object-level decision-making. In the online setting, this also includes specifying the physical outcomes of reasoning actions in the environment. We do however investigate a simple didactic form of epistemic uncertainty in the ObstacleRacetrack domain, by representing the agent's belief about the state of an obstacle with a simple deterministic belief model where the epistemic uncertainty is immediately resolved upon reaching the obstacle. Though simple, this type of epistemic state feature has been used in real applications (Lacerda et al., 2019). Because belief in the obstacle state is represented as a ternary variable with simple dynamics, the problem can be represented as a standard MDP and standard MDP methods can be used to solve it, rather than needing to maintain continuous belief distributions as in Chapter 5. However, this simple example works to demonstrate the potential benefits of online reasoning in epistemically uncertain environments. The agent learns to carry out in-depth planning only after checking the obstacle state, meaning it avoids expending planning effort on every possible eventuality.

There is scope for substantial future work on metareasoning with uncertain models. Recent work has formalised the meta-BAMDP reasoning problem and looked at a simple multi-armed bandit domain (Godara et al., 2024), but to our knowledge there are currently no practical methods for metalevel control of multi-step decision-making in epistemically uncertain models.

In the current metalevel MDP formulation, the metalevel controller chooses between planning or executing an object-level action at each timestep. However, many practical systems can do both at the same time: executing real-world actions while reasoning about the next action to take. Concurrent planning and acting has been investigated in the heuristic search (Gu et al., 2022) and situated temporal planning (Elboher et al., 2023) settings. To extend our method to concurrent planning and acting, the metalevel MDP action space could be reformulated.

Defining the metalevel action space as the Cartesian product of reasoning actions and $\{\text{ACT}, \text{NOP}\}$ would allow the metalevel controller to choose between reasoning while acting or reasoning without committing to an action.

One limitation of the online metareasoning case compared to the offline case is that both the object-level and reasoning processes must standardise on the timestep of the object-level problem. In the offline case, we can define an arbitrarily fine-grained timestep for the metalevel MDP, as the object-level MDP is entirely separate. In the online case, the decision-making costs and outcomes are defined in the object-level MDP, giving less flexibility in the design of the metalevel MDP.

Similarly to the previous chapter, the metalevel control separation of object-level algorithm and metalevel controller has some drawbacks as well as advantages. In the online setting, a new issue is that the objectives of the metalevel controller and object-level algorithm may not be fully aligned. For example, if there exists an easily reachable state that has excellent reasoning dynamics (low cost to reason, safe to remain in), then an optimal metareasoning policy would choose to go to that state and reason there. However, if the state does not contribute to solving the object-level problem, then the object-level algorithm will not choose to visit that state. In most cases there is some shared objective between the metalevel and object-level algorithms due to their coupled costs, so this is not a practical issue. For example, in the radiation domain, both the metalevel controller and object-level algorithm benefit from the agent staying in low radiation areas.

As noted in the previous paragraph, real environment actions are always chosen by the object-level policy, other than NOP actions which may be the result of the metalevel controller's decision. Despite this, we noted that in some problem settings the metalevel controller could learn to use the NOP action to achieve object-level behaviour not intended by the object-level algorithm. In one test case, the metalevel controller learned to use the NOP action to block the object-level algorithm from executing actions that are part of the object-level algorithm's policy, but that the metalevel controller has learned are not useful in the current state for the problem distribution. Although this does result in better performance, it subverts

the intended separation between the metalevel and object-level reasoning processes. Investigating this type of impact of the metalevel control framework on online decision-making is an interesting avenue for future research.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Think Fast! Learning to Control Online Reasoning in Stochastic Environments
Publication Status	□ Published □ Accepted for Publication ☑ Submitted for Publication □ Unpublished and unsubmitted work written in a manuscript style
Publication Details	**Matthew Budd**, Bruno Lacerda, Nick Hawes. Think Fast! Learning to Control Online Reasoning in Stochastic Environments. *Under Review.*

Student Confirmation

Student Name:	Matthew Budd		
Contribution to the Paper	• I developed the approach proposed in the paper. • I implemented the method and tested the system in simulation. • I wrote the paper with review and input from Bruno Lacerda and Nick Hawes.		
Signature		Date	April 24, 2025

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title:	Professor Nick Hawes		
Supervisor comments	I confirm that Matthew made a substantial contribution to the publication, and that the description above is accurate.		
Signature		Date	April 25, 2025

This completed form should be included in the thesis, at the end of the relevant chapter.

8 Conclusion

In this thesis, we began by identifying different types of reasoning problems and approaches that arise for situated agents operating in epistemically uncertain environments. We identified two key axes of variation: (i) Offline vs. online reasoning, chosen based on the agent's reasoning capabilities and whether online reasoning offers advantages in the deployment environment; and (ii) Object-level reasoning vs. metalevel control of reasoning, distinguishing between directly solving tasks and deciding how to allocate reasoning resources. These axes were used to define four research questions, **RQ1** to **RQ4**.

Chapters 3 to 5 presented algorithms for decision-making in epistemically uncertain environments, varying in their onboard computational demands and adaptability to the environment. These methods addressed **RQ1** and **RQ3**. Two of these algorithms (Chapters 3 and 4) were integrated and tested on real-robot platforms, demonstrating their practical applicability.

In Chapters 6 and 7, we developed metalevel control methods that enable situated agents to learn how best to employ object-level reasoning algorithms, given their computational constraints and problem setting. These methods addressed **RQ2** and **RQ4**. However, a significant gap remains between the object-level algorithms used in these chapters and the object-level reasoning algorithms introduced in Chapters 3 to 5. This issue is present because the computational cost of collecting metalevel training data scales with the complexity of the object-level reasoning algorithm. The ability of the metalevel controllers to learn how to use complex, computationally expensive object-level reasoning is therefore limited by the high data requirements of the model-free RL methods they are based on.

Taking a broader perspective on metareasoning algorithms, we observe a growing trend toward utilizing more online computation at test time, rather than relying solely on pre-trained or pre-solved models. For instance, some recent offline RL methods incorporate online planning despite being trained offline (Zhan et al., 2021; Sikchi et al., 2022). The introduction of chain-of-thought reasoning to improve large language model performance at inference time (Wei et al., 2022) is another example of this trend. As machine learning systems continue to scale in model complexity and resource consumption (Lin et al., 2024; Baraniuk, 2024), autonomous systems that can dynamically select the most appropriate algorithm or model for a given task will become increasingly valuable. As discussed in Section 8.1, we believe there is substantial potential for improving metalevel control methods, especially in the context of mobile robotics.

Although two of our decision-making algorithms were tested on real-robot platforms, many robots deployed in real-world applications still rely on hand-crafted mission scripts (Pebody, 2007; Phillips et al., 2023). In the future, we hope to see broader integration of autonomous decision-making algorithms into real-world robotic systems. This shift toward more autonomous reasoning will likely be accelerated by the growing deployment of autonomous systems and their increasing complexity.

8.1 Future Work

In this final section, we discuss potential future research directions that could extend the work presented in this thesis.

Robust offline planning using generative simulators In Chapter 3, we make use of parameterised stochastic simulators of different aspects of the environment. Our method attempts to address epistemic uncertainty without using a robust MDP formulation, and achieves good performance according to the real and simulated experiments. However, a robust MDP formulation may be able to achieve better performance in some settings, for example in cases where water

currents are stronger compared to the vehicle's propulsion. Robust MDP methods are more computationally expensive than the method we proposed, which may preclude solving for the same large, complete policies that our method produces. However, a robust MDP method with a smaller policy representation may be able to produce policies that are more robust to worst-case scenarios during deployment.

We did not consider the problem of updating the stochastic simulators based on new data from the environment. Further work could also investigate the use of Bayesian methods to update these models based on new data, potentially including modelling of environment dynamics changing over time.

Uncertain environment models In Chapter 4 and 5, we focused on Gaussian processes as a representative family of environment models. These models are well-suited to the continuous, natural environments we considered, but are not the only option. Deep learning-based models have been studied in a variety of settings and have shown impressive performance in some domains. However, we would be limited to data-efficient dynamics models that have some notion of epistemic uncertainty.

One approach with significant research interest is recurrent state space models (RSSMs), often referred to as "world models" (Hafner et al., 2021). Although RSSM formulations can include stochastic components, they are not usually explicitly designed to model epistemic uncertainty. Recent exceptions to this are LAMBDA (As et al., 2022) and MAMBA (Rimon et al., 2024), which develop Bayesian RSSM and meta-RL methods respectively. The world model literature usually focuses on pixel-based observations, rather than the structured sensor-based observations common in robotics. Future work could investigate RSSM designs better suited to explicitly modelling epistemic uncertainty in large-scale environments, using appropriate observation models.

Offline model-based reinforcement learning methods commonly employ deep ensembles of stochastic dynamics models to enhance adaptivity under some notion of epistemic uncertainty (Yu et al., 2020). However, such approaches do not provide an explicit separation between the aspects of the environment that are

well-known and those that remain uncertain. Their focus is on epistemic uncertainty due to a small dataset or a poorly-performing behaviour policy, rather than the fundamental uncertainty in the environment dynamics that we address in this thesis. Future work could investigate how to incorporate priors over robot and environment dynamics into deep ensemble methods, in a similar manner as the GP-based methods we used in Chapters 4 and 5.

Evaluation problems and domains In Chapter 5, we formulated the problem as a SSP MDP in order to enable goal-oriented behaviour rather than short-horizon reward maximisation. However, we evaluated our approach on problems with a simple goal – namely, reaching a target location – instead of considering tasks with more complex goals. Some standardised evaluation domains, such as those for offline RL (Fu et al., 2020), incorporate more complex goals that require longer-horizon reasoning than the continuous control tasks commonly used to evaluate RL methods (Brockman et al., 2016). However, these domains are often deterministic or exhibit limited stochasticity (for example, small Gaussian control or sensing noise), and do not emphasise epistemic uncertainty.

Finally, no widely-used benchmark domains exist for simulating the type of highly epistemically uncertain domains we study in this thesis. For example, there is no widely-used simulator and benchmark for underwater autonomous vehicles under the influence of ocean currents. Development of such domains would enable better standardisation and comparison between methods. Along similar lines, no widely-used benchmark domains exist for evaluating metalevel control methods. Compared to existing works on metalevel control, which often evaluate their methods on very small object-level problems, we defined new domain distributions in Chapters 6 and 7 that are more representative of real-world problems. However, there is still scope for defining larger domains with more complex dynamics, potentially including epistemic uncertainty in the object-level problems.

Multi-Agent Metareasoning and Communication In our studies of metareasoning in Chapters 6 and 7, we focused on a single-agent metalevel control problem. As prior work has primarily addressed this single-agent case, metalevel control in multi-agent systems remains underexplored (Raja and Lesser, 2007; Shynkar et al., 2022). In particular, we are interested in the process of offloading decision-making from the agent to remote computers or other agents. Mobile robots often have communication links as well as on-board computers, and could choose to offload computation to a remote computer to decrease usage of their own battery life, at the cost of added communication latency. Some existing methods use metalevel MDPs to reason about the costs and benefits of offloading computation (Chinchali et al., 2021; Penmetcha and Min, 2021; Navardi et al., 2024) for immediate tasks such as object detection and localisation. In comparison, offloading sequential decision-making tasks introduces longer-term consequences that require a more sophisticated metareasoning framework.

Further work into this setting would allow metareasoning about systems that rely on offloading computation to remote computers or other agents. As well as reasoning about the costs and benefits of offloading computation, this would also require reasoning about the communication costs and constraints of the system. One example of this is long mission duration underwater robots, the decision-making for which is carried out remotely and sent to the vehicle via a low-bandwidth satellite link. This setting introduces new costs and constraints relating to communication abilities to the metalevel control problem. For example, in the underwater vehicle example, the satellite link only works when the vehicle is at the surface and incurs monetary costs per byte of data sent. Methods based on decentralised POMDPs (Spaan et al., 2008; Wu et al., 2011) could be integrated into a multi-agent metalevel control framework to reason about communication costs and constraints.

Metareasoning Generalisation and Scalability In Chapters 6 and 7, we defined distributions of object-level problems and tested trained metalevel policies on previously unseen object-level problems drawn from the same distributions.

Therefore we did not test generalisation abilities: for example, we did not test the performance of metalevel policies trained only with small object-level problems on a distribution that included larger object-level problems than those seen during training. It is almost certain that starting with a metalevel policy trained on small object-level problems and then fine-tuning it on larger object-level problems would be more efficient than training a metalevel policy from scratch (Svegliato et al., 2020). This raises the question of whether it is possible to train general metalevel policies that can be fine-tuned on a small number of object-level problems, rather than training a metalevel policy for each new distribution of object-level problems. The structure of such a general metalevel policy could provide insight into common patterns of reasoning across different object-level problems.

Offline RL offers a potential avenue for reducing the training burden of running object-level algorithms during metalevel training. However, no works to date have investigated the use of offline RL for metalevel control. One question to be investigated is whether datasets of historical reasoning traces would allow learning of metalevel policies that differ from the reasoning approaches taken to generate the dataset. For example, if the dataset was generated by a behaviour policy that always reasoned until solution convergence, there would be a large distribution shift between this behaviour policy and a metalevel control policy that could stop computation at any time.

Finally, future work could look into tighter integration between metalevel control and object-level reasoning. This may be able to offer some of the benefits of fully integrated metalevel control, while still allowing control of arbitrary object-level reasoning algorithms.

Bibliography

Yossiri Adulyasak, Pradeep Varakantham, Asrar Ahmed, and Patrick Jaillet. Solving uncertain MDPs with objectives that are separable over instantiations of model uncertainty. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

Aliakbar Akbari and Sara Bernardini. Informed autonomous exploration of subterranean environments. *IEEE Robotics and Automation Letters*, 6(4):7957–7964, 2021.

Rika Antonova, Silvia Cruciani, Christian Smith, and Danica Kragic. Reinforcement learning for pivoting task. *arXiv preprint arXiv:1703.00472*, 2017.

Dilip Arumugam and Satinder Singh. Planning to the information horizon of BAMDPs via epistemic state abstraction. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:20482–20497, 2022.

Yarden As, Ilnura Usmanova, Sebastian Curi, and Andreas Krause. Constrained policy optimization via bayesian world models. In *International Conference on Learning Representations (ICLR)*, 2022.

Thom Badings, Licio Romao, Alessandro Abate, and Nils Jansen. Probabilities are not enough: Formal controller synthesis for stochastic dynamical models with epistemic uncertainty. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 14701–14710, 2023.

Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

Chris Baraniuk. Electricity grids creak as AI demands soar. *BBC News*, 2024. URL `https://www.bbc.co.uk/news/articles/cj5ll89dy2mo`. Accessed: 2025-04-10.

Andrew G Barto, Steven J Bradtke, and Satinder P Singh. Learning to act using real-time dynamic programming. *Artificial intelligence*, 72(1-2):81–138, 1995.

Ana Batinovic, Tamara Petrovic, Antun Ivanovic, Frano Petric, and Stjepan Bogdan. A multi-resolution frontier-based planner for autonomous 3D exploration. *IEEE Robotics and Automation Letters*, 6(3):4528–4535, 2021.

Richard Bellman. Dynamic programming. *science*, 153(3731):34–37, 1966.

Felix Berkenkamp, Matteo Turchetta, Angela P Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 908–919, 2017.

Abhinav Bhatia, Justin Svegliato, Samer B Nashed, and Shlomo Zilberstein. Tuning the hyperparameters of anytime planning: a metareasoning approach with deep reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 32, pages 556–564, 2022.

Antoine Blanchard and Themistoklis Sapsis. Informative path planning for anomaly detection in environment exploration and monitoring. *Ocean Engineering*, 243: 110242, 2022.

Blai Bonet and Hector Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 3, pages 12–21, 2003.

Alessandro Bottero, Carlos Luis, Julia Vinogradska, Felix Berkenkamp, and Jan R Peters. Information-theoretic safe exploration with Gaussian processes. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:30707–30719, 2022.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI gym. *arXiv preprint arXiv:1606.01540*, 2016.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. RT-1: robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

Adam Bry and Nicholas Roy. Rapidly-exploring random belief trees for motion planning under uncertainty. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 723–730. IEEE, 2011.

Matthew Budd, Bruno Lacerda, Paul Duckworth, Andrew West, Barry Lennox, and Nick Hawes. Markov decision processes with unknown state feature values for safe exploration using Gaussian processes. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

Matthew Budd, Georgios Salavasidis, Izzat Karnarudzaman, Catherine A Harris, Alexander B Phillips, Paul Duckworth, Nick Hawes, and Bruno Lacerda. Probabilistic planning for AUV data harvesting from smart underwater sensor networks. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

Matthew Budd, Paul Duckworth, Nick Hawes, and Bruno Lacerda. Bayesian reinforcement learning for single-episode missions in partially unknown environments. In *Conference on Robot Learning (CoRL)*. PMLR, 2023. URL `https://proceedings.mlr.press/v205/budd23a/budd23a.pdf`.

Matthew Budd, Bruno Lacerda, and Nick Hawes. Stop! Planner Time: metareasoning for probabilistic planning using learned performance profiles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20053–20060, 2024.

Matthew Budd, Bruno Lacerda, and Nick Hawes. Think Fast! Learning to control online reasoning in stochastic environments. *Under review*, 2025a.

Matthew Budd, Georgios Salavasidis, Izzat Kamarudzaman, Benjamin Sherlock, Jeffrey Neasham, Bruno Lacerda, Nick Hawes, Alexander B. Phillips, and Catherine Harris. AUV-based data harvesting from underwater sensor networks using probabilistic planning. *Under review, Journal of Field Robotics*, 2025b.

Ethan Burns, Wheeler Ruml, and Minh Binh Do. Heuristic search when time matters. *Journal of Artificial Intelligence Research (JAIR)*, 47:697–740, 2013.

F Callaway, S Gul, P Krueger, TL Griffiths, and F Lieder. Learning to select computations. In *34th Conference on Uncertainty in Artificial Intelligence (UAI 2018)*, pages 776–785, 2018.

Yuhong Cao, Yizhuo Wang, Apoorva Vashisth, Haolin Fan, and Guillaume Adrien Sartoretti. CAtNIPP: Context-aware attention-based network for informative path planning. In *Conference on Robot Learning (CoRL)*, pages 1928–1937. PMLR, 2023.

Michael Cashmore, Andrew Coles, Bence Cserna, Erez Karpas, Daniele Magazzeni, and Wheeler Ruml. Temporal planning while the clock ticks. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 28, 2018.

Michael Cashmore, Andrew Coles, Bence Cserna, Erez Karpas, Daniele Magazzeni, and Wheeler Ruml. Replanning for situated robots. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 29, pages 665–673, 2019.

Xiong-Hui Chen, Yang Yu, Qingyang Li, Fan-Ming Luo, Zhiwei Qin, Wenjie Shang, and Jieping Ye. Offline model-based adaptable policy learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:8432–8443, 2021.

Wang Chi Cheung, David Simchi-Levi, and Ruihao Zhu. Reinforcement learning for non-stationary Markov decision processes: The blessing of (More) optimism. In Hal Daumé III and Aarti Singh, editors, *International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 1843–1854. PMLR, 13–18 Jul 2020. URL `https://proceedings.mlr.press/v119/cheung20a.html`.

Sandeep Chinchali, Apoorva Sharma, James Harrison, Amine Elhafsi, Daniel Kang, Evgenya Pergament, Eyal Cidon, Sachin Katti, and Marco Pavone. Network offloading policies for cloud robotics: a learning-based approach. *Autonomous Robots*, pages 1–16, 2021.

Yinlam Chow and Mohammad Ghavamzadeh. Algorithms for CVaR optimization in MDPs. *Advances in Neural Information Processing Systems (NeurIPS)*, 27, 2014.

Dylan Cope, Justin Svegliato, and Stuart Russell. Learning to plan with tree search via deep RL. In *Bridging the Gap Between AI Planning and Reinforcement Learning Workshop (PRL@ IJCAI 2023) at the International Joint Conference on Artificial Intelligence*, 2023.

Adrien Couëtoux, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, and Nicolas Bonnard. Continuous upper confidence trees. In *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*, pages 433–445. Springer, 2011.

Michael T Cox and Anita Raja. *Metareasoning: Thinking about thinking*. MIT Press, 2011.

Peng Dai, Daniel S Weld, Judy Goldsmith, et al. Topological value iteration algorithms. *Journal of Artificial Intelligence Research (JAIR)*, 42:181–209, 2011.

Patrick Dallaire, Camille Besse, Stephane Ross, and Brahim Chaib-draa. Bayesian reinforcement learning in continuous POMDPs with Gaussian processes. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2604–2609. IEEE, 2009.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

Ron Dorfman, Idan Shenfeld, and Aviv Tamar. Offline meta learning of exploration. *arXiv preprint arXiv:2008.02598*, 2020.

Michael O'Gordon Duff. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. University of Massachusetts Amherst, 2002.

Amihay Elboher, Ava Bensoussan, Erez Karpas, Wheeler Ruml, Shahaf S Shperberg, and Eyal Shimony. A formal metareasoning model of concurrent planning and execution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12427–12435, 2023.

Jonathan Fink and Vijay Kumar. Online methods for radio signal mapping with mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1940–1945. IEEE, 2010.

G. Flaspohler, V. Preston, A. P. M. Michel, Y. Girdhar, and N. Roy. Information-guided robotic maximum seek-and-sample in partially observable continuous environments. *IEEE Robotics and Automation Letters*, 4(4):3782–3789, 2019.

Vojtěch Forejt, Marta Kwiatkowska, and David Parker. Pareto curves for probabilistic model checking. In *Automated Technology for Verification and Analysis: 10th International Symposium, ATVA 2012, Thiruvananthapuram, India, October 3-6, 2012. Proceedings 10*, pages 317–332. Springer, 2012.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018.

Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, and Aviv Tamar. Bayesian reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 8(5-6):359–483, 2015.

Dibya Ghosh, Jad Rahme, Aviral Kumar, Amy Zhang, Ryan P Adams, and Sergey Levine. Why generalization in RL is difficult: Epistemic POMDPs and implicit partial observability. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:25502–25515, 2021.

Dibya Ghosh, Anurag Ajay, Pulkit Agrawal, and Sergey Levine. Offline RL policies should be trained to be adaptive. In *International Conference on Machine Learning (ICML)*, pages 7513–7530. PMLR, 2022.

Robert Givan, Sonia Leach, and Thomas Dean. Bounded-parameter Markov decision processes. *Artificial Intelligence*, 122(1-2):71–109, 2000.

Prakhar Godara, Tilman Diego Aléman, and Angela J Yu. Metareasoning in uncertain environments: a meta-BAMDP framework. *arXiv preprint arXiv:2408.01253*, 2024.

Javier González, Michael Osborne, and Neil Lawrence. GLASSES: Relieving the myopia of Bayesian optimisation. In *Artificial Intelligence and Statistics*, pages 790–799. PMLR, 2016.

Thomas L Griffiths, Frederick Callaway, Michael B Chang, Erin Grant, Paul M Krueger, and Falk Lieder. Doing more with less: meta-reasoning and meta-learning in humans and machines. *Current Opinion in Behavioral Sciences*, 29: 24–30, 2019.

Shangding Gu, Long Yang, Yali Du, Guang Chen, Florian Walter, Jun Wang, and Alois Knoll. A review of safe reinforcement learning: Methods, theories and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

Tianyi Gu, Wheeler Ruml, Shahaf S Shperberg, Eyal Solomon Shimony, and Erez Karpas. When to commit to an action in online planning and search. In *Proceedings of the International Symposium on Combinatorial Search*, volume 15, pages 83–90, 2022.

Arthur Guez, David Silver, and Peter Dayan. Scalable and efficient Bayes-adaptive reinforcement learning based on Monte-Carlo tree search. *Journal of Artificial Intelligence Research (JAIR)*, 48:841–883, 2013.

Arthur Guez, Nicolas Heess, David Silver, and Peter Dayan. Bayes-adaptive simulation-based search with value function approximation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 27, pages 451–459, 2014a.

Arthur Guez, David Silver, and Peter Dayan. Better optimism by Bayes: Adaptive planning with rich models. *arXiv preprint arXiv:1402.1958*, 2014b.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, pages 1861–1870. PMLR, 2018.

Danijar Hafner, Timothy P Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations (ICLR)*, 2021.

Brian Ferris Dirk Hähnel and Dieter Fox. Gaussian processes for signal strength-based location estimation. In *Robotics: Science and Systems*, 2006.

Eric A Hansen and Shlomo Zilberstein. Monitoring and control of anytime algorithms: a dynamic programming approach. *Artificial Intelligence*, 126(1-2): 139–157, 2001.

Nicholas Hay, Stuart Russell, David Tolpin, and Solomon Eyal Shimony. Selecting computations: theory and applications. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, pages 346–355, 2012.

Michael Ho, Sami El-Borgi, Devendra Patil, and Gangbing Song. Inspection and monitoring systems subsea pipelines: a review paper. *Structural Health Monitoring*, 19(2):606–645, 2020.

Marcus Hoerger, Hanna Kurniawati, Dirk Kroese, and Nan Ye. A surprisingly simple continuous-action POMDP solver: lazy cross-entropy search over policy trees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20134–20142, 2024.

Eric Horvitz. Reasoning under varying and uncertain resource constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 88, pages 111–116, 1988.

Mahdi Imani, Seyede Fatemeh Ghoreishi, and Ulisses M Braga-Neto. Bayesian control of large MDPs with unknown dynamics in data-poor environments. *Advances in Neural Information Processing Systems (NeurIPS)*, 31:8146–8156, 2018.

Garud N Iyengar. Robust dynamic programming. *Mathematics of Operations Research*, 30(2):257–280, 2005.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *AIJ*, 101(1-2), 1998.

Sammie Katt, Frans A Oliehoek, and Christopher Amato. Learning in POMDPs with Monte Carlo tree search. In *International Conference on Machine Learning (ICML)*, pages 1819–1827. PMLR, 2017.

Thomas Keller and Malte Helmert. Trial-based heuristic tree search for finite horizon MDPs. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 23, pages 135–143, 2013.

Bassam A. Khuwaileh and Walid A. Metwally. Gaussian process approach for dose mapping in radiation fields. *Nuclear Engineering and Technology*, 2020. ISSN 1738-5733. doi: https://doi.org/10.1016/j.net.2020.01.013. URL `https://www.sciencedirect.com/science/article/pii/S1738573319305935`.

Beomjoon Kim, Kyungjae Lee, Sungbin Lim, Leslie Kaelbling, and Tomás Lozano-Pérez. Monte Carlo tree search in continuous spaces using Voronoi optimistic optimization with regret bounds. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9916–9924, 2020.

Diederik P Kingma, Max Welling, et al. Auto-encoding variational Bayes, 2013.

D Kinny and M George. Commitment and effectiveness of situated agents. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI'91)*, pages 82–88, 1991.

Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning (ECML)*, pages 282–293. Springer, 2006.

Lars Kunze, Mohan Sridharan, Christes Dimitrakakis, and Jeremy Wyatt. Adaptive sampling-based view planning under time constraints. In *2017 European Conference on Mobile Robots (ECMR)*, pages 1–6. IEEE, 2017.

Malte Kuss and Carl Rasmussen. Gaussian processes in reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 16, 2003.

Bruno Lacerda, David Parker, and Nick Hawes. Multi-objective policy generation for mobile robots under probabilistic time-bounded guarantees. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 27, pages 504–512, 2017.

Bruno Lacerda, Fatma Faruq, David Parker, and Nick Hawes. Probabilistic planning with formal performance guarantees for mobile service robots. *The International Journal of Robotics Research (IJRR)*, 38(9), 2019.

Pawel Ladosz, Hyondong Oh, Gan Zheng, and Wen-Hua Chen. Gaussian process based channel prediction for communication-relay UAV in urban environments. *IEEE Transactions on Aerospace and Electronic Systems*, 56(1):313–325, 2019.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Michael H Lim, Claire J Tomlin, and Zachary N Sunberg. Voronoi progressive widening: efficient online solvers for continuous state, action, and observation POMDPs. In *2021 60th IEEE conference on decision and control (CDC)*, pages 4493–4500. IEEE, 2021.

Christopher H Lin, Andrey Kolobov, Ece Kamar, and Eric Horvitz. Metareasoning for planning under uncertainty. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, 2015.

Liuzixuan Lin, Rajini Wijayawardana, Varsha Rao, Hai Nguyen, Emmanuel Wedan GNIBGA, and Andrew A Chien. Exploding AI power use: an opportunity to rethink grid planning and management. In *Proceedings of the 15th ACM International Conference on Future and Sustainable Energy Systems*, pages 434–441, 2024.

Michael L Littman, Anthony R Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *Machine Learning Proceedings 1995*, pages 362–370. Elsevier, 1995.

Mausam and Andrey Kolobov. *Planning with Markov decision processes: An AI perspective*. Morgan & Claypool Publishers, 2012.

H Brendan McMahan, Maxim Likhachev, and Geoffrey J Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *International Conference on Machine Learning (ICML)*, pages 569–576, 2005.

Jiawei Meng, Yuanchang Liu, Richard Bucknall, Weihong Guo, and Ze Ji. Anisotropic GPMP2: a fast continuous-time Gaussian processes based motion planner for unmanned surface vehicles in environments with ocean currents. *IEEE Transactions on Automation Science and Engineering*, 19(4):3914–3931, 2022.

John Mern, Anil Yildiz, Zachary Sunberg, Tapan Mukerji, and Mykel J Kochenderfer. Bayesian optimized Monte Carlo planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11880–11887, 2021.

Andrew Mitchell, Wheeler Ruml, Fabian Spaniol, Jorg Hoffmann, and Marek Petrik. Real-time planning as decision-making under uncertainty. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2338–2345, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Teodor Mihai Moldovan and Pieter Abbeel. Safe exploration in Markov decision processes. In *International Conference on Machine Learning (ICML)*, pages 1451–1458, 2012.

Manfred Morari and Jay H Lee. Model predictive control: past, present and future. *Computers & chemical engineering*, 23(4-5):667–682, 1999.

Philippe Morere, Roman Marchant, and Fabio Ramos. Sequential Bayesian optimization as a POMDP for environment monitoring with UAVs. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6381–6388. IEEE, 2017.

Mustafa Mukadam, Xinyan Yan, and Byron Boots. Gaussian process motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 9–15. IEEE, 2016.

Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In *International Conference on Learning Representations*, 2018.

Mozhgan Navardi, Edward Humes, and Tinoosh Mohsenin. MetaTinyML: End-to-end metareasoning framework for TinyML platforms. *IEEE Embedded Systems Letters*, 16(4):393–396, 2024.

Arnab Nilim and Laurent El Ghaoui. Robust control of Markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798, 2005.

Dylan O'Ceallaigh and Wheeler Ruml. Metareasoning in real-time heuristic search. In *Proceedings of the International Symposium on Combinatorial Search*, volume 6, pages 87–95, 2015.

Ian Osband and Benjamin Van Roy. Why is posterior sampling better than optimism for reinforcement learning? In *International Conference on Machine Learning (ICML)*, pages 2701–2710. PMLR, 2017.

Ian Osband, Daniel Russo, and Benjamin Van Roy. (More) efficient reinforcement learning via posterior sampling. *Advances in Neural Information Processing Systems (NeurIPS)*, 26, 2013.

Michael Painter, Mohamed Baioumy, Nick Hawes, and Bruno Lacerda. Monte Carlo tree search with Boltzmann exploration. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:78181–78192, 2023.

Priyam Parashar, Ashok K Goel, Bradley Sheneman, and Henrik I Christensen. Towards life-long adaptive agents: using metareasoning for combining knowledge-based planning with situated learning. *The Knowledge Engineering Review*, 33: e24, 2018.

Miles Pebody. The contribution of scripted command sequences and low level control behaviours to autonomous underwater vehicle control systems and their impact on reliability and mission success. In *OCEANS 2007-Europe*, pages 1–5. IEEE, 2007.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3803–3810. IEEE, 2018.

Manoj Penmetcha and Byung-Cheol Min. A deep reinforcement learning-based dynamic computational offloading method for cloud robotics. *IEEE Access*, 9: 60265–60279, 2021.

Alexander B Phillips, Robert Templeton, Daniel Roper, Richard Morrison, Miles Pebody, Philip M Bagley, Rachel Marlow, Ed Chaney, James Burris, Alberto Consensi, et al. Autosub long range 1500: a continuous 2000 km field trial. *Ocean Engineering*, 280:114626, 2023.

Joelle Pineau, Geoff Gordon, Sebastian Thrun, et al. Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, volume 3, pages 1025–1032, 2003.

Joelle Pineau, Geoffrey Gordon, and Sebastian Thrun. Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research (JAIR)*, 27:335–380, 2006.

Kyriakos Polymenakos, Alessandro Abate, and Stephen Roberts. Safe policy search using gaussian process models. In *Proceedings of the 18th international conference on autonomous agents and multiagent systems*, pages 1565–1573, 2019.

Vitchyr H Pong, Ashvin V Nair, Laura M Smith, Catherine Huang, and Sergey Levine. Offline meta-reinforcement learning with online self-supervision. In *International Conference on Machine Learning (ICML)*, pages 17811–17829. PMLR, 2022.

Manish Prajapat, Matteo Turchetta, Melanie Zeilinger, and Andreas Krause. Near-optimal multi-agent learning for safe coverage control. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:14998–15012, 2022.

Manish Prajapat, Johannes Köhler, Matteo Turchetta, Andreas Krause, and Melanie N Zeilinger. Safe guaranteed exploration for non-linear systems. *IEEE Transactions on Automatic Control*, 2025.

Martin L Puterman. Markov decision processes: Discrete stochastic dynamic programming, 1994.

Anita Raja and Victor Lesser. A framework for meta-level control in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 15:147–196, 2007.

Fabio Ramos, Rafael Possas, and Dieter Fox. BayesSim: Adaptive domain randomization via probabilistic inference for robotics simulators. *Robotics: Science and Systems XV*, 2019.

CE. Rasmussen and CKI. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. ISBN 9780262182539.

Marc Rigter, Bruno Lacerda, and Nick Hawes. Minimax regret optimisation for robust planning in uncertain Markov decision processes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11930–11938, 2021a.

Marc Rigter, Bruno Lacerda, and Nick Hawes. Risk-averse bayes-adaptive reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:1142–1154, 2021b.

Zohar Rimon, Tom Jurgenson, Orr Krupnik, Gilad Adler, and Aviv Tamar. Mamba: an effective world model approach for meta-reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2024.

Stephane Ross, Brahim Chaib-draa, and Joelle Pineau. Bayes-adaptive POMDPs. In *NIPS*, pages 1225–1232, 2007.

Stuart Russell and Eric Wefald. Principles of metareasoning. *Artificial intelligence*, 49(1-3):361–395, 1991.

Stuart J Russell and Devika Subramanian. Provably bounded-optimal agents. *Journal of Artificial Intelligence Research (JAIR)*, 2:575–609, 1994.

Daniel J Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, Zheng Wen, et al. A tutorial on thompson sampling. *Foundations and Trends® in Machine Learning*, 11(1):1–96, 2018.

Yves-Laurent Kom Samo and Stephen J Roberts. String and membrane gaussian processes. *Journal of Machine Learning Research*, 17(131):1–87, 2016.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

Matthias Seeger et al. Low rank updates for the cholesky decomposition. *University of California at Berkeley, Tech. Rep*, 2004.

Eren Sezener and Peter Dayan. Static and dynamic values of computation in MCTS. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 31–40. PMLR, 2020.

Shahaf S Shperberg, Andrew Coles, Bence Cserna, Erez Karpas, Wheeler Ruml, and Solomon E Shimony. Allocating planning effort when actions expire. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2371–2378, 2019.

Shahaf S Shperberg, Andrew Coles, Erez Karpas, Solomon Eyal Shimony, and Wheeler Ruml. Trading plan cost for timeliness in situated temporal planning. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI'20)*, pages 4176–4182, 2020.

Yaroslava Shynkar, Anita Raja, Ana LC Bazzan, and Marin Marinov. Multiagent meta-level control for adaptive traffic systems: a case study. *Transportation Research Procedia*, 62:236–244, 2022.

Harshit Sikchi, Wenxuan Zhou, and David Held. Learning off-policy with online planning. In *Conference on Robot Learning (CoRL)*, pages 1622–1633. PMLR, 2022.

David Silver and Joel Veness. Monte-Carlo planning in large POMDPs. *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1–9, 2010. ISSN 1049-5258.

Fergus Simpson, Vidhi Lalchand, and Carl Edward Rasmussen. Marginalised gaussian processes with nested sampling. *Advances in neural information processing systems*, 34:13613–13625, 2021.

Patrick Slade, Preston Culbertson, Zachary Sunberg, and Mykel Kochenderfer. Simultaneous active parameter estimation and control using sampling-based Bayesian reinforcement learning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 804–810. IEEE, 2017.

Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. *Advances in Neural Information Processing Systems (NeurIPS)*, 18, 2005.

Christian Soize. *Uncertainty quantification*. Springer, 2017.

Matthijs TJ Spaan, Frans A Oliehoek, and Nikos Vlassis. Multiagent planning under uncertainty with stochastic communication delays. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 338–345, 2008.

Sharan Srinivas, Surya Ramachandiran, and Suchithra Rajendran. Autonomous robot-driven deliveries: a review of recent developments and future directions. *Transportation research part E: logistics and transportation review*, 165:102834, 2022.

Alex Stephens, Matthew Budd, Michal Staniaszek, Benoit Casseau, Paul Duckworth, Maurice Fallon, Nick Hawes, and Bruno Lacerda. Planning under uncertainty for safe robot exploration using Gaussian process prediction. *Autonomous Robots*, 48 (7):18, 2024.

Yanan Sui, Alkis Gotovos, Joel W. Burdick, and Andreas Krause. Safe exploration for optimization with Gaussian processes. In *International Conference on Machine Learning (ICML)*, page 997–1005. JMLR.org, 2015.

Marnix Suilen, Thom Badings, Eline M Bovy, David Parker, and Nils Jansen. Robust Markov decision processes: a place where AI and formal methods meet. In *Principles of Verification: Cycling the Probabilistic Landscape: Essays Dedicated to Joost-Pieter Katoen on the Occasion of His 60th Birthday, Part III*, pages 126–154. Springer, 2024.

Timothy John Sullivan. *Introduction to uncertainty quantification*, volume 63. Springer, 2015.

Zachary Sunberg and Mykel Kochenderfer. Online algorithms for POMDPs with continuous state, action, and observation spaces. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2018.

Yoonchang Sung, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning when to quit: meta-reasoning for motion planning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4692–4699. IEEE, 2021.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Justin Svegliato, Kyle Hollins Wray, and Shlomo Zilberstein. Meta-level control of anytime algorithms with online performance prediction. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI'18)*, 2018.

Justin Svegliato, Prakhar Sharma, and Shlomo Zilberstein. A model-free approach to meta-level control of anytime algorithms. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 11436–11442. IEEE, 2020.

Andreas Svensson, Johan Dahlin, and Thomas B Schön. Marginalizing gaussian process hyperparameters using sequential monte carlo. In *2015 IEEE 6th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, pages 477–480. IEEE, 2015.

Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. Monte Carlo tree search: a review of recent modifications and applications. *Artificial Intelligence Review*, 56(3):2497–2562, 2023.

Wil Thomason, Marlin P Strub, and Jonathan D Gammell. Task and motion informed trees (TMIT*): Almost-surely asymptotically optimal integrated task and motion planning. *IEEE Robotics and Automation Letters*, 7(4):11370–11377, 2022.

Michalis Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *Artificial intelligence and statistics*, pages 567–574. PMLR, 2009.

M Tranzatto, F Mascarich, L Bernreiter, C Godinho, M Camurri, S Khattak, T Dang, V Reijgwart, J Löje, D Wisth, et al. Cerberus: autonomous legged and aerial robotic exploration in the tunnel and urban circuits of the darpa subterranean challenge. *Field Robotics*, 2, 2022.

Ioannis Tsitsimpelis, C James Taylor, Barry Lennox, and Malcolm J Joyce. A review of ground-based robotic systems for the characterization of nuclear environments. *Progress in nuclear energy*, 111:109–124, 2019.

Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. Safe exploration in finite Markov decision processes with Gaussian processes. In *Advances in Neural Information Processing Systems (NeurIPS)*, NIPS'16, pages 4312–4320, 2016. ISBN 978-1-5108-3881-9.

Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. Safe exploration for interactive machine learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 2891–2901, 2019.

Akifumi Wachi and Yanan Sui. Safe reinforcement learning in constrained Markov decision processes. In *International Conference on Machine Learning (ICML)*, pages 9797–9806. PMLR, 2020.

Akifumi Wachi, Yanan Sui, Yisong Yue, and Masahiro Ono. Safe exploration and optimization of constrained MDPs using Gaussian processes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Jack Wang, Aaron Hertzmann, and David J Fleet. Gaussian process dynamical models. *Advances in neural information processing systems*, 18, 2005.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:24824–24837, 2022.

Andrew West, Ioannis Tsitsimpelis, Mauro Licata, Anze Jazbec, Luka Snoj, Malcolm J Joyce, and Barry Lennox. Use of gaussian process regression for radiation mapping of a nuclear reactor with a mobile robot. *Scientific reports*, 11 (1):13975, 2021.

Andrew Gordon Wilson, Christoph Dann, and Hannes Nickisch. Thoughts on massively scalable gaussian processes. *ArXiv*, abs/1511.01870, 2015. URL `https://api.semanticscholar.org/CorpusID:11198138`.

James Wilson, Viacheslav Borovitskiy, Alexander Terenin, Peter Mostowsky, and Marc Deisenroth. Efficiently sampling functions from Gaussian process posteriors. In *International Conference on Machine Learning (ICML)*, pages 10292–10302. PMLR, 2020.

Michael Wooldridge. Agent-based software engineering. *IEE Proceedings-software*, 144(1):26–37, 1997.

Feng Wu, Shlomo Zilberstein, and Xiaoping Chen. Online planning for multi-agent systems with bounded communication. *Artificial Intelligence*, 175(2):487–511, 2011.

Nan Ye, Adhiraj Somani, David Hsu, and Wee Sun Lee. DESPOT: Online POMDP planning with regularization. *Journal of Artificial Intelligence Research (JAIR)*, 58:231–266, Jan 2017. ISSN 1076-9757. doi: 10.1613/jair.5328. URL `http://dx.doi.org/10.1613/jair.5328`.

Sung Wook Yoon, Alan Fern, and Robert Givan. FF-Replan: a baseline for probabilistic planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 7, pages 352–359, 2007.

Sung Wook Yoon, Alan Fern, Robert Givan, and Subbarao Kambhampati. Probabilistic planning via determinization in hindsight. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1010–1016, 2008.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. MOPO: Model-based offline policy optimization. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:14129–14142, 2020.

Enrica Zereik, Marco Bibuli, Nikola Mišković, Pere Ridao, and António Pascoal. Challenges and future trends in marine robotics. *Annual Reviews in Control*, 46: 350–368, 2018.

Xianyuan Zhan, Xiangyu Zhu, and Haoran Xu. Model-based offline planning with trajectory pruning. *arXiv preprint arXiv:2105.07351*, 2021.

Shlomo Zilberstein. Operational rationality through compilation of anytime algorithms. *AI Magazine*, 16(2):79–79, 1995.

L Zintgraf, K Shiarlis, M Igl, S Schulze, Y Gal, K Hofmann, and S Whiteson. VariBAD: a very good method for Bayes-adaptive deep RL via meta-learning. International Conference on Learning Representations (ICLR), 2020.

A Appendices of Chapter 3

AUV-Based Data Harvesting from Underwater Sensor Networks Using Probabilistic Planning

Matthew Budd
Oxford Robotics Institute
University of Oxford
mbudd@robots.ox.ac.uk

Georgios Salavasidis
National Oceanography Centre
Southampton, UK
geosal@noc.ac.uk

Izzat Kamarudzaman
National Oceanography Centre
Southampton, UK
Izzat.Kamarudzaman@noc.ac.uk

Benjamin Sherlock
School of Engineering
Newcastle University
Newcastle-upon-Tyne, UK
benjamin.sherlock@ncl.ac.uk

Jeffrey Neasham
School of Engineering
Newcastle University
Newcastle-upon-Tyne, UK
jeff.neasham@ncl.ac.uk

Bruno Lacerda
Oxford Robotics Institute
University of Oxford
bruno@robots.ox.ac.uk

Nick Hawes
Oxford Robotics Institute
University of Oxford
nickh@robots.ox.ac.uk

Alexander B. Phillips
National Oceanography Centre
Southampton, UK
abp@noc.ac.uk

Catherine Harris
National Oceanography Centre
Southampton, UK
catherine.harris@noc.ac.uk

Abstract

Underwater acoustic and wireless sensor networks enable transformative spatio-temporal ocean data collection for applications such as climate monitoring, marine life analysis, and industrial diagnostics. However, retrieving data from submerged sensor nodes in near real-time remains challenging due to environmental constraints and the high costs of traditional methods, such as deploying research vessels. This work presents an integrated adaptive framework that combines modelling approaches and planning under uncertainty to enable an Autonomous Underwater Vehicle (AUV) to serve as a mobile sink for sparse underwater sensor networks. The framework dynamically prioritises nodes based on data value, and utilises acoustic communication links for simultaneous data harvesting and time-of-flight localisation. Key contributions include the systems integration effort that addresses real-world operational challenges, a novel modelling approach that underpins the planning methodology, and extensive field trials validating the framework's performance. The results demonstrate significant improvements over conventional hand-designed behaviours for underwater data retrieval, advancing sustainable and autonomous ocean monitoring solutions through robust integration, innovative planning, and real-world validation.

Keywords: Underwater Sensor Networks, Autonomous Underwater Vehicles, Probabilistic Planning, Data Harvesting, AUV navigation, Underwater communication, Robot decision-making

Appendices for:

Matthew Budd, Georgios Salavasidis, Izzat Kamarudzaman, Benjamin Sherlock, Jeffrey Neasham, Bruno Lacerda, Nick Hawes, Alexander B. Phillips, and Catherine Harris. AUV-based data harvesting from underwater sensor networks using probabilistic planning. *Under review, Journal of Field Robotics*, 2025b

A Experiment and planner configuration details

A.1 Planner common parameters

The fixed number of clusters $|\hat{T}|$ when clustering action durations in the TMDP was set to 6. The comms model belief for all experiments consisted of a uniform belief between 110 dB and 120 dB (re 1uPa @ 1m) for environment acoustic noise, and a categorial belief over multipath interference levels of `low`, and `high` with equal probability. The navigation TMDP timestep was set to 40 s for all experiments.

A.2 Scenario 1

This scenario was used in simulation experiments (Section 5.2.1) and for the field trial unknown data contents case (Section 6.4.1).

Node	Measurement data (B)	Event data (B)
N2	400	120
N3	400	80
N4	400	280
N5	400	400
N6	400	900

Table 9: Initial data contents of nodes for unknown data distribution experiments.

Each node was modelled with $|S_{\phi,d_e}| = 4$ and $|S_{\phi,d_m}| = 2$, where the expected number of bytes of measurement data is 400 and increasingly large bins covering between 0 and 1000 bytes of event data. For the unknown data contents case, the prior belief over node data contents was equal for each node and assigned higher probabilities to lower event data contents.

A.3 Scenario 2

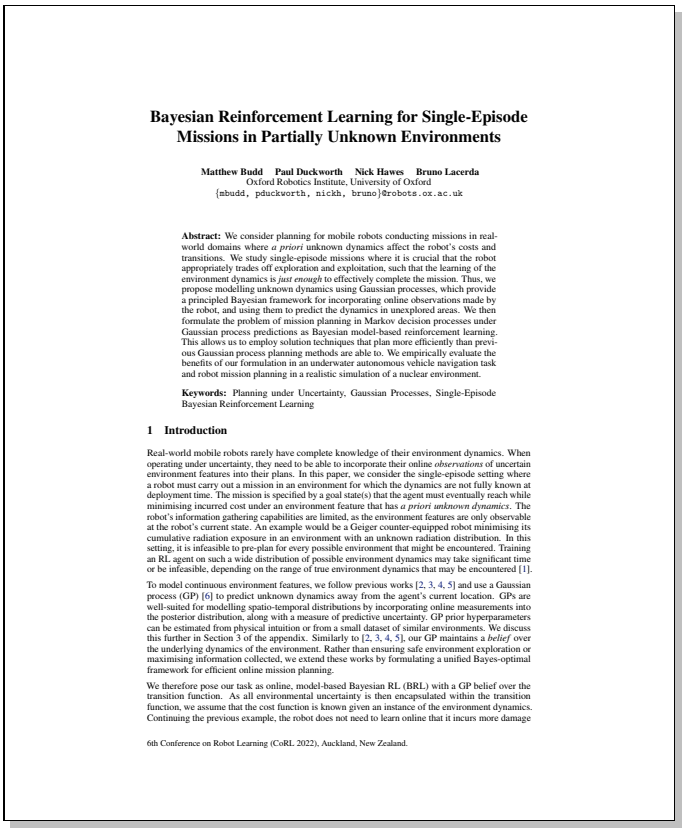
This scenario was used in the field trial known data contents case (Section 6.4.2).

Node	Measurement data (B)	Event data (B)
N2	960	160
N3	416	500
N4	1060	160
N5	400	320
N6	976	80

Table 10: Initial data contents of nodes for unknown data distribution experiments.

Each node was modelled with $|S_{\phi,d_e}| = 4$ and $|S_{\phi,d_m}| = 2$, where each block of event data was 250 bytes large and each block of measurement data was 600 bytes large.

B Appendices of Chapter 5



Bayesian Reinforcement Learning for Single-Episode Missions in Partially Unknown Environments

Matthew Budd Paul Duckworth Nick Hawes Bruno Lacerda
Oxford Robotics Institute, University of Oxford
{mbudd, pduckworth, nickh, bruno}@robots.ox.ac.uk

Abstract: We consider planning for mobile robots conducting missions in real-world domains where *a priori* unknown dynamics affect the robot's costs and transitions. We study single-episode missions where it is crucial that the robot appropriately trades off exploration and exploitation, such that the learning of the environment dynamics is *just enough* to effectively complete the mission. Thus, we propose modelling unknown dynamics using Gaussian processes, which provide a principled Bayesian framework for incorporating online observations made by the robot, and using them to predict the dynamics in unexplored areas. We then formulate the problem of mission planning in Markov decision processes under Gaussian process predictions as Bayesian model-based reinforcement learning. This allows us to employ solution techniques that plan more efficiently than previous Gaussian process planning methods are able to. We empirically evaluate the benefits of our formulation in an underwater autonomous vehicle navigation task and robot mission planning in a realistic simulation of a nuclear environment.

Keywords: Planning under Uncertainty, Gaussian Processes, Single-Episode Bayesian Reinforcement Learning

1 Introduction

Real-world mobile robots rarely have complete knowledge of their environment dynamics. When operating under uncertainty, they need to be able to incorporate their online *observations* of uncertain environment features into their plans. In this paper, we consider the single-episode setting where a robot must carry out a mission in an environment for which the dynamics are not fully known at deployment time. The mission is specified by a goal state(s) that the agent must eventually reach while minimising incurred cost under an environment feature that has *a priori unknown dynamics*. The robot's information gathering capabilities are limited, as the environment features are only observable at the robot's current state. An example would be a Geiger counter-equipped robot minimising its cumulative radiation exposure in an environment with an unknown radiation distribution. In this setting, it is infeasible to pre-plan for every possible environment that might be encountered. Training an RL agent on such a wide distribution of possible environment dynamics may take significant time or be infeasible, depending on the range of true environment dynamics that may be encountered [1].

To model continuous environment features, we follow previous works [2, 3, 4, 5] and use a Gaussian process (GP) [6] to predict unknown dynamics away from the agent's current location. GPs are well-suited for modelling spatio-temporal distributions by incorporating online measurements into the posterior distribution, along with a measure of predictive uncertainty. GP prior hyperparameters can be estimated from physical intuition or from a small dataset of similar environments. We discuss this further in Section 3 of the appendix. Similarly to [2, 3, 4, 5], our GP maintains a *belief* over the underlying dynamics of the environment. Rather than ensuring safe environment exploration or maximising information collected, we extend these works by formulating a unified Bayes-optimal framework for efficient online mission planning.

We therefore pose our task as online, model-based Bayesian RL (BRL) with a GP belief over the transition function. As all environmental uncertainty is then encapsulated within the transition function, we assume that the cost function is known given an instance of the environment dynamics. Continuing the previous example, the robot does not need to learn online that it incurs more damage

6th Conference on Robot Learning (CoRL 2022), Auckland, New Zealand.

Appendices for:

Matthew Budd, Paul Duckworth, Nick Hawes, and Bruno Lacerda. Bayesian reinforcement learning for single-episode missions in partially unknown environments. In *Conference on Robot Learning (CoRL)*. PMLR, 2023. URL `https://proceedings.mlr.press/v205/budd23a/budd23a.pdf`

Appendix

1 List of acronyms and abbreviations

AUV	Autonomous underwater vehicle
BAMCP	Bayes-adaptive Monte-Carlo planning
BAMDP	Bayes-adaptive Markov decision process
BO	Bayesian optimisation
BRL	Bayesian reinforcement learning
GNC	Guidance, navigation, control
GP	Gaussian process
GPDM	Gaussian process dynamical model
MCTS	Monte-Carlo tree search
POMDP	Partially observable Markov decision process
ROS	Robot operating system
SSP	Stochastic shortest path
U-MDP	MDP with unknown feature values

2 Proof of Proposition 2

Proof. With individual belief updates at every stage, the history density is (shortening $\mathcal{P}_\pi^{h_t}(h_{t+\tau})$ to $\mathcal{P}_\pi^{h_t}$ and $\tilde{\mathcal{P}}_\pi^{h_t}(h_{t+\tau})$ to $\tilde{\mathcal{P}}_\pi^{h_t}$):

$$\begin{aligned}
\mathcal{P}_\pi^{h_t} &= p(a_t s_{t+1} a_{t+1} ... s_{t+\tau} \mid h_t, \pi) \\
&= p(a_t \mid h_t, \pi) p(s_{t+1} \mid h_t, \pi, a_t) p(a_{t+1} \mid h_{t+1}, \pi) ... p(s_{t+\tau} \mid h_{t+\tau-1}, a_{t+\tau}, \pi) && (5) \\
&= \prod_{t \le t' < t+\tau} \pi(h_{t'}, a_{t'}) \cdot \prod_{t < t' \le t+\tau} p(s_{t'} \mid h_{t'-1}, a_{t'-1}) && (6) \\
&= \prod_{t \le t' < t+\tau} \pi(h_{t'}, a_{t'}) \cdot \prod_{t < t' \le t+\tau} \int_T T(s_{t'-1}, a_{t'-1}, s_{t'}) p(T \mid h_{t'-1}) dT, && (7)
\end{aligned}$$

where $s_t = (s_k, s_e)$.

Given the definition of T^+ in (2), and the fact that a history h_t uniquely specifies a GP $\mathcal{GP}_{\mathcal{D}_h}$ of observations up to time t:

$$\mathcal{P}_\pi^{h_t} = \prod_{t \le t' < t+\tau} \pi(h_{t'}, a_{t'}) \cdot \prod_{t < t' \le t+\tau} \left[T^o\left(s_{t'-1}, a, s_{k,t'}\right) p^{\mathcal{GP}}\left(s_{e,t'} \mid s_{k,t'}, \mathcal{D}_{t'-1}\right) \right]. \quad (8)$$

The GP posterior $p^{\mathcal{GP}}\left(s_{e,t'} \mid s_{k,t'}, \mathcal{D}_{t'-1}\right)$ is a multivariate normal distribution (MVN). A GP belief update with a noise-free sampled observation is performed by conditioning the posterior MVN on the sampled value (for compactness we remove s_k from the MVN probability density function $p^{\mathcal{GP}}$):

$$\begin{aligned}
\mathcal{P}_\pi^{h_t} = \prod_{t \le t' < t+\tau} \pi(h_{t'}, a_{t'}) \cdot \prod_{t < t' \le t+\tau} T^o\left(s_{t'-1}, a, s_{k,t'}\right) \cdot \\
\left[p^{\mathcal{GP}}\left(s_{e,t+1} \mid \mathcal{D}_t\right) \cdot \prod_{\substack{t+1 < t' \\ \le t+\tau}} p\left(s_{e,t'} \mid s_{e,t'-1}, \ldots, s_{e,t+1}\right) \right]. && (9)
\end{aligned}$$

The repeated belief update product in the square brackets in (9) can be recognised as being equivalent (via the chain rule for probability) as being equivalent to the joint distribution across all values of $s_{e,t'}$:

$$\left[\cdots \right] = p^{\mathcal{GP}}\left(s_{e,t+1}, ..., s_{e,t+\tau} \mid \mathcal{D}_t\right). \quad (10)$$

Therefore the rollout distribution is identical between individual belief updates and root sampling:

$$\mathcal{P}_{\pi}^{h_t} = \prod_{t \leq t' < t+\tau} \pi(h_{t'}, a_{t'}) \cdot \prod_{t < t' \leq t+\tau} T^o\big(s_{t'-1}, a, s_{k,t'}\big) \cdot p^{\mathcal{GP}}\big(s_{e,t+1}, ..., s_{e,t+\tau} \mid \mathcal{D}_t\big) = \tilde{\mathcal{P}}_{\pi}^{h_t}. \tag{11}$$

Given the rollout distribution equivalence, search tree node statistics for both methods will converge to the same values in expectation. □

3 Gaussian Process Kernel Function and Hyperparameter Priors

The choices of kernel function and its hyperparameters encode assumptions about the smoothness, periodicity, and variability of the target function. Priors on hyperparameters are particularly important in the low-data regime, as they directly influence the GP's ability to generalise from limited observations ([6], Ch. 5). Overly broad priors can allow hyperparameters to take on extreme or unrealistic values, leading to poor generalisation.

In many real-world settings, physical laws and geometric constraints provide useful guidance for designing priors. Physics-based or domain-specific knowledge can inform the choice of hyperparameters, such as the expected smoothness and magnitude of the function being modelled [29, 30, 31]. For example, in geostatistics, variance and lengthscale parameters are often assigned Gamma or related priors to enforce positivity and encode scale information [32]. When modelling radiation fields, user-provided bounds on the maximum expected radiation source intensity and the minimum possible distance between the robot and the radiation source can suggest a characteristic lengthscale and variance for the GP kernel. More directly, in robotics we often know the characteristics of the robot's sensors, such as noise variance and spatial resolution, which can also inform hyperparameter choices [33, 34].

When direct physical or system-based information is harder to come by, users or automated systems can attempt to estimate hyperparameters from small datasets of similar environments [35, 36]. At a minimum, empirical estimates of smoothness and variance from similar environments may provide a principled starting point for hyperparameter prior design.

4 Radiation Domain Details

4.1 Radiation simulation

Radiation is simulated using $1/r^2$ "solid angle" radiation physics. Radiation sources $\{(s_i, \mathbf{x}_i), ...\}_{i=0}^{n}$ have strength s_i and pose $\mathbf{x}_i$. Source strength is the exposure value at a distance of 1m. The radiation exposure $\lambda(\mathbf{x})$ at robot pose $\mathbf{x}$ from these radiation sources is then

$$\lambda(\mathbf{x}) = \sum_{i}^{n} \frac{s_i^{src}}{\|\mathbf{x} - \mathbf{x}_i^{src}\|^2}. \tag{12}$$

4.2 Domain Details

A problem instance consists of the following components: **a)** a randomly generated distribution of radiation sources in the environment (below), **b)** the start location in the grid map, sampled from a uniform distribution across the map, and **c)** 3 goal states, also uniformly sampled. Sampled problem instances are discarded when they result in trivial solutions (e.g. due to the start and goal locations being too close) or where one goal is significantly closer to the start location than the other sampled goals.

Random radiation fields are generated in both of the following ways:

1. Random point-source distribution: insert between 5 to 20 radiation sources (uniform random sampling), with randomly sampled z position values $z \in \{1.0, 1.5, 2.5\}$ and randomly sampled strengths $s \in \{1000, 2000, 5000, 10000\}$. x and y position values are sampled uniformly within the bounds of the map ± 2.0m.

2. Randomly generated Gaussian random field distribution: evenly cover the map with radiation sources at $z = 1.0$ and draw their log strengths from a Gaussian random field. The Gaussian random field is generated with a radial basis function kernel, using uniformly sampled lengthscale hyperparameter $l \in \{3.0, 5.0, 7.0\}$ and variance hyperparameter $\sigma \in \{60, 75, 90\}$.

4.3 Gazebo simulation

A visualisation of the reactor room world in Gazebo is shown in Figure 4. The simulated robot is a Clearpath Jackal[2], using standard Gazebo lidar and odometry sensor simulation and standard ROS components such as AMCL localisation[3].

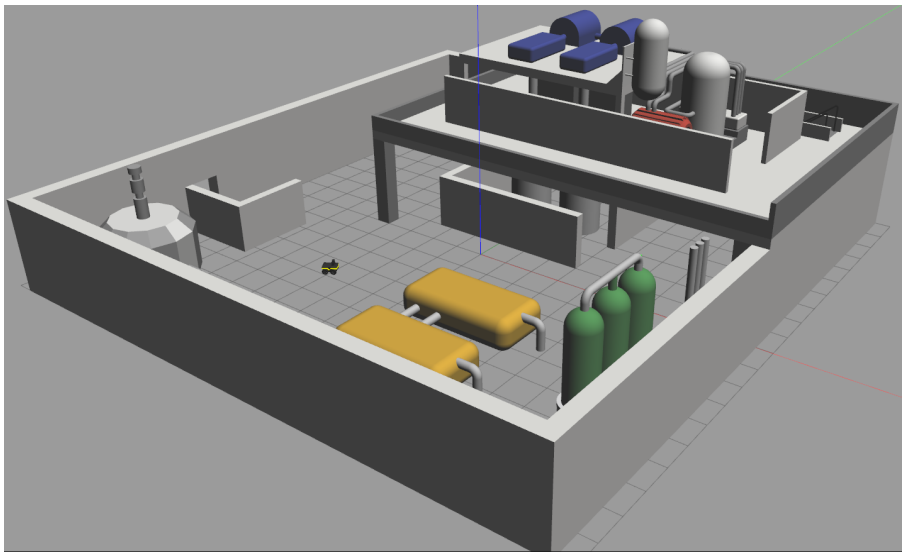


Figure 4: Visualisation of the reactor room Gazebo world in the middle of a simulated mission.

4.4 Algorithm parameters

The UCT exploration constant was dynamically set to equal the value of the decision node multiplied by 1.414. The guided rollout policy attempts to minimise the L2 distance from the goal state at the next state:

$$\pi_{rollout}(s_k) = \underset{a \in A(s_k)}{\arg\min}\, \text{dist}\,(s'_k, s_g)\, \forall s_g \in G, \tag{13}$$

where $A(s_k)$ are the enabled actions at a state and dist is a function that returns the shortest grid map path distance between two states in the grid map. The value of the ϵ parameter was 1.0 in the log GP space, meaning that MCTS search nodes covers increasingly large ranges of continuous value with increasing radiation level.

4.5 U-MDP Cost and Transition Structures

Transitions between grid map states are assumed to be deterministic as robot navigation does not fail in this easy-to-localise environment.

The state space $S^o = S_k \times S_e$ where $\{x, y\} \subseteq S_k$ and $rad\_exp \in S_e$. The set of actions $A^o = \{\text{left}, \text{left-up}, \text{left}, \text{up}, \text{right-up}, \text{right}, \text{right-down}, \text{down}, \text{left-down}\}$.

The cost structure is

$$C^o\big((s_k, s_e), a\big) = rad\_exp \cdot (\text{mcts-time} + d_a / v_{robot})\,, \tag{14}$$

[2] https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/

[3] http://wiki.ros.org/amcl

where mcts-time is the MCTS time budget allocated to the algorithm, d_a is the travel distance associated with the action (diagonal actions are 1.414 times further), and v_{robot} is the manually estimated average velocity of the simulated robot, taken to be $= 0.3\text{ms}^{-1}$.

4.6 GP model

The GP is a single-output GP that models $o : \mathbb{R}^2 \to \text{Dist}(\mathbb{R})$ where the GP input is $\{x, y\} \subseteq S_k$. The GP is trained on log radiation measurements and predicts log radiation levels.

The GP kernel is a combination of a bias kernel

$$k_{\text{bias}}(s_k, s'_k) = \sigma_b^2, \tag{15}$$

and a radial basis function kernel

$$k_{\text{rbf}}(s_k, s'_k) = \sigma^2 \exp\left(-\frac{\|s_k - s'_k\|^2}{2l^2}\right). \tag{16}$$

The radial basis function kernel hyperparameters are assigned the following uninformative Gamma distribution priors:

GP hyperparameter	Prior
Lengthscale l	$\Gamma(1, 0.5)$
Variance σ	$\Gamma(1, 4)$

This corresponds roughly to an expected lengthscale of 2m (standard deviation 2m) and expected log magnitude variance of 0.5 (standard deviation 0.5). The bias kernel variance was set to a constant $\sigma_{\text{bias}} = 1$, allowing a zero-mean GP prior to represent functions with a constant offset. At the beginning of each experiment the agent is provided with observations at the start state and two immediately neighbouring states as prior knowledge.

5 Underwater Currents Experiment

5.1 Domain Details

A problem instance consists of the following components: **a)** a 10km $\times$ 10km current field drawn from the real-world currents dataset in the same manner as [12], sampled from a fixed set of 12 fields where there is some variation in current across the field (rather than e.g. consistent current in one direction), **b)** the AUV start location, sampled from a uniform distribution across the field, and **c)** between 1 and 3 (with uniform probability) goal states, also uniformly sampled. Sampled problem instances are discarded when they result in trivial solutions (e.g. due to the start and goal locations being too close) or where one goal is significantly closer to the start location than the other sampled goals.

This experiment makes use of E.U. Copernicus Marine Service Information[4]. This is a dataset of north/east ocean current vectors on a 1500m spaced grid. To allow sampling of ground truth values at locations other than the grid locations, interpolation of the dataset using a spatio-temporal GP is carried out in the same manner as [12]. The currents experiment, the dataset used covers the region from approximately 47 to 62 latitude and -12 to 5 longitude. The dataset was originally collected on May 1 2020.

5.2 Kinematic GNC simulation

The vehicle's movement is determined by a kinematic calculation given of the vehicle's yaw, pitch and velocity control demands, process noise and the currents acting on the vehicle. Currents are drawn from the currents dataset at the vehicle's simulated true position. Underwater localisation is via an underwater acoustic beacon-aided extended Kalman filter. The vehicle uses this acoustic time-of-flight position feedback to navigate to the target location of the action selected by the MCTS planner. An example detailed run through the kinematic GNC simulator is shown in Figure 5.

[4] Available: https://resources.marine.copernicus.eu/?option=com_csw&view=details&product_id=NORTHWESTSHELF_ANALYSIS_FORECAST_PHY_004_013. . Accessed 2021-08.

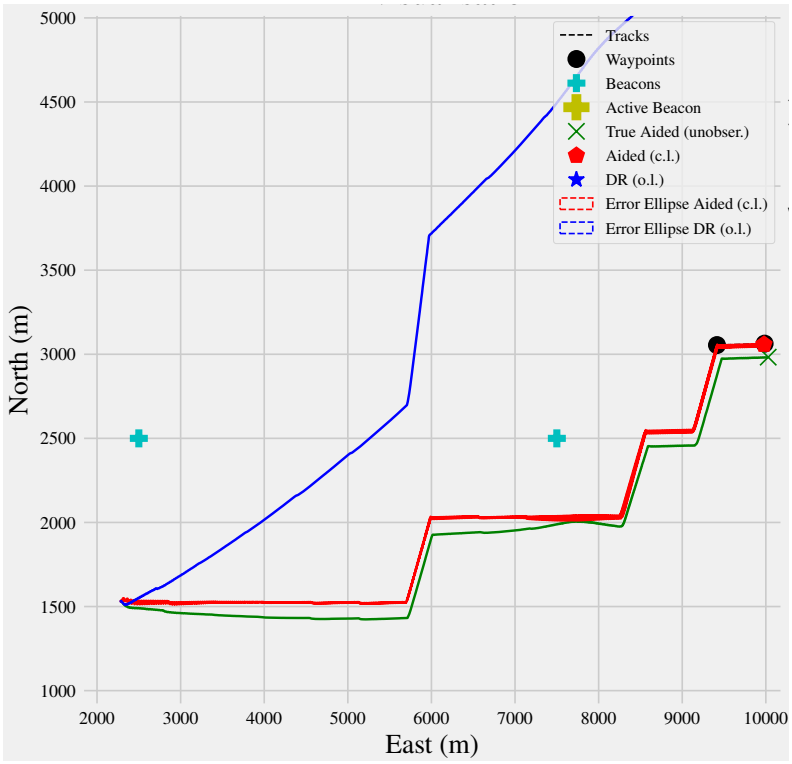


Figure 5: Example run through kinematics / GNC simulator. The green line is the vehicle's true path, the red line is the demanded path the vehicle has attempted to follow (where the demanded path is a result of an MCTS action selection at each grid MDP state), and the blue line is the "dead-reckoned/odometry-only" position estimate for the AUV. This is where the vehicle would localise to without any external EKF acoustic time-of-flight feedback.

5.3 Algorithm parameters

The UCT exploration constant was dynamically set to equal the value of the decision node multiplied by 1.414. The guided rollout policy attempts to minimise the L2 distance from the goal state at the next state:

$$\pi_{rollout}(s_k) = \underset{a \in A(s_k)}{\arg\min} \|s'_k - s_g\|_2 \forall s_g \in G, \tag{17}$$

where $A(s_k)$ are the enabled actions at a state. The value of the ϵ parameter was 0.1.

5.4 U-MDP Cost and Transition Structures

Transitions between grid map states are assumed to be deterministic as the AUV will always eventually reach a specified goal. The state space $S^o = S_k \times S_e$ where $\{x, y\} \subseteq S_k$ and $\{v_x, v_y\} \in S_e$. The set of actions $A^o = \{0°, 60°, 120°, 180°, 240°, 300°, \}$ corresponds to the direction the vehicle wishes to travel to the hex grid state in that direction. The vehicle travels against the current such that its net movement is in the action-specified direction.

The cost is the time taken to carry out the transition, given a constant vehicle speed $v = 0.6\text{m s}^{-1}$, the currents acting on the vehicle, and the requirement from the navigation controller that the net direction of travel is in the action-specified direction. Given the chosen action this is solved to find the net velocity v_a^{eff} and the direction the vehicle must steer against the current at the state.

$$C^o\big((s_k, s_e), a\big) = \frac{d_s}{v_a^{eff}}, \tag{18}$$

where d_s is the distance between states in the hexagonal grid.

5.5 GP model

The GP is a vector-output coregionalised GP that models $o : \mathbb{R}^2 \to \text{Dist}(\mathbb{R}^2)$ where the GP input is $\{x, y\} \subseteq S_k$.

The GP kernel is the sum of a bias kernel and a radial basis function kernel. The RBF hyperparameters are assigned the following relatively broad Gamma distribution priors based on sensible current values seen in the whole currents dataset, and distribution of optimised lengthscale parameters seen when GPs are trained on a small number of random subsets of the dataset:

GP hyperparameter	Prior
Lengthscale l	$\Gamma(a = 49.0, b = 0.014)$
Variance σ	$\Gamma(a = 1.0, b = 4.0)$

This corresponds roughly to an expected lengthscale of 3500m (standard deviation 500m) and expected current magnitude variance of 0.25 (standard deviation 0.0625). At the beginning of each experiment the agent is provided with observations at the start state and two immediately neighbouring states as prior knowledge.

C Appendices of Chapter 6

The Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI-24)

Stop! Planner Time: Metareasoning for Probabilistic Planning Using Learned Performance Profiles

Matthew Budd, Bruno Lacerda, Nick Hawes

Oxford Robotics Institute, University of Oxford
{mbudd, bruno, nickh}@robots.ox.ac.uk

Abstract

The metareasoning framework aims to enable autonomous agents to factor in planning costs when making decisions. In this work, we develop the first non-myopic metareasoning algorithm for planning with Markov decision processes. Our method learns the behaviour of anytime probabilistic planning algorithms from performance data. Specifically, we propose a novel model for metareasoning, based on contextual performance profiles that predict the value of the planner's current solution given the time spent planning, the state of the planning algorithm's internal parameters, and the difficulty of the planning problem being solved. This model removes the need to assume that the current solution quality is always known, broadening the class of metareasoning problems that can be addressed. We then employ deep reinforcement learning to learn a policy that decides, at each timestep, whether to continue planning or start executing the current plan, and how to set hyperparameters of the planner to enhance its performance. We demonstrate our algorithm's ability to perform effective metareasoning in two domains.

Introduction

Rational agents should reason about the consequences of their actions. However, this reasoning process itself has its own consequences: thinking is a physical process that requires time and energy. In many systems this issue can be ignored, in particular where the costs of any computation are negligible compared to the costs of actions. In other systems, the real-world effects of reasoning cannot be discounted. As an example, for mobile robots thinking and acting are often closely coupled. An autonomous vehicle can use its finite battery capacity on its on-board computer or its motors. To what extent is the energy saved by executing a better motion plan offset by the energy cost of additional computation? This scenario is illustrated in Figure 1. Of course, this assumes that the agent is reasoning *online*. If it is able to pre-plan all of its tasks offline, planning effort is negligible at runtime and no metareasoning tradeoff exists.

Over the last 40 years, the field of rational metareasoning has developed bounded optimal algorithms for reasoning about these tradeoffs (Russell and Wefald 1991; Cox and Raja 2011; Hay et al. 2012). Recent problem settings have included motion planning and heuristic search (Burns, Ruml, and Do 2013; Sung, Kaelbling, and Lozano-Pérez 2021; Bhatia et al. 2022). Optimal exact metareasoning is polynomially harder than reasoning (Lin et al. 2015), so all practical algorithms carry out approximate metareasoning. A common simplification is to only consider the immediate effects of a single reasoning step: this is known as the *meta-greedy* or *meta-myopic* assumption (Russell and Wefald 1989, 1991).

Figure 1: A deep sea treasure problem. Each policy represents the results of planning for a longer duration, and achieves reward indicated by the number of coins. π_{t_2} would be optimal overall if the extra reward gained by π_{t_3} is less than the cost (duration) of the additional planning time.

Under the umbrella of metareasoning problems, we focus specifically on metalevel control of anytime algorithms (Cox and Raja 2011). In this setting, reasoning is carried out by an *object-level* process, which is an anytime interruptible algorithm. This class of algorithms provides its current solution at any time during computation, but the longer the algorithm runs the better the solution will be. The object-level process is supervised by a *meta-level* process, which observes the object-level algorithm's state. Metalevel actions consist of allowing object-level computation to continue, or terminating the algorithm and acting using its current solution.

We wish to apply metareasoning to probabilistic planning with Markov decision processes (MDPs). Recent works have used deep reinforcement learning (RL) to learn non-myopic metalevel policies (Sung, Kaelbling, and Lozano-Pérez 2021; Bhatia et al. 2022). However, these works apply only to object-level algorithms where the current solution quality is known exactly, for example the current minimum path length in RRT*. Relaxing the solution quality observability

Appendices for:

Matthew Budd, Bruno Lacerda, and Nick Hawes. Stop! Planner Time: metareasoning for probabilistic planning using learned performance profiles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20053–20060, 2024

Supplementary Materials

Weighted BRTDP (WBRTDP)

Prelims. In heuristic search, a *heuristic function* $h(s)$ guides search by estimating the expected cost of the optimal policy π^* starting from state s. A lower-bound (optimistic) heuristic is admissible if it never overestimates the true cost to reach the goal. An upper-bound (pessimistic) heuristic is admissible if it never underestimates the true cost to reach the goal.

Algorithm overview. WBRTDP is a weighted hyperparameterised version of BRTDP (McMahan, Likhachev, and Gordon 2005). It maintains multiple lower-bound value functions, each of which use different heuristic value initialisations. We define state-action Q-values with respect to a value function v as

$$Q_v(s,a) = c(s,a) + \sum_{s'} T(s,a,s')v(s'). \qquad (1)$$

In WBRTDP, $\boldsymbol{v_l}(s)$ is a tuple of N_v lower-bound value functions $\left(v_l^i(s)\right)_{i=0}^{N_v}$, each initialised with a different lower-bound heuristic. Let κ be an index into $\boldsymbol{v_l}(s)$, and let $\boldsymbol{h_l}(s)$ be a N_v-length tuple of lower-bound heuristic functions used to initialise $\boldsymbol{v_l}(s)$. Each lower-bound value function is Bellman updated in parallel during planning using the cost and transition function for the MDP. The action selected for each state during a trial is the action that maximises the lower-bound value function estimate at index κ. κ therefore acts as a hyperparameter which selects which lower-bound heuristic to use during planning, and may be altered during the planning process. The WBRTDP_PLAN method accepts a hyperparameter κ to use for that planning cycle.

$\boldsymbol{h_l}(s)$ is expected to be a tuple of the heuristic function with increasing weights applied, or different heuristic functions with increasing inadmissibility. As we use a tuple of fixed values in our paper, the tuple of heuristic initialisation values are effectively weighted versions of a single fixed-value heuristic function. Such a heuristic function was demonstrated in experiments in the original BRTDP paper (McMahan, Likhachev, and Gordon 2005), and is simpler than the heuristic generation algorithm also described in that paper. We also tested weighted versions of simple heuristic functions, e.g. Manhattan and Euclidean distance, but did not find they performed any better than the fixed-value heuristic in either domain.

Note that only the lower bound heuristics are weighted. The upper bound heuristic remains admissible, and as in BRTDP, the output policy is greedy with respect to the upper bound value function estimate. This means that the upper bound value function remains an upper bound on the true value function, and the weighted heuristics affect only the speed of convergence by biasing search actions during planning. With an admissible upper bound heuristic, the upper

Algorithm 2: WBRTDP

Input: Solution convergence tolerance α, trial termination constant τ, initial state s_0, lower bound heuristic function $\boldsymbol{h_l}(s)$, lower bound heuristic index κ, upper bound heuristic function $h_u(s)$, maximum number of state visits to run $N_{\text{visit}}^{\max}$

Output: Policy $\pi_{s_0}^{v_u}$, greedy with respect to the upper bound value function estimate v_u

procedure WBRTDP_INIT
  $v_u(s) \leftarrow h_u(s)\ \forall s \in S$ ▷ Or lazy initialise on visit
  $\boldsymbol{v_l}(s) \leftarrow \boldsymbol{h_l}(s)\ \forall s \in S$ ▷ Or lazy initialise on visit
  $N_{\text{visit}} \leftarrow 0$
end procedure

procedure WBRTDP_PLAN($\alpha, \tau, s_0, \kappa, N_{\text{visit}}^{\max}$)
  while $(v_u(s_0) - v_l^\kappa(s_o)) > \alpha$ **and** $N_{\text{visit}} < N_{\text{visit}}^{\max}$ **do**
    WBRTDP_TRIAL(s_0)
  end while
  return $\pi_{s_0}^{v_u}$ ▷ Calculate greedy policy wrt v_u
end procedure

procedure WBRTDP_TRIAL(s_0)
  visited $\leftarrow$ (empty stack)
  $s \leftarrow s_0$
  while true **do**
    visited.push(s)
    $v_u(s) \leftarrow \min_a Q_{v_u}(s,a)$
    $a, \boldsymbol{v_l}(s) \leftarrow$ BELLMANTUPLEUPDATE(s)
    $\forall s', b(s') \leftarrow T(s,a,s')(v_u(s') - v_l^\kappa(s'))$
    $B \leftarrow \sum_{s'} b(s')$
    if $B < ((v_u(s_0) - v_l^\kappa(s_0))/\tau)$ **then break**
    end if
    $s \leftarrow$ sample from distribution $b(y)/B$
  end while
  $N_{\text{visit}} \leftarrow N_{\text{visit}} +$ visited.size
  while visited.size > 0 **do**
    $s \leftarrow$ visited.pop()
    $v_u(s) \leftarrow \min_a Q_{v_u}(s,a)$
    $\_, \boldsymbol{v_l}(s) \leftarrow$ BELLMANTUPLEUPDATE(s)
  end while
end procedure

procedure BELLMANTUPLEUPDATE(s)
  $\boldsymbol{Q}_{\text{best}} \leftarrow \left(\min_a Q_{v_l^i}(s,a)\right)_{i=0}^{N_v}$
  $a \leftarrow \arg\max_a Q_{v_l^\kappa}(s,a)$
  return $a, \boldsymbol{Q}_{\text{best}}$
end procedure

bound performance guarantee of BRTDP will therefore be maintained.

If the algorithm is queried for a solution in the middle of running a trial, it returns the current best solution found so far in previous trials.

Algorithm description. The structure of WBRTDP (Algorithm 2) is similar to BRTDP, with the addition of a hyperparameter κ to select the lower bound heuristic to use for the current planning cycle. Some components are separated into subroutines for clarity. The WBRTDP_INIT routine is called at the start of planning, to initialise the value functions and the state visit counter. In practice, the heuristic values are initialised lazily on first visit to a state, rather than all at once at the start of planning.

The WBRTDP_PLAN routine is called to run the planning algorithm. This is expected to be called multiple times by the metalevel agent, each time specifying the cumulative number of state visits to run (line 6). The WBRTDP_PLAN routine carries out trials starting from the root node s_0 (line 8) until the termination condition is met. The trials loop terminates when the difference between the upper and lower bound value functions at the root node is less than a threshold α, or the number of state visits exceeds a maximum $N_{\text{visit}}^{\max}$ (line 7). When the loop terminates, the greedy policy with respect to the upper bound value function is returned (line 10).

Note that the termination condition depends on the current lower bound heuristic index κ. If the solution has converged within α for one lower bound heuristic, it may not have converged for another lower bound heuristic. Planning can be continued using a different lower bound heuristic by calling the plan routine again with a different κ.

The WBRTDP_TRIAL routine carries out a single trial, starting from the root node s_0. The structure is much the same as in BRTDP, with three differences. Firstly, Bellman updates for the lower bound value function (lines 18 and 29) are separated into a subroutine BELLMANTUPLEUPDATE. Secondly, the weighted sampling (line 19) and trial termination steps (line 22) are dependent on the lower bound heuristic index κ. Thirdly, we track the number of trial state visits in line 25.

The BELLMANTUPLEUPDATE subroutine is called to update the lower bound value function at a state s and to return the action greedy with respect to the κth lower bound value function. As the transition and cost function components are the same for each Q-value update in the tuple (line 33), we can compute the Q-value updates for all lower bound value functions in a single pass over actions and outcome combinations. This leads to a negligible computational overhead compared to updating a single lower bound value function.

The overall computational overhead of WBRTDP compared to BRTDP is negligible as long as evaluating the tuple of lower-bound heuristic functions is not signficantly more expensive than evaluating a single heuristic function. This applies when using weighted multiples of a heuristic function.

Algorithm features. Algorithm features for WBRTDP are the value of the upper and lower bound value functions at the root node s_0, the number of trials completed, the total number of state visits during all trials, and the number of state visits during the last trial.

Hyperparameters. For the experiments we provide uniform uninformative (fixed-value) heuristics for the upper and lower bound value functions. The lower bound heuristic function therefore provides fixed values rather than a weighted version of a heuristic function: $\boldsymbol{h_l}(s) = (0.0, 10.0, 20.0, 30.0)\ \forall s$. This results in a metalevel action space size of 5 (4 hyperparameter options and 1 EXEC action). The heuristic initialisation for the upper bound function is fixed at $h_u(s) = 100.0\ \forall s$, which is admissible for all problems.

Deep Q Network

The DQN (Mnih et al. 2015) algorithm implementation is from Stable Baselines 3, version 1.8.0 (Raffin et al. 2021). Training is vectorised with 10 parallel environments. Default parameters are used, other than setting the number of gradient steps to match the number of transitions collected across all parallel environment instances. As a basic hyperparameter evaluation, experiments were also run using the default parameters of the DQN implementation from the work of Bhatia et al. (2022). This yielded slightly decreased performance so these hyperparameters were not used for further experiments.

As is common with deep RL, we normalise algorithm features and the context vector to the range 0 to 1. Value functions are normalised using the maximum possible cost-to-goal from s_0 for the problem distribution $p(\mathcal{M})$. The number of trial state visits feature is normalised by the number of trial state visits corresponding to the maximum number of timesteps per episode.

Alternative solution quality etimation methods

The *PolicyEval* and *MidBound* methods estimate the value of the current solution and use this estimate in the reward function defined by (Bhatia et al. 2022). This consists of rewarding the RL algorithm with the improvement in solution cost minus the cost of thinking. The computational overhead of PolicyEval is evaluated by recording the runtimes of planning and policy evaluation for each metalevel timestep. Using the ratio of these times, policy evaluation cost can be expressed in cost-of-thinking units. The total deliberation cost is the combination of planning cost and policy evaluation cost. The *PolicyEval* RL agent is provided with this total cost during training and evaluation. The same method is used to adjust the incurred cost of thinking during evaluation on the held-out problem set.

Experiments/Reproducibility

The held-out problem set used to evaluate algorithms' performance is reproducibly seeded. Each held-out environment is seeded with the same value across all algorithms. This ensures identical object-level policy execution and planner dynamics across each evaluation.

Experiment-running machines were Amazon Web Services *G4dn.4xlarge* instances with 16 vCPUs, 64 GiB RAM, and 1x 16GiB NVIDIA T4 GPU. The operating system was Ubuntu 20.04. The object-level MDP simulation and WBRTDP were implemented in C++.

Deep Sea Treasure domain

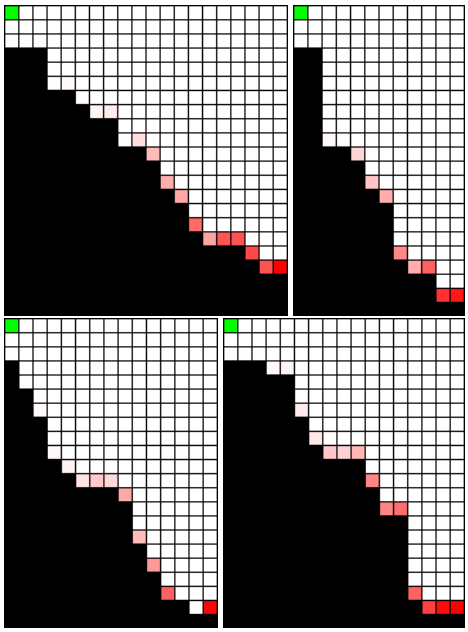


Figure 1: Four randomly generated deep sea treasure problems. The agent start location is shown in green, and increasingly dark red colours show increasingly valuable treasures.

This section describes the generation of problems $\mathcal{M} \sim p(\mathcal{M})$ for the deep sea treasure domain.

Figure 1 shows four randomly generated deep sea treasure layouts. Grid dimensions are sampled with $\text{dim}_x \sim \text{Uniform}\{10, 20\}$ and $\text{dim}_y \sim \text{Uniform}\{18, 25\}$. The ocean floor is generated by sampling a depth $\sim \text{Uniform}\{3, \text{dim}_y\}$ for each for each x up to dim_x, and then sorting these points in ascending order to give monotonically increasing depth with x. A treasure is placed at each point just above the ocean floor, with a probability $P_{\text{treasure}} = 0.9$ for each point. Treasure values are sampled using a Poisson distribution with depth-dependent mean. The mean value for a depth is proportional to a polynomial $(\text{depth})^2$. The maximum treasure value that can be sampled is 99. Problem instances are generated by combining a deep sea treasure layout with a uniformly sampled cost of thinking $\lambda \sim \text{Uniform}(0.0, 10.0)$, a per-axis speed limit $v_{\text{max}} \sim \text{Uniform}\{1, 2\}$, and an action failure probability $P_{\text{fail}} \sim \text{Uniform}(0, 0.3)$.

The initial state is always the top left corner of the grid. The default policy action is always to descend towards the ocean floor, and to proceed to the right if the agent is at the ocean floor but no treasure is present at that location.

Race track domain

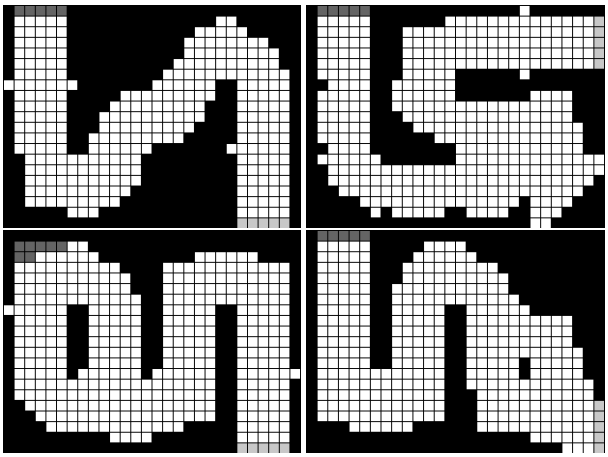


Figure 2: Four randomly generated race track problems. The start line is shown in dark grey, and the finish line is shown in light grey.

This section describes the generation of problems $\mathcal{M} \sim p(\mathcal{M})$ for the race track domain.

Figure 2 shows four randomly generated race track layouts. Race track layouts are generated by sampling 2 random paths in a 4×3 grid of nodes, each with a minimum path length of 5 and a maximum path length of 9. These paths are combined to form a high-level layout which may include forks and loops. The high-level layout is upscaled into a 28×21 cell race track using a set of 7×7 cell templates which map to possible combinations of paths leading into/out of a node. If the resulting race track is too short (minimum path length from start to goal < 50), then the paths are resampled. A race track being too short could result from the combination of the two sampled node paths being too short, even if each individual path matches the path length requirements. The start line is placed at the start of the path in the top left corner, and the finish line is placed at the end of the path. Problems instances are generated by combining a race track layout with a uniformly sampled cost of thinking $\lambda \sim \text{Uniform}(0.0, 10.0)$, a per-axis speed limit $v_{\text{max}} \sim \text{Uniform}\{3, 4\}$, and an action failure probability $P_{\text{fail}} \sim \text{Uniform}(0, 0.3)$.

The centre point of the start line is chosen as the deterministic initial state for the agent. The default policy action is to proceed at a speed of 1 along the track towards the goal.

References

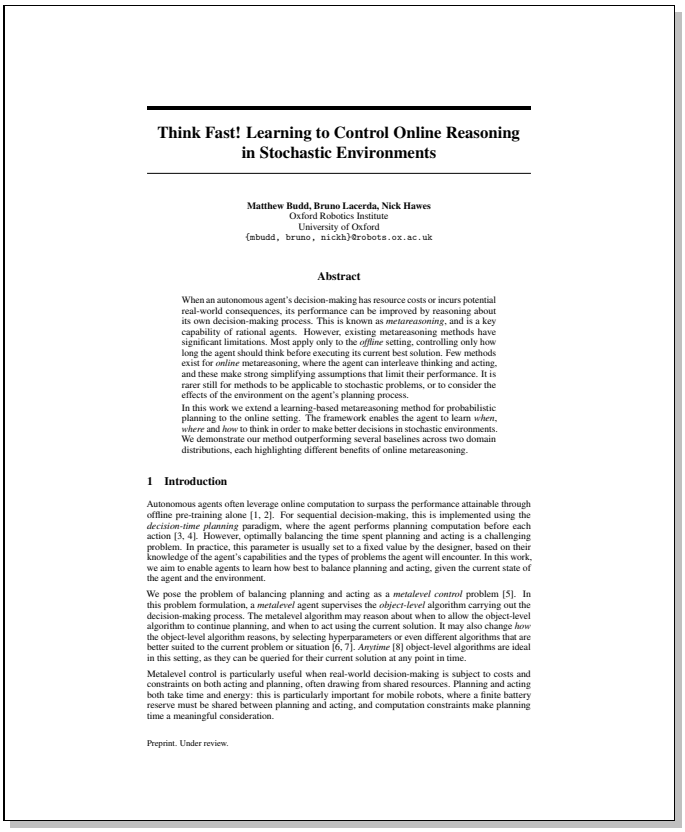
Bhatia, A.; Svegliato, J.; Nashed, S. B.; and Zilberstein, S. 2022. Tuning the hyperparameters of anytime planning: A metareasoning approach with deep reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, 556–564.

McMahan, H. B.; Likhachev, M.; and Gordon, G. J. 2005. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *Proceedings of the 22nd International Conference on Machine learning*, 569–576.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533.

Raffin, A.; Hill, A.; Gleave, A.; Kanervisto, A.; Ernestus, M.; and Dormann, N. 2021. Stable-baselines3: Reliable reinforcement learning implementations. *The Journal of Machine Learning Research*, 22(1): 12348–12355.

D Appendices of Chapter 7



Think Fast! Learning to Control Online Reasoning in Stochastic Environments

Matthew Budd, Bruno Lacerda, Nick Hawes
Oxford Robotics Institute
University of Oxford
{mbudd, bruno, nickh}@robots.ox.ac.uk

Abstract

When an autonomous agent's decision-making has resource costs or incurs potential real-world consequences, its performance can be improved by reasoning about its own decision-making process. This is known as *metareasoning*, and is a key capability of rational agents. However, existing metareasoning methods have significant limitations. Most apply only to the *offline* setting, controlling only how long the agent should think before executing its current best solution. Few methods exist for *online* metareasoning, where the agent can interleave thinking and acting, and these make strong simplifying assumptions that limit their performance. It is rarer still for methods to be applicable to stochastic problems, or to consider the effects of the environment on the agent's planning process.

In this work we extend a learning-based metareasoning method for probabilistic planning to the online setting. The framework enables the agent to learn *when*, *where* and *how* to think in order to make better decisions in stochastic environments. We demonstrate our method outperforming several baselines across two domain distributions, each highlighting different benefits of online metareasoning.

1 Introduction

Autonomous agents often leverage online computation to surpass the performance attainable through offline pre-training alone [1, 2]. For sequential decision-making, this is implemented using the *decision-time planning* paradigm, where the agent performs planning computation before each action [3, 4]. However, optimally balancing the time spent planning and acting is a challenging problem. In practice, this parameter is usually set to a fixed value by the designer, based on their knowledge of the agent's capabilities and the types of problems the agent will encounter. In this work, we aim to enable agents to learn how best to balance planning and acting, given the current state of the agent and the environment.

We pose the problem of balancing planning and acting as a *metalevel control* problem [5]. In this problem formulation, a *metalevel* agent supervises the *object-level* algorithm carrying out the decision-making process. The metalevel algorithm may reason about when to allow the object-level algorithm to continue planning, and when to act using the current solution. It may also change *how* the object-level algorithm reasons, by selecting hyperparameters or even different algorithms that are better suited to the current problem or situation [6, 7]. *Anytime* [8] object-level algorithms are ideal in this setting, as they can be queried for their current solution at any point in time.

Metalevel control is particularly useful when real-world decision-making is subject to costs and constraints on both acting and planning, often drawing from shared resources. Planning and acting both take time and energy: this is particularly important for mobile robots, where a finite battery reserve must be shared between planning and acting, and computation constraints make planning time a meaningful consideration.

Preprint. Under review.

Appendices for:

Matthew Budd, Bruno Lacerda, and Nick Hawes. Think Fast! Learning to control online reasoning in stochastic environments. *Under review*, 2025a

Supplementary Material

A Experiment Design

A.1 Domains

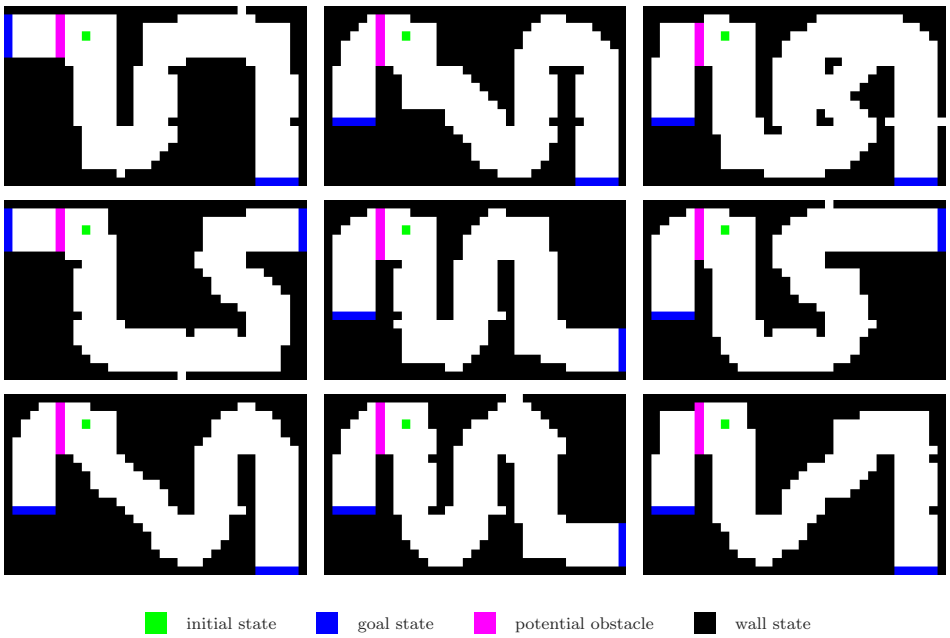


Figure 4: Example ObstacleRacetrack environments from the evaluation set.

ObstacleRacetrack. ObstacleRacetrack MDPs are generated by sampling a graph from a high level graph structure in a grid of 5 x 3 nodes. The close goal line must be within 2 cells of the initial position, and the far goal line must 10 cells from the initial position. Loops are possible in the high-level graph. The high-level graph is upscaled to a 2D grid of cells, with each graph node being represented by a 7×7 grid of cells. 7×7 cell grids are randomly generated to match the connectivity of the corresponding graph node. Obstacle states (which act as wall cells or empty cells depending on whether the obstacle is present) are placed to block the path between the initial position and close goal. This forms the x and y state factor space of the sampled MDP. The maximum velocity is 3 in any direction, so $v_x \in \{-3, -2, \ldots, 2, 3\}$ and $v_y \in \{-3, -2, \ldots, 2, 3\}$. Colliding with a wall or obstacle sets x and y velocities to 0, and the agent transitions to the closest cell to the wall or obstacle in the direction of its velocity vector before the collision. The action failure probability for the MDP is sampled $p_{\text{fail}} \sim \mathcal{U}(0.0, 0.2)$. The probability of the obstacle being present is sampled $p_{\text{obstacle}} \sim \mathcal{U}(0.5, 0.8)$. Both of these probabilities are provided as context features to the metalevel controller. As the obstacle state factor is $\in -1, 0, 1$, the maximum number of states in ObstacleRacetrack MDPs is 108045 ($7[v_x] \times 7[v_x] \times 3[\text{obstacle}] \times 35[x] \times 21[y]$). As wall states are unreachable, the number of reachable states is smaller. 9 examples of ObstacleRacetrack environments from the evaluation set are shown in Figure 4.

In ObstacleRacetrack, the duration τ of one PLAN action in the object-level algorithm is 12500 Bellman updates. The maximum budget of Bellman updates is 400K, so the maximum number of PLAN actions the agent can take in one episode is 32.

For *OffLearn*, the default object-level policy in this domain is to accelerate to a velocity of 1 along the centre of the track to the far away goal. Introducing obstacle-adaptive behaviour into the default policy would make the default policy overly well-performing compared to other metalevel algorithms without access to a default policy.

For this domain, the object-level algorithm is provided with uninformative initial value function values, set to 0 for all states for the lower bound value function, and 100 for the upper bound value function.

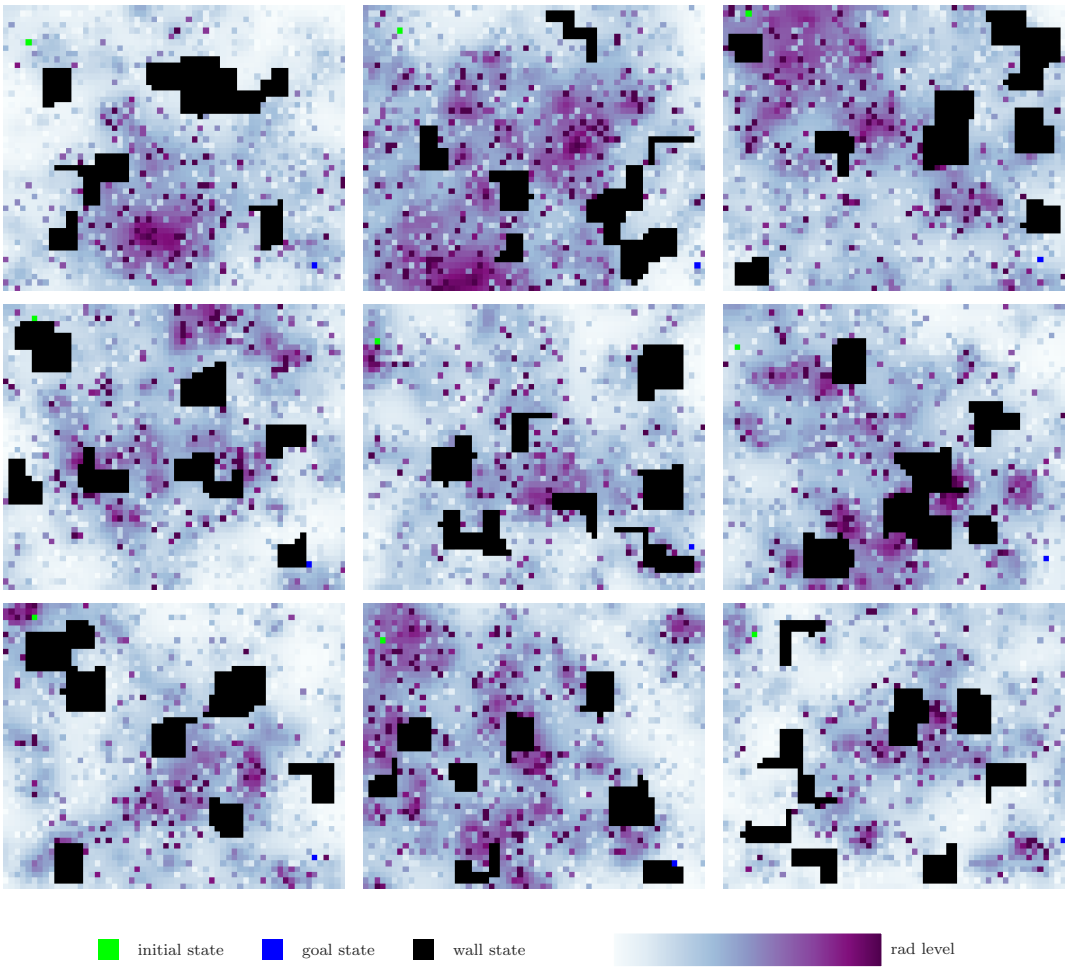


Figure 5: Example RadWorld environments from the evaluation set.

RadWorld. RadWorld problems are generated with a multi-step process. First, physical obstacles are placed in a 60 $\times$ 50 grid. The maximum number of obstacles is 10, and the maximum total number of obstacle cells is 15% of the grid. Obstacles are variations of rectangles and L-shapes with randomly sampled orientations, side lengths and thicknesses. The obstacle grid undergoes a binary dilation with a 3$\times$3 kernel to blend nearby obstacles together, followed by applying random binary noise. This is finally followed by a binary erosion with a 3$\times$3 kernel to remove small holes and small obstacles. The initial and goal positions are randomly sampled from a truncated Gaussian distribution with mean $(5, 5)$ and $(55, 45)$ respectively, and a standard deviation of 2. Problem instances are discarded where obstacle placement makes the goal unreachable.

The radiation level distribution in the environment is generated by simulation of radiation sources with $\frac{1}{r^2}$ radiation decay with distance. The joint distribution of radiation source numbers, strengths and locations is designed to generate few large radiation sources and increasingly large numbers of small radiation sources. Noise is added to the radiation level distribution to represent random variation due to environment construction and shadowing effects. For 30% of the cells in the environment, the radiation level is multiplied by $0.5 + 0.5 \cdot \text{Gamma}(1.2, 1.2)$. In order not to unfairly bias results against the offline metalevel controller and to reduce variance between evaluation runs, the radiation level at the initial state is set to the median radiation level across the environment. 9 examples of RadWorld environments from the evaluation set are shown in Figure 5.

In RadWorld, the duration τ of one PLAN action in the object-level algorithm is 5000 Bellman updates. The maximum budget of Bellman updates is 320K, so the maximum number of PLAN actions the agent can take in one episode is 64.

For *OffLearn*, the default object-level policy in this domain was to to take take actions that led to states with the lowest 8-connected total distance to goal (taking into account obstacles), with ties

broken via the smallest Euclidean distance to goal. This results in a default policy that takes the shortest available diagonal path to the goal while avoiding obstacles.

For this domain, the object-level algorithm was provided with initial value function values based on a scaled 8-connected distance to goal (taking into account obstacles). This distance measure was multiplied by the maximum possible radiation level for the upper bound value function and by the minimum possible radiation level for the lower bound value function, to give admissible initial value function values. We also tested uninformative initial value function values as in the previous domain, but this leads to high variance in how long BRTDP takes to find a goal state to terminate the first trial. This can lead to metalevel controller performance being heavily biased by how quickly the object-level algorithm finds a goal state in the first trial, which is not representative of the performance of the metalevel controller.

A.2 Implementation

The metalevel MDP is implemented with a Gymnasium interface [26]. *Bounds* and *DTP* algorithms use a smaller action set consisting only of PLAN and ACT actions, rather than the full hyperparameter setting action space.

DTP aim to choose between PLAN/ACT actions to keep the ratio of planning to acting as close as possible to their target. They are limited by the same total planning time budget, meaning that higher planning ratios may exceed the budget during the episode and only ACT thereafter. The same total planning budget applied to *Bounds*, but this algorithm did not tend to use the full budget. Also, *DTP* agents must PLAN when they have no action specified by their object-level policy, so they may PLAN for more than one timestep in the initial state to generate an initial valid policy.

RL algorithm implementations were from Stable Baselines3 [27], version 2.3.2. DQN parameters are set to their default values, except for increasing the number of gradient steps per rollout to match the number of environment steps taken in that rollout. The size of the hidden layer was set to 64, and the number of hidden layers to 2.

We enforced a maximum planning time budget, both to ensure that training episodes terminate and because there is no benefit to planning longer than the convergence point of the object-level algorithm's solution. As the DQN implementation used does not support action masking, we force the ACT action to be selected if the maximum planning time budget is reached. We also implemented truncation at a maximum number of environment interactions, to stop episodes in the case where the agent ACTs on a poor policy and spreads its planning time budget over many object-level states, leading to it not generating a valid goal-reaching policy within the maximum planning time budget. We did not observe this occur in practice for the domains tested.

A.2.1 Object-level Planner: WBRTDP

WBRTDP was implemented in single-threaded C++. The WBRTDP algorithm is described in Budd et al. [7]. (W)BRTDP trials have varying lengths when they terminate only on reaching a goal state. This means that the end of a metalevel timestep τ likely does not coincide with the end of a BRTDP trial. Although idealised as an anytime algorithm, the state of the object-level algorithm is not consistent in the middle of a trial if interrupted. Rather than implementing functionality to break trials and restore the state of the object-level algorithm as it was before the trial if the number of Bellman updates exceeds that allowed for the metalevel timestep, we instead use checkpointing of the object-level algorithm state to return a copy of the state at the start of the trial. When future planning actions are taken, the state at the end of the trial can be returned if it was reached within the new allowed number of Bellman updates.

We use similar algorithm features as in Budd et al. [7]. As we are using a deep learning-based agent, we can provide many algorithm features to the learning agent, and it should learn which are useful. However, we do not want to calculate complex observation features because this adds metareasoning overhead. The features provided are the upper and lower Q-value bounds at the current state, the total number of trials completed so far, the number of states visited during all trials, the number of states visited during the last trial, and the number of trials that have passed through the current state.

When updating the planner with the new state s', as described by the transition function $T^{\chi}_{\mathcal{M}}(\chi'|\chi, s')$, the planner is set to run from a state sampled from the transition function $T_{\mathcal{M}}(s', \text{NOP}, \cdot)$, *not* from

the current state. This is because after committing to planning, the earliest that plan can be executed is after the NOP action is completed, one object-level timestep later.

A.2.2 Evaluation

During evaluation, stochastic outcomes in the object-level MDP are seeded based on the evaluation MDP ID. The ground-truth values of epistemic state factors, such as the presence of the obstacle in ObstacleRacetrack, are sampled when the MDP is created. During evaluation, outcomes of stochastic actions which depend on epistemic state factors are rejection sampled to match the ground-truth epistemic state factor values. Evaluation is carried out with deterministic metalevel action selection for learning-based policies.

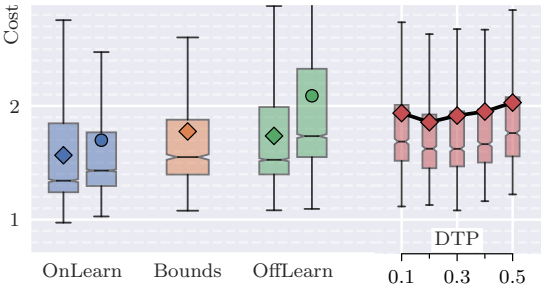
A.3 Compute Hardware

Training was carried out with 16 training environments trained in parallel on a machine with a AMD Ryzen Threadripper 7960X CPU (24 cores, base clock 4.2 GHz) and a NVIDIA GeForce RTX 4090 GPU. Training was CPU-bound due to CPU-based object-level planning dominating the time taken to simulate episode steps. With this parallel training configuration, training one agent on the ObstacleRacetrack domain took approximately 12 hours, and training one agent on the RadWorld domain took approximately 3 hours.

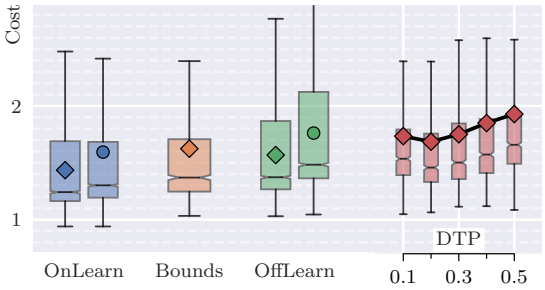
This compute hardware was also used for evaluation, and for prelimiary experiments (not reported in this work) to validate the system implementation and analyse performance.

B Additional Results

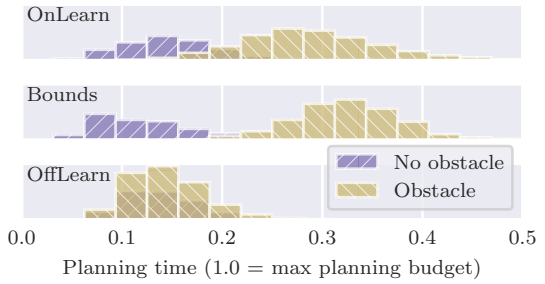
B.1 Varying Cost of Planning



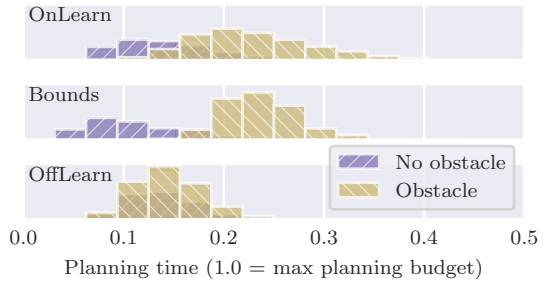
(a) Normalised cost-to-goal for the method and baselines, reproduced from Figure 2a in the main text.



(b) Normalised cost-to-goal for the method and baselines, when the cost of planning is halved.



(c) Planning time distribution for the method and baselines, reproduced from Figure 2b in the main text.

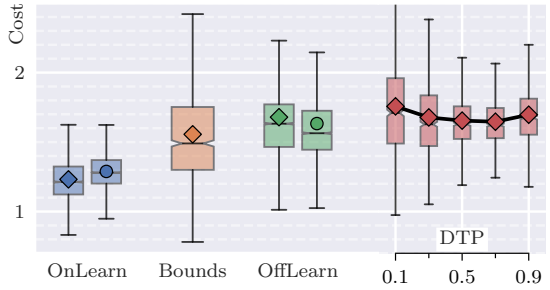


(d) Planning time distribution for the method and baselines, when the cost of planning is halved.

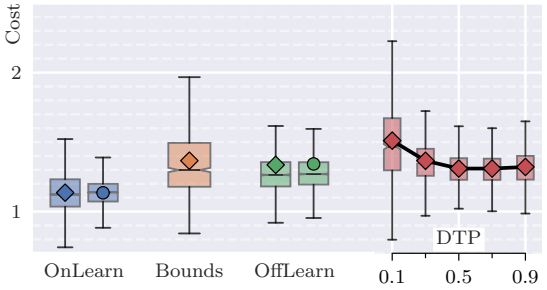
Figure 6: Cost performance and planning time distributions in the ObstacleRacetrack domain, when planning is made cheaper by a factor of 2.

We carried out an additional round of training and evaluation in the ObstacleRacetrack domain, doubling the value of τ to 25K Bellman updates and the maximum planning time budget to 800K Bellman updates. This is equivalent to halving the cost of planning in the environment. The results are shown on the right hand side of Figure 6, where the left hand side shows the results from the main

text for comparison. The same trends in cost performance are observed, with *OnLearn* significantly outperforming the baselines. General performance is improved across the board due to the decreased cost of planning, but the relative performance between methods is similar. The *DTP* methods still perform best with the plan/act ratio set to 0.2, but the difference between parameterisations is more pronounced at this lower cost of planning. In terms of planning time (Figures 6c and 6d), both online methods tended to use fewer planning steps for the blocked path case, and a similar number of planning steps for the no obstacle case.



(a) Normalised cost-to-goal for the method and baselines, reproduced from Figure 3a in the main text.



(b) Normalised cost-to-goal for the method and baselines, when the cost of planning is halved.

Figure 7: Cost performance and planning time distributions in the RadWorld domain, when planning is made cheaper by a factor of 2.

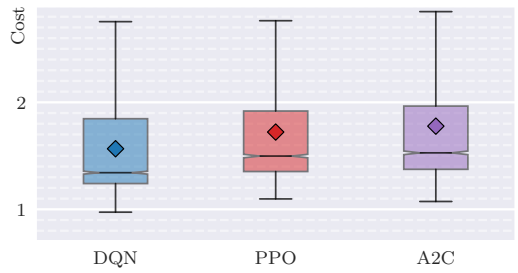
We carried out an additional round of training and evaluation in the RadWorld domain, doubling the value of τ to 10K Bellman updates. The maximum planning time budget was left unchanged, so the maximum number of PLAN actions the agent can take in one episode is 32. This makes planning faster, scaling radiation cost per Bellman update down by a factor of 2. The results are shown on the right hand side of Figure 7, where the left hand side shows the results from the main text for comparison. Compared to the full planning cost case, the mean excess cost decreases by 41% for *OnLearn*, 50% for *OffLearn*, and 40% for *Bounds*. In this setting, OfflineAlg outperforms *Bounds* significantly. The best parameterisation of *DTP* is still around 0.5 compared to 0.7 previously, but there is no significant performance difference between the two parameterisations. The performance difference between parameterisations is more pronounced at this lower cost of planning. *DTP* with a plan/act ratio of 0.5 slightly outperforms *Bounds*, but the difference is not statistically significant.

B.2 Other Learning Algorithms

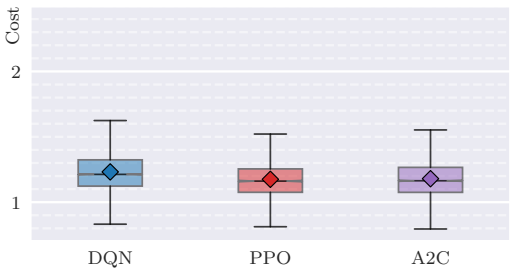
Results shown in previous sections used DQN [23] as the learning algorithm for the online metareasoner *OnLearn*. In this section we evaluate two alternative learning algorithms: PPO [28] and A2C [29]. For the two policy gradient methods, we used the default hyperparameters from Stable Baselines3 [27], other than changes to better match the distributional nature of the problem setting. For PPO, the batch size and number of steps for each environment per update were set to 512. For A2C, the number of steps for each environment per update was set to 64. These values were identified as the best performing values using a parameter sweep. Similarly to DQN, we trained with 16 environments in parallel and used frame stacking with a stack size of 3. We also tested PPO with a recurrent policy, but found this did not outperform frame stacking.

Figure 8 compares the performance of DQN, PPO, and A2C for the online metareasoner in both the ObstacleRacetrack and RadWorld domains. In ObstacleRacetrack (Figure 8a), DQN outperforms PPO and A2C by a small margin ($\sim$10%), but in RadWorld (Figure 8b) it incurs slightly higher cost. PPO and A2C perform similarly across both domains, with PPO slightly outperforming A2C in ObstacleRacetrack. In both domains, PPO and A2C perform statistically significantly better than the baselines, similarly to DQN.

The replay buffer and TD updates used by DQN allow it to learn more efficiently in our setting, where the object-level problem distribution induces a meta-RL problem, compared to the on-policy methods. The two on-policy methods required more than twice as much training to reach comparable performance, and were more sensitive to hyperparameter settings than DQN.



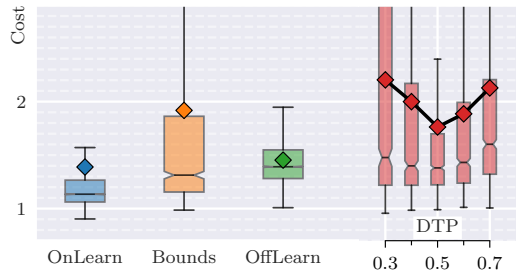
(a) Normalised cost-to-goal for the online metareasoner with different learning algorithms in the ObstacleRacetrack domain.



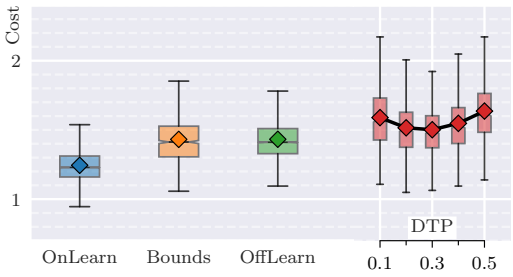
(b) Normalised cost-to-goal for the online metareasoner with different learning algorithms in the RadWorld domain.

Figure 8: Comparing the performance of using different deep reinforcement learning algorithms for the online metareasoner.

B.3 Domains from Budd et al. [7]



(a) Normalised cost-to-goal for the method and baselines in the DeepSeaTreasure domain [7].



(b) Normalised cost-to-goal for the method and baselines in the Racetrack domain [7].

Figure 9: Cost performance results for the domains from Budd et al. [7].

We evaluated the method and baselines on the two domains from Budd et al. [7], DeepSeaTreasure and Racetrack. The results are shown in Figure 9. Values reported for *OffLearn* differ from those in Budd et al. [7] because we fix the cost of planning for *OffLearn* to be the cost of the NOP action in the initial state, whereas in Budd et al. [7] the cost of planning was drawn from a distribution.

In the DeepSeaTreasure domain, we observe a large gap between mean and median cost performance for most algorithms. This arises from a long tail of particularly high-cost episodes, caused by a poor initial policy and the stochastic nature of the environment. *Bounds* performs especially poorly in terms of mean performance in this domain, as it frequently myopically decides not to improve its initial low-quality policy. The performance of *DTP* is highly sensitive to the plan/act ratio in this domain, resulting in widely varying mean cost performance.

B.4 Additional object-level state observations for RadWorld

We also ran RadWorld experiments with an agent-centric observation window around the current state, giving a field of view of 9x9 cells. This resulted in a 10% (statistically significant) decrease in performance, which did not decrease with additional training time.