Planning for robotic exploration based on forward simulation

Mikko Lauri Risto Ritala

Abstract

A robotic agent is tasked to explore an a priori unknown environment. The objective is to maximize the amount of information about the partially observable state. The problem is formulated as a partially observable Markov decision process (POMDP) with an information-theoretic objective function, further approximated to a form suitable for robotic exploration. An open-loop approximation is applied with receding horizon control to solve the problem. Algorithms based on evaluating the utilities of sequences of actions by forward simulation are presented for both finite and uncountable action spaces. The advantages of the receding horizon approach to myopic planning are demonstrated in simulated and real-world exploration experiments. The proposed method is applicable to a wide range of domains, both in dynamic and static environments, by modifying the underlying state transition and observation models.

1 Introduction

Autonomous robotic agents performing tasks such as monitoring, surveillance or exploration must be able to plan their future measurements and other information-gathering actions. In partially observable dynamic environments, planning requires reasoning over uncertain outcomes in the presence of sensor noise. The agent’s actions have effects that are not precisely known beforehand, and the sensory information the agent perceives is noisy and/or incomplete. Probabilistic models quantify the uncertainties in system dynamics and observations. Consequently, the agent’s knowledge of the state is represented by a belief state, which is a probability density function

*The authors are with the Department of Automation Science and Engineering, Tampere University of Technology, P.O. Box 692, FI-33101 Tampere, Finland. Email: firstname.lastname (at) tut.fi
The utility of actions is measured by an appropriate reward function, and the agent’s objective is to maximize the sum of expected rewards over a specified horizon of time. The resulting planning problems are instances of partially observable Markov decision processes (Aström, 1965; Kaelbling et al., 1998), or POMDPs, a type of sequential decision-making problem. The solution of a POMDP is a control policy, i.e. a mapping from belief states to actions.

Several authors have proposed applying information theoretic quantities such as entropy or mutual information as suitable reward functions for robotic information gathering problems, see e.g. Stachniss and Grisetti (2005); Hitz et al. (2014); Charrow et al. (2014). The result is a POMDP with a reward function nonlinear in the belief state. Many standard POMDP solution algorithms assume rewards to be linear in the belief state (see Hauskrecht (2000); Ross et al. (2008) for an overview), and thus cannot handle information theoretic reward functions.

POMDPs in general are known to be computationally hard (PSPACE-complete; Papadimitriou and Tsitsiklis, 1987). Nevertheless, sequential maximization of mutual information has efficient approximate solutions in some special cases. Submodularity of mutual information in a sensor placement problem has been exploited to derive a polynomial time approximation algorithm (Krause et al., 2008). In mixed-observability problems, a part of the state is fully observable and evolves deterministically. Atanasov et al. (2014) showed that if the observation model in a mixed observability target tracking problem is linear-Gaussian w.r.t. the target state, the optimal policy for maximizing mutual information is open-loop. In a mixed-observability case where possible sensing actions are constrained by the fully observable part of the state, Lauri and Ritala (2015) showed that under certain conditions the problem can be relaxed into a multi-armed bandit problem, leading to easily computable bounds for the value function.

In this article, we consider sequential maximization of mutual information for autonomous exploration and surveillance tasks in partially observable domains. We apply forward simulation methods for finding a solution to the open loop approximation of a POMDP. This approach allows general belief-dependent reward functions and avoids discretization of the continuous planning space. We derive an approximation for mutual information that can be applied in conjunction with forward simulation based planning. We provide an empirical evaluation of our proposed approach in simulated and real-world exploration experiments.

The rest of the article is organized as follows. Section 2 provides a survey of related work and specifies the novelty of our contribution in relation
to the state-of-the-art. In section 3 we formulate the POMDP problem we wish to solve. Suitable solution methods are discussed, and an open-loop approximation combined with an online receding horizon control scheme is proposed. Section 4 reviews two forward simulation based methods for finding the best open-loop control sequence for an exploration problem. Section 5 derives an approximation of mutual information suitable for robotic exploration problems. Section 6 describes the results of simulation experiments in a dynamic occupancy grid world. Section 7 presents the software architecture for implementation of our approach and reports the results of real-world exploration trials. Finally, section 8 provides concluding remarks.

2 Related work

Robots typically collect information on both their internal state and the state of the environment they are interacting with. In simultaneous localization and mapping (SLAM) (see Durrant-Whyte and Bailey (2006) for a review) the robot must jointly estimate both its pose (internal state) and the map of the environment based on its actions and observations. Robots' information-gathering actions consist of actions that affect their pose, and hence the area covered by their sensors, and actions explicitly selecting between sensors or their operating modes. In the active SLAM problem (Carlone et al, 2010), the robot's actions are selected to obtain best estimates on the pose and the map. Thus, the active SLAM problem is an exploration or sensor selection problem. The goal is to maximize information on both the robot pose and the map.

Juliá et al (2012) survey exploration methods for mobile robots where locations to explore are selected e.g. based on heuristic rules or decision theory. They classify the methods according to the levels of multi-robot coordination and integration with SLAM algorithms. They concluded that SLAM-integrated exploration performs best with regard to quality of map information. Their results also agreed with Amigoni (2008); Amigoni and Caglioti (2010), who found decision theoretic criteria combining both utility of exploration and its cost (e.g. time or distance) to be preferable if both extent of the explored area and exploration time were optimized. In the following, we review in more detail some single-robot exploration techniques that employ decision theoretic criteria to guide the exploration process.

Information on the location of the robot and environment features, or landmarks, may be modelled by a multivariate Gaussian distribution. The SLAM problem can then be solved for example by applying the Extended
Kalman Filter. Exploration with such feature-based maps was studied by Sim and Roy (2005) who describe an A-optimal exploration method, i.e. they minimize the trace of the state covariance matrix. They discretize the location of the robot to a grid and plan an informative trajectory in open loop as a sequence of discrete positions via a breadth-first search. A similar objective function was used by Huang et al (2005), adopting a model predictive control (MPC) approach for optimization over multiple time steps. Discretization of the action space was also applied by Kollar and Roy (2008), who applied reinforcement learning to learn parameterized robot trajectories for exploration.

Martinez-Cantin et al (2009) relaxed many of the assumptions made in earlier work, such as discretization of the action space and the myopic optimization horizon. They applied a Monte Carlo search algorithm in policy space and applied a Gaussian process approximation of the objective function. The policies to evaluate were selected by minimizing the average mean square error of the state estimate consisting of robot and landmark locations.

A belief space planning approach investigated by Indelman et al (2014, 2015) addressed many of the limitations of earlier studies. A planner architecture consisting of an estimation layer and a decision layer combined with a model predictive control strategy for non-myopic planning was applied, assuming a Gaussian belief over robot and landmark poses. Discretization was avoided by applying a gradient descent method for computing optimal actions. Possible future measurements were treated as random variables, relaxing the assumption of maximum likelihood measurements. Exploration was considered in the sense that the objective function included an A-optimality criterion for state covariance.

Another body of work in exploration employs metric map representations such as occupancy grids (Moravec 1988). Bourgault et al (2002) combined occupancy grid mapping with feature-based SLAM and used mutual information as the reward function. A discretized action space was applied with myopic optimization.

State-of-the-art SLAM for occupancy grid maps employs Rao-Blackwellized particle filtering (RBPF), see e.g. Grisetti et al (2007), where each particle represents a map and trajectory hypothesis. Stachniss and Grisetti (2005) studied myopic exploration in RBPF SLAM by discretizing the action space to a set of possible waypoints and then evaluating the approximate expected information gain when traveling to the waypoints by sampling. This work was later expanded upon (Blanco et al, 2008; Carbone et al, 2010) by considering alternative measures of uncertainty of the SLAM solution. These approaches consider both exploration of new areas and maintaining...
the consistency of the particle filter approximation.

In this work, we extend current literature in several directions. In contrast to Martinez-Cantin et al (2009); Indelman et al (2014, 2015) and others, we do not assume a Gaussian belief state, but instead apply more general models of the environment and robot states. We consider uncountable action spaces and belief dependent objective functions that may be non-differentiable, ruling out gradient methods for computing optimal actions. Instead, we propose applying forward-simulation based search methods that make use of state samples to evaluate the objective function. We derive a Monte Carlo approximation of mutual information suitable for robotic exploration that can be integrated with forward simulation methods, and avoids computing SLAM filter updates during the computation of optimal actions. This combined with approximating the underlying POMDP as an open loop control problem allows extending the optimization horizon beyond myopic optimization applied e.g. by Stachniss and Grisetti (2005).

3 A POMDP with belief-dependent rewards

Consider an agent in a partially observable environment. Let $s_t \in S$ denote the hidden state of the system at time $t$. The agent’s knowledge of the system state is represented by a probability density function (pdf) $b_t = p(s_t)$ over the state space $S$. This pdf is referred to as the belief state, and the belief space $B$ is the space of all pdfs over $S$. At each time step, the agent takes a control action $u_t \in U$. A Markovian state transition model $p(s_{t+1} | s_t, u_t)$ determines the evolution of the state conditional on the agent’s actions. The agent obtains data about the state via observations $z_{t+1} \in Z$, and its perceptual capabilities are described by a probabilistic observation model $p(z_{t+1} | s_{t+1}, u_t)$.

The agent’s belief state is tracked by Bayesian filtering, consisting of a prediction step and an update step. The prediction step computes the pdf of $s_{t+1}$ given the previous belief state $b_t$ and an action $u_t$ via

$$p(s_{t+1} | b_t, u_t) = \int_{s_t \in S} p(s_{t+1} | s_t, u_t)p(s_t)ds_t. \quad (1)$$

Given an observation $z_{t+1}$, the posterior belief $b_{t+1} = p(s_{t+1} | b_t, u_t, z_{t+1})$ is obtained in the update step by applying Bayes’ rule. We denote $b_{t+1} = \tau(b_t, u_t, z_{t+1})$, where $\tau : B \times U \times Z \rightarrow B$ is referred to as the belief update function

$$\tau(b_t, u_t, z_{t+1}) = \frac{p(z_{t+1} | s_{t+1}, u_t)p(s_{t+1} | b_t, u_t)}{p(z_{t+1} | b_t, u_t)}. \quad (2)$$
The normalization term in the denominator of (2) is the prior of the observation, \( p(z_{t+1} \mid b_t, u_t) = \int_{s_{t+1} \in S} p(z_{t+1} \mid s_{t+1}, u_t) p(s_{t+1} \mid b_t, u_t) ds_{t+1} \).

Given the initial belief state \( b_0 \) and a history \( h_t = \{u_0, z_1, u_1, z_2, \ldots, u_{t-1}, z_t\} \) of control actions and observations (with \( h_0 = \emptyset \)), the belief state \( b_k \) for any \( 0 < k \leq t \) may be calculated recursively applying equations (1) and (2). A belief state \( b_t \) summarises all information about the state provided by \( b_0 \) and \( h_t \) available to the agent to decide the next action \( u_t \).

The agent acts optimally when the expectation of the sum of discounted rewards over a finite horizon of \( H \) decisions is maximized. The discount factor is \( \gamma \in [0, 1] \). The reward for each decision is determined by a reward function \( R : B \times S \times U \rightarrow \mathbb{R} \). The decision making problem thus defined is called a finite horizon partially observable Markov decision process, or POMDP. Contrary to most POMDP formulations, we assume the reward function to be belief-dependent. This allows information-theoretic quantities such as mutual information or entropy to be applied as rewards.

For the sake of brevity, we denote in the following \( s_t, u_t, b_t, z_{t+1} \) and \( u_{t+1} \) as \( s, u, b, z' \) and \( u' \), respectively. According to Bellman’s (1957) principle of optimality, the expected sum of discounted rewards of executing control action \( u \) in belief state \( b \) with \( k \) decisions remaining until the end of the optimization horizon is obtained by the backward in time recursion

\[
Q_k(b, u) = \mathbb{E}_S[R(b, s, u)] + \gamma \cdot \mathbb{E}_{Z'}[\sup_{u' \in U} Q_{k-1}(\tau(b, u, z'), u')],
\]

where \( \mathbb{E} \) denotes mathematical expectation, taken over \( S \sim b \) and \( Z' \sim p(z' \mid b, u) \), respectively. The recursion starts from the expected value of a single decision, \( Q_1(b, u) = \mathbb{E}_S[R(b, s, u)] \). The solution of the POMDP is an optimal policy \( \pi_H = [\mu_1, \mu_2, \ldots, \mu_H]^{\top} \), with \( \mu_k(b) = \arg\sup_{u \in U} Q_k(b, u) \) mapping belief states to optimal actions when \( k \) decisions are remaining (Kaelbling et al, 1998). The optimal policy is closed loop; all information from the initial belief state and the history of actions and observations until the current time step is used to determine the next decision. The function \( V_k(b) = \sup_{u \in U} Q_k(b, u) \) is called the optimal value function, as it determines the maximum expected reward that may be obtained at belief state \( b \) following the optimal policy for the next \( k \) decisions.

Finding \( \pi_H \) requires reasoning over uncertain outcomes over the next \( H \) decisions, leading to a computationally hard problem (Papadimitriou and Tsitsiklis, 1987). Several approximate methods have been developed for POMDPs.
that trade off optimality for faster computation. The majority of approximation algorithms assume that the reward function is dependent on the state \(s\) and the action \(u\) only. The expected reward is then linear w.r.t. the belief state and the optimal finite-horizon value function is piecewise linear and convex (PWLC) (Smallwood and Sondik, 1973). Approximate methods can be broadly classified into offline and online methods. Offline methods (Hauskrecht, 2000; Spaan and Vlassis, 2005; Pineau et al., 2006; Kurniawati et al., 2008) calculate an approximation of the optimal policy before the agent begins executing the task. During task execution, the agent uses the offline policy as a look-up table to determine an action based on the current belief state. In contrast, the key idea of online POMDP algorithms (see Ross et al. (2008) for a review) is to repeatedly find an optimal action only for the current belief state during task execution.

Reward functions that are nonlinear in the belief state have been studied in current research literature under various assumptions. Araya et al. (2010) approximated a reward function convex w.r.t the belief state by a piecewise linear function and were able to apply standard POMDP algorithms to approximate the optimal policy. Krishnamurthy and Dionin (2007) studied a sensor scheduling problem with a nonlinear reward function and identified conditions under which the optimal policy has a monotone structure. Online belief space search algorithms operate by constructing a search tree of belief states reachable within the horizon \(H\) starting from the current belief state. Such methods may be applied with very general reward functions, provided they do not assume the PWLC property for the value function e.g. to compute bounds to prune the search tree.

A common feature of robotics problems is that the space of observations \(Z\) is very large or uncountable. This is a major problem for belief space search methods, as the number of reachable belief states is \((|U||Z|)^H\). We consider an approximation that ignores the effect of information provided by future observations, instead choosing actions only on the basis of currently available information. This leads to an open loop control problem, i.e. a problem of computing the best sequence of control actions over the finite horizon based only on the current belief state. In practice this is achieved by swapping the order of the latter expectation and supremum operators in (3). The resulting open loop objective function is

\[
J_H(b_t, u_{t:t+H-1}) = \mathbb{E} \left[ \gamma^{t-H} \sum_{k=t}^{t+H-1} \gamma^{k-t} R(b_k, s_k, u_k) \right].
\] (4)

The expectation is w.r.t \(p(s_{t:t+H}, z_{t+1:t+H} | b_t, u_{t:t+H-1})\) induced when ex-
cuting the sequence $u_{t:t+H-1}$ of control actions at belief state $b_t$. This PDF is obtained via the state transition and observation models. The belief states $b_k$ are computed according to (2). The optimal open loop action sequence is

$$\begin{align*}
u_{t:t+H-1}^* &= \arg\max_{u_{t:t+H-1} \in U^{|H|}} J_H(b_t, u_{t:t+H-1}).
\end{align*}$$

(5)

We apply a receding horizon control scheme with the open loop approximation. The problem (5) is first solved, the agent then executes the first action $u_t^*$ of the optimal action sequence and perceives an observation $z_{t+1}$. The belief state is revised to $b_{t+1} = \tau(b_t, u_t^*, z_{t+1})$, and the resulting new optimization problem of the form (5) is solved again at time $(t+1)$.

This control strategy is also referred to as open-loop feedback control (OLFC) and it has been shown to always perform at least as well as following the pure open-loop policy of executing $u_t^*$:

We consider two cases, with either a finite or uncountable action space, leading to finite or uncountable search spaces, respectively. The observation space $Z$ may be either finite or uncountable.

For finite search spaces, we present a sampling based tree search algorithm for POMDPs that we have modified to solve the open loop problem. For uncountable search spaces, we present an algorithm that applies sequential Monte Carlo (SMC) methods. Both algorithms approximately evaluate
4.1 Partially observable Monte Carlo planning

The partially observable Monte Carlo planning algorithm (POMCP) of Silver and Veness (2010) is a version of Monte Carlo Tree Search (MCTS) (Browne et al, 2012) to solve POMDPs with finite action and observation spaces. POMCP runs a sequence of forward simulations called episodes on the problem, and collects the rewards obtained in the simulation and uses them to evaluate alternative strategies by constructing a search tree over the possible policies. An exploration parameter $\epsilon$ determines the balance between exploitation and exploration behaviour during the search. We describe a version of POMCP that has been adapted to solve the open loop control problem (5).

Each node $T(h)$ in the search tree $T$ corresponds to a history trace $h$ after the current belief state $b_t$. As the open loop problem is equivalent to a POMDP without observations, every history corresponds to a sequence of actions and the successor nodes of $T(h)$ are denoted by $T(hu)$ for any $u \in U$. The root node is referred to as $T(\emptyset)$. The algorithm state is determined by a value $V(h)$ and a count $N(h)$ for each node in the search tree. The value is the mean reward over all simulations starting from $h$, and the count is the number of times a history $h$ has been perceived during the simulations. Both quantities are updated iteratively, and converge as the number of simulation episodes increases such that the optimal action can be extracted by finding the successor of the root node with the greatest value, $\text{argmax}_u V(u)$.

The procedure is stated in Algorithm 1. The algorithm takes as constant parameters the POMDP model, the exploration parameter $\epsilon$, and the optimization horizon $H$. The main function is called for the current belief state $b_t$. The search tree is initialized as empty, and the while loop (Line 3) is executed until a specified stopping criterion, e.g. the number of simulation episodes, has been reached. Each iteration of the while loop corresponds to one simulation episode.

Each episode starts by sampling an initial state $s_t$ from $b_t$ (Line 4). The function SIMULATE returns a sample of the discounted reward for an episode starting from the situation described by the input parameters: a belief state $b$, a state sample $s$, the current history $h$ and the current depth $d$ in the episode. The function is called for $b_t$, $s_t$, the initially empty history and a depth of 0.

The function SIMULATE consists of two stages. In the tree policy stage, the current search tree $T$ is applied to guide action selection. Once a history is encountered that the current search tree does not include, a rollout stage...
**Algorithm 1** POMCP ([Silver and Veness, 2010](#)) adapted for open-loop control

**Parameters:** The POMDP model, exploration parameter $e$, horizon $H$.

1: function MAIN($b_t$)
2: Initialize $T = \emptyset$
3: while stopping conditions not fulfilled do
4: Sample $s_t \sim b_t$
5: SIMULATE($b_t, s_t, \emptyset, 0$)
6: $N(\emptyset) = N(\emptyset) + 1$
7: end while
8: return argmax$_u V(u)$
9: end function

10: function SIMULATE($b, s, h, d$)
11: if $d = H$ then return 0
12: end if
13: if $h \notin T$ then
14: Add $T(hu)$ to tree $\forall u \in U$
15: return ROLLOUT($b, s, d$)
16: end if
17: $u = \text{argmax}_{u'} V(hu') + e \sqrt{\frac{\log N(h)}{N(hu')}}$
18: Sample $s' \sim p(s' | s, u), z' \sim p(z' | s', u)$
19: $b' = \tau(b, u, z')$
20: $r = R(b, s, u) + \gamma \cdot \text{SIMULATE}(b', s', hu, d + 1)$
21: $N(hu) = N(hu) + 1$
22: $V(hu) = V(hu) + \frac{r - V(hu)}{N(hu)}$
23: return $r$
24: end function

25: function ROLLOUT($b, s, d$)
26: if $d = H$ then return 0
27: end if
28: Sample $u$ from uniform distribution on $U$
29: Sample $s' \sim p(s' | s, u), z' \sim p(z' | s', u)$
30: $b' = \tau(b, u, z')$
31: return $R(b, s, u) + \gamma \cdot \text{ROLLOUT}(b', s', d + 1)$
32: end function
is entered. In the rollout stage, a given rollout policy is followed until the end of the optimization horizon.

If the condition on line 13 is false, we can take advantage of the current search tree to guide the action selection in the tree policy stage. The action at node $T(h)$ is determined by maximizing an upper confidence bound (Auer et al., 2002; Kocsis and Szepesvári, 2006),

$$u = \arg\max_{u'} V(hu') + e \sqrt{\log N(h)/N(hu')}$$

(Line 17). The first term is the current estimate of the expected value of the action, and the second term is an exploration bonus dependent on the exploration parameter $e$. The parameter $e$ is typically selected such that it is on a similar scale than typical rewards in the problem. The exploration bonus encourages trying actions that have rarely been tried, prompting exploratory behavior. Samples for the resulting state and observation after executing $u$ are drawn, and the next belief state is computed (Lines 18-19).

Recursion is applied to find the discounted reward until the end of the horizon (Line 20). Finally, the counts and values of all nodes visited during the episode are updated with the reward information gained during the episode in the backpropagation phase (Lines 21-22).

If the condition on line 13 is true, a node not yet in the tree has been reached. The node $T(h)$ and its possible successor nodes are added to the tree with counts and values equal to zero. Nonzero values can be used to incorporate prior heuristic information, if desired. The rollout stage is then entered. The rollout function takes a similar set of input arguments as simulate except for the history $h$ which is not required. Likewise, it returns a sample of the discounted reward until end of the optimization horizon, but under a given rollout policy. We apply the widely used rollout policy of sampling actions uniformly at random from $U$ (Line 28). Samples of the next state and observation are drawn, and the corresponding next belief state is calculated (Lines 29-30). The discounted reward until the end of the horizon is obtained by recursion (Line 31).

Once the while loop (Line 3) terminates, the optimal action $u^*_t$ according to the information accrued to the search tree $T$ by simulation episodes is returned (Line 8). To apply the receding horizon control scheme, after executing $u^*_t$ and perceiving $z_{t+1}$ from the real system, the search process is repeated at decision step $t + 1$ for the resulting belief state $\tau(b_t, u^*_t, z_{t+1})$. POMCP has been shown to converge in probability to the optimal solution (Silver and Veness, 2010). Computation time can be decreased by running several episodes in parallel, see Browne et al. (2012) for an overview of par-

---

1If $N(hu') = 0$ for any $u' \in U$, the action is instead sampled uniformly at random from $\{u' \in U \mid N(hu') = 0\}$. 

11
allelisation strategies for MCTS algorithms.

4.2 Sequential Monte Carlo optimization

Maximizing (4) is a special case of a more general problem of maximizing a function of the form

\[ J_H(\phi) = \int_Y G(y, \phi)p(y | \phi)dy, \]

where \( y = [s_{t:t+H}, z_{t+1:t+H}]^T, \phi = [b_t, u_{t:t+H-1}]^T \) and \( G \) is a function that encompasses the summation over the rewards as in (4).

Kantas et al (2009) apply a sequential Monte Carlo (SMC) method to optimization problems such as (6). They generate particles by sampling from \( p(y | \phi) \), and evaluate a Monte Carlo approximations of the expectation of \( G(y, \phi)\). The evaluation guides a search in the space of \( \phi \), leading to discarding or keeping particles for the next evaluation phase.

The approach is related to the Bayesian interpretation of simulated annealing and maximum likelihood estimation (Johansen et al, 2008). The key idea is to construct a sequence of distributions \( \lambda_\eta \sim p(\phi)J_H(\phi)^\eta \) where \( p(\phi) \) is an arbitrary prior that is nonzero at the maximizers of (4). As \( \eta \rightarrow \infty \), \( \lambda_\eta(\phi) \) becomes concentrated on the set of maximizers of \( J_H \). In practice, the algorithm evaluates a set of particles via the objective function and based on the evaluation result particles are either discarded or carried forward to the next evaluation round. Eventually the particle set converges to represent an action sequence maximizing (4). For this method, \( G \) must always be finite and strictly positive. By adding a finite positive constant, any finite-valued function can be modified to fulfill this requirement without affecting the argument maximizing (4). Note that as in our case \( b_t \) is given, the search is effectively over \( u_{t:t+H-1} \in U^{[H]} \).

At time \( t \), the algorithm iterates over \( l = 1, \ldots, l_{\text{max}} \) the procedure shown in Algorithm 2. The number of iterations \( l_{\text{max}} \) is chosen according to the accuracy and run-time requirements. The algorithm maintains a set of \( M \) particles. At iteration \( l \), each particle indexed by \( i \) with an associated weight \( w_l^{(i)} \) represents a sequence of actions \( u_{k:k+H-1,l}^{(i)} \) and a set of \( \eta_l \) state and observation sequences conditional on the action sequence, denoted \( s_{k+1:k+H-1,l}^{(i)} \) and \( z_{j+1:j+H-1,l}^{(i)} \), \( j = 1, \ldots, \eta_l \), respectively. The integers \( \{\eta_l\}_{l \geq 1} \) must form a strictly increasing sequence. The algorithm has been shown to converge to the optimal solution for logarithmic sequences, although in practice faster increasing linear or geometric sequences are used (Johansen et al).
**Algorithm 2** Sequential Monte Carlo optimization, adapted from (Kantas et al, 2009).

**Parameters:** The POMDP model, number of iterations $l_{\text{max}}$, increasing integer sequence $\{\eta_l\}_{l=1}^{l_{\text{max}}}$, sampling kernels $q_l$, horizon $H$.

```
1: function SMC($b_t$)
2:     for $l = 1, \ldots, l_{\text{max}}$ do
3:         for $i = 1, \ldots, M$ do
4:             $u_{t:t+H-1,i}^{(i)} \sim q_l(\cdot \mid u_{t:t+H-1,i,-1})$
5:             for $j = 1, \ldots, \eta_l$ do
6:                 $\{s_{t:t+H,j}^{(i)}, z_{t+1:t+H,j}^{(i)}\} \sim p(s_{t:t+H}, z_{t+1:t+H} \mid b_t, u_{t:t+H-1,i}^{(i)})$
7:             end for
8:             $w_l^{(i)} = w_{l-1}^{(i)} \prod_{j=1}^{t+H-1} \sum_{n=t}^{\gamma^{(n-i)}} R(b_{n,j}^{(i)}, s_{n,j}^{(i)}, u_{n}^{(i)})$
9:             where $b_{n+1,j}^{(i)} = \tau(b_{n,j}^{(i)}, u_{n,l;i}, z_{n+1,j}^{(i)})$, and $b_{t,j}^{(i)} = b_t \forall j$
10:         end for
11:     Normalize weights: $w_l^{(i)} = \frac{w_l^{(i)}}{\sum_{i=1}^{M} w_l^{(i)}}$
12:     if resampling required then
13:         Resample particles and set equal weights: $w_l^{(i)} = 1/M \forall i$
14:     end if
15: end for
16: end function
```
The sequence \( \{\eta_l\}_{l \geq 1} \) is analogous to the temperature cooling schedule in simulated annealing.

An action sequence is sampled for each particle from a kernel \( q_l \) (Line 4). In general, the choice of \( q_l \) depends on the problem specifics. The selection of the kernel determines how new action sequence samples depend on the earlier ones, and it has a central role in determining how efficient the algorithm is. For further discussion on kernel selection, we direct the reader to Del Moral et al (2006). For \( l = 1 \), the initial action sequences are sampled from a prior distribution that is nonzero at the maximizers of (6). After action trajectories have been sampled, they are applied to generate \( \eta_l \) replicas of possible state-observation sequences when executing the control sequence \( u_{k:k+H-1,l} \) (Lines 5-7). The sampling is simplified by noting

\[
p(s_{t:t+H}, z_{t+1:t+H} \mid b_t, u_{t:t+H-1,l}^{(i)}) = p(s_t) \prod_{n=t}^{t+H-1} p(z_{n+1} \mid s_{n+1}, u_{n,l}^{(i)}) p(s_{n+1} \mid s_n, u_{n,l}^{(i)}). \tag{7}
\]

The weights of the particles are updated by approximating the reward function applying the state and observation samples contained in each particle (Lines 8-9). Finally, resampling is performed if the effective sample size \( N_{\text{eff}} = 1 / \sum_{i=1}^{M} (w_l^{(i)})^2 \) falls below a threshold value (Lines 12-14).

After step \( l = l_{\text{max}} \), the maximizer estimate is extracted as \( u_{t:t+H-1,l_{\text{max}}}^{(i_m)} \), where \( i_m = \arg\max_i w_l^{(i)} \). In the receding horizon manner, the first action \( u_{l,l_{\text{max}}}^{(i_m)} \) is applied and an observation \( z_{t+1} \) is perceived. The belief state is updated and the algorithm is repeated to make the decision for step \( (t + 1) \). Algorithm 2 is easily parallelized, as each particle can be processed independently.

Fig. 1 illustrates how the SMC algorithm behaves over the iterations \( l \). The figure shows a trajectory optimization problem where a robot starts from an initial position the middle of the opening in the top part of the map, and must optimize its trajectory according to some objective function. Each subfigure depicts a specified iteration of the algorithm. Each line in the figure depicts a trajectory corresponding to a single particle. The trajectory corresponding to the particle with the greatest weight is drawn with a thick dashed line. The initially scattered trajectories at \( l = 1 \) become increasingly concentrated on maximizers of the objective function as the
Figure 1: Convergence of the SMC algorithm in a trajectory optimization problem for iterations \( l = 1, 4, 8, 12 \). The dark areas denote traversable and light areas non-traversable parts of the environment. The lines denote possible trajectories, and the trajectory for the particle with the greatest weight is shown as a thicker dashed line.

iterations progress towards the final iteration step \( l = l_{\text{max}} = 12 \).

5 Mutual information in robotic exploration problems

Mutual information (MI) has been applied as an objective function in various exploration and information gathering problems in robotics, see e.g. (Atanasov et al, 2014; Hitz et al, 2014; Charrow et al, 2014; Lauri and Ritala, 2015). Some information gathering tasks such as classification or target tracking (Spaan and Lima, 2009; Chong et al, 2009) can be formulated through
purely state-dependent reward functions, but they are not a natural choice e.g. for environmental surveillance problems, where the objective is to maintain an accurate belief over the state. In many problems there may also be multiple goals; for example in the case of robotic exploration, simultaneous localization and mapping. Kreucher et al. (2005) showed that policies optimizing MI can perform nearly as well over several goals as specific policies only optimizing performance w.r.t. a single goal. A policy of sequential maximization of mutual information will choose actions that reduce the entropy of the state estimate the most.

In robotic exploration the state can be decomposed into the robot’s internal state \(x \in X\) and the environment state \(m \in M\), i.e. \(S = X \times M\). The internal state represents the pose of the robot, and the environment state represents all relevant features of the world. We denote in the following \(x_t, m_t, u_t, b_t, x_{t+1}, m_{t+1}\) and \(z_{t+1}\) as \(x, m, u, b, x', m'\) and \(z'\), respectively. By definition, MI between the state and observation given the belief \(b\) and action \(u\) is expressed by the measurement prior \(p(z' \mid b, u)\), the state posterior \(p(x', m' \mid b, u, z')\) and the state prediction \(p(x', m' \mid b, u)\) as

\[
I(X, M; Z \mid b, u) = \int_{Z} p(z' \mid b, u) \\
\int_{X} \int_{M} p(x', m' \mid b, u, z') \log \frac{p(x', m' \mid b, u, z')}{p(x', m' \mid b, u)} \mathrm{d}m' \mathrm{d}x' \mathrm{d}z'.
\]

The robot is assumed unable to affect the environment state through its actions, and thus the robot’s internal state and environment state evolve independently:

\[
p(x', m' \mid x, m, u) = p(x' \mid x, u)p(m' \mid m).
\]

This does not imply that the internal and environment state are independent, in general \(p(x, m) \neq p(x)p(m)\).

We derive a particle approximation for (8) under assumption (9). By the chain rule for mutual information (Cover and Thomas, 2006), (8) is equal to

\[
I(X, M; Z \mid b, u) = I(X; Z \mid b, u) + I(M; Z \mid X, b, u).
\]

The first term of (10) is the MI of \(X\) and \(Z\),

\[
I(X; Z \mid b, u) = \int_{Z} \int_{X} p(x', z' \mid b, u) \log \frac{p(x', z' \mid b, u)}{p(x' \mid b, u)} \mathrm{d}x' \mathrm{d}z'.
\]
The second term of (10) is the conditional MI of $M$ and $Z$ given $X$. By definition (Cover and Thomas, 2006)

$$I(M; Z | X, b, u) = \mathbb{E} \left[ \log \frac{p(m', z' | b, u, x')} {p(m' | b, u, x') p(z' | b, u, x')} \right],$$

(12)

where the expectation is w.r.t $p(z', x', m' | b, u)$. By (9), $p(m' | b, u, x') = p(m' | b, u)$, and by the rule of conditional probability we obtain

$$I(M; Z | X, b, u) = \int_{Z} \int_{X} p(x', z' | b, u) \int_{M} p(m' | b, u, z', x') \log \frac{p(m' | b, u, z', x')} {p(m' | b, u)} \text{d}m' \text{d}x' \text{d}z'. \tag{13}$$

Since (11) and (13) are both expectations under the joint pdf $p(x', z' | b, u)$, by sampling from this pdf we can derive a particle approximation for (8). We draw a sample from $b$, the current belief state: $\{x(i), m(i)\}_{i=1}^{N} \sim p(x, m)$. The samples are propagated through the state transition model (9), we get $x'(i) \sim p(x' | x(i), u)$ and $m'(i) \sim p(m' | m(i))$. The particles thus obtained follow the distribution $p(x', m' | b, u)$. Since

$$p(x', z' | b, u) = \int_{M} p(z' | x', m', u) p(x', m' | b, u) \text{d}m',$$

(14)

we then draw $z'(i) \sim p(z' | x'(i), m'(i), u)$ applying the observation model. Finally, we have obtained a particle set $\{x(i), z(i)\}_{i=1}^{N} \sim p(x', z' | b, u)$. We obtain a particle approximation of (8):

$$I(X, M; Z | b, u) \approx \frac{1}{N} \sum_{i=1}^{N} \left[ \log \frac{p(x'(i) | b, u, z'(i))} {p(x'(i) | b, u)} + \int_{M} p(m' | b, u, z'(i), x'(i)) \log \frac{p(m' | b, u, z'(i), x'(i))} {p(m' | b, u)} \text{d}m' \text{d}x' \text{d}z'(i) \right]. \tag{15}$$

The first term in the sum is the expected information gain on the internal state, and the second sum term is the expected information gain on the environment state following an action. The integral term is equal to the Kullback-Leibler (KL) divergence between the posterior and predictive environment state information. Note that we have elected not to apply a particle approximation to this integral, since it is often possible to compute
the pdf $p(m' \mid b, u, z^{(i)}, x^{(i)})$ in closed form for known state and observation. The particle approximation \cite{15} may be combined with the evaluation of the objective function in Algorithm 1 (Lines 20, 31) and Algorithm 2 (Line 8).

Our approximation places some light restrictions on the system models. First, we must be able to draw samples from the state transition and observation models either directly or e.g. applying importance sampling. Secondly, evaluating the first sum term in (15) requires us to be able to evaluate the predictive probability and the posterior probability of an internal state $x^{(i)}$ given an action $u$ and a measurement $z^{(i)}$. The predictive pdf can be evaluated applying the state transition model. It is not generally easy to find the posterior $p(x^{(i)} \mid b, u, z^{(i)})$, but in some cases it can be approximated by a unimodal (typically Gaussian) distribution. As argued by Grisetti et al. \cite{2007}, such a case arises e.g. for robots equipped with accurate range sensors such as laser range finders (LRFs). LRF data can be leveraged via scan matching to obtain localization estimates that are significantly more precise than estimates based only on robot’s state transition models. Consequently, the measurement likelihood $p(z' \mid x', m', u)$ is strongly peaked and the posterior pdf $p(x' \mid b, u, z') \propto p(z' \mid x', m', u)p(x' \mid b, u)$ is unimodal, dominated by the measurement likelihood and concentrated on a significantly smaller area of $X$ than the predictive distribution $p(x' \mid b, u)$. These factors allow approximation of the posterior as a normal distribution $p(x^{(i)} \mid b, u, z^{(i)}) \approx N(x^{(i)} \mid \hat{x}, \Sigma)$ with a small covariance $\Sigma$, where the mean value is equal to the maximum likelihood pose

$$\hat{x} = \arg\max_{x' \in X} p(z^{(i)} \mid x', m^{(i)}, u). \quad (16)$$

Finally, evaluating the integral term in the sum of (15) requires us to be able to compute posterior map pdfs given a pose and an observation. This corresponds to the problem of mapping with known poses \cite{Moravec1988}. We must also be able to compute the KL divergence between the two pdfs over the environment state. The difficulty of this depends on the map representation, but is simple e.g. for occupancy grid maps due to the independence assumption of the grid cells.

Note that this approximation can equally well be used in cases where the robot pose can be assumed to be known exactly. In this case the first sum term in (15) is zero as the pdfs are degenerate Dirac delta functions.

The overall effect of the particle approximation is that we avoid computing the state posterior pdf and solving the implied SLAM problem when evaluating (8). This in no way precludes executing a SLAM algorithm for
the real process independently of the task of finding an optimal open loop action sequence. The current belief state \( b = p(x, m) \) provided by the SLAM algorithm is applied as initial information for finding the optimal open loop solution. Sometimes the real process SLAM particle filter may be susceptible to losing consistency (Blanco et al. 2008; Carlone et al. 2010), and applying the proposed approximation may lead to a failure state as this possibility is not considered while planning trajectories. In such cases, either the SLAM algorithm must be improved or an alternative approximation for MI applied.

6 Simulation experiments

We studied the performance of the POMCP algorithm 1 and SMC algorithm 2 for MI-exploration problems in dynamic environments by simulation. The agent was assumed to be a car-like vehicle traversing a Cartesian space \( W \subset \mathbb{R}^2 \), controlled by \( u = (v, \omega) \), where \( v \) is the linear velocity and \( \omega \) is the steering angle. Throughout the simulations, the agent’s pose was assumed to be fully observable, and the motion model noiseless.

![Simulation environment and example trajectories](image)

Figure 2: The simulation environment and example trajectories in the slow dynamics case from the MC, POMCP (\( N = 4000, H = 5 \)) and SMC (\( M = 100, H = 5 \)) algorithms. Obstacles are marked by white color and traversable area by black color. The poses of the agent at decision times are shown by circle markers. The agent starts at the pose below the opening in the middle-top part of the figure.
An environment as shown in Fig. 2 was defined. Obstacles known to the agent are marked white. The black area consists of cells whose occupancy was a priori unknown. While considering potential control sequences, those that lead the agent to collide with a known obstacle were rejected. The initial pose $x_0$ of the agent is at the middle of the map, between the opening in the wall-like obstacle at the top part of the middle room, with orientation facing directly towards the bottom of the figure. The size of the area was 14 by 14 meters with 0.10 meter rectangular cells, in total 19600 cells. The environment features three dead-ends in the upper and lower left corner and lower middle sections of the map, respectively.

To model environment dynamics, we apply dynamic occupancy grids (Meyer-Delius et al., 2012), an extension of classic occupancy grids Moravec (1988). The cells are in one of two hidden states, free (0) or occupied (1), respectively. Cell dynamics are statistically mutually independent two-state Markov chains with parameters $p^{o|o}_c$ and $p^{o|f}_c$ known to the agent, depicting the conditional probabilities that the cell $c$ remains in the occupied state and that it transitions from the free state to the occupied state, respectively. Due to the independence of cells, the environment state transition model $p(m' | m)$ is the product of the individual cells’ state transition models.

![Figure 3: Occupancy grids and observed areas. The robot’s location $x$ is marked by a black circle, and the dashed sector depicts sensor range. The grey cells denote the subset $\tilde{W}(x) \subset W$ of the occupancy grid that is potentially sensed. Left: a single measurement. Right: measurements along a trajectory depicted by the dashed line between locations.](image)

A cell’s occupancy probability is updated according to the state transition model and whenever an observation of the cell is made. Depending on the properties of the robot’s sensors, a subset $\tilde{W}(x) \subset W$ of the map is
sensed from pose $x$ (see Fig. 3). The agent was equipped with a proximity sensor that measured the occupancy state of all cells within 1.5 meters of the agent’s location, with false positive and false negative probabilities equal to 0.05.

Due to cell independence, the integral term in (15) becomes a sum over the KL divergences between the cells’ posterior and predictive occupancy probabilities for each $c \in W$. As seen from Fig. 3, only a subset of the map cells is observed at any given time. Computational resources can be saved by sampling lazily, i.e. only drawing samples $m^{(i)}$ in parts of the map that are actually observed and thus affect $z'$. The implementation details of such a sampling scheme are given in Appendix A.

In a two-state Markov chain, the pdf over the states tends monotonically towards the stationary distribution. If the occupancy probability of a cell $c$ at time $t$ is $p_c(t)$ and no observations are perceived, the Markov chain distribution after $n$ steps is

$$p_c(t + n) = p_c^s + \exp(-n/n_0)(p_c(t) - p_c^s),$$

(17)

where $p_c^s = p_c^{o|f}/(1 - p_c^{o|o} + p_c^{o|f})$ is the stationary distribution of the Markov chain and $n_0 = -1/\log(p_c^{o|o} - p_c^{o|f})$ quantifies the rate at which the stationary distribution is approached. We considered three cases for the environment dynamics. In the first case, we set $p_c^{o|f} = 0.01$ and $p_c^{o|o} = 0.99\forall c \in W$, resulting in slow dynamics ($n_0 \approx 49.5$). In the second case, $p_c^{o|f} = 0.15$ and $p_c^{o|o} = 0.85\forall c \in W$, resulting in fast dynamics ($n_0 \approx 2.8$). The final case was a combination of the two, with $p_c^{o|f} \in [0.01, 0.15], p_c^{o|o} \in [0.85, 0.99]$ sampled uniformly at random independently for each cell. We refer to this case as having medium dynamics. In all cases, the initial information $p_c(0)$ on the map was equal to the stationary distribution $p_c^s$ for each cell $c$.

For comparison, the informativeness of each action was evaluated with a simple myopic Monte Carlo (MC) approximation of the MI according to (15) with 50 samples. The action space was discretized uniformly to 72 control values. This corresponds to maximizing $J_1$ (see (1)), and is similar to the myopic objective function applied e.g. in [Stachniss and Grisetti, 2005; Sim and Roy, 2005; Kollar and Roy, 2008].

For POMCP (Algorithm 1), the number of simulation episodes $N$ was set equal to 1000, 2000 or 4000. The same discretized action space was applied as for the MC approximation. The exploration parameter $e$ was set to 200 to scale the exploration bonus to a range comparable with typical per-decision MI values. An upper bound for $e$ may be estimated by $e < \frac{N \cdot I_{\text{max}}}{H} \sum_{t=0}^{H-1} \gamma^t$, where
where $N_c$ is an upper bound for the number of cells observed per time step, and $I_{max}$ is an upper bound for MI per cell, obtained from the observation model.

For SMC (Algorithm 2), the number of particles $M$ was set equal to 25, 50 or 100, and $l_{max} = 12$ iterations were run. The schedule for number of state-observation replicas was linear with $\eta_l = 5 + 3(l - 1)$, where $l$ is the iteration number. Resampling was triggered by the effective sample size $N_{eff}$ becoming less than $M/4$. We empirically determined these parameters to be suitable for the experiment, observing e.g. that the trajectory judged optimal remained almost constant after a few iterations, see Fig. 1. It is more likely that the agent obtains more information while covering a large fraction of the environment with its sensors. This fact was taken into account when sampling the initial control sequences by giving higher probability to velocities closer to the agent’s maximum velocity. Steering angles were sampled uniformly at random. Independent Gaussian kernels were applied when modifying solution candidates (Line 4 in Algorithm 2) with variance decreasing proportional to the inverse square of the iteration $l$ as a fraction of the full range of possible velocity and steering angle values.

The discount factor in all cases was $\gamma = 0.95$. The optimization horizon $H$ was varied between 1 and 7. In each simulation run, 30 decisions were implemented and the results recorded. For each algorithm and value of $H$, the simulation was repeated 12 times. All simulations started from the same underlying true state.

Table 1 summarizes the results for the reference MC method. As the rate of dynamics increases, the total MI tends to increase. The stationary occupancy probability $p_s$ is 0.5 in each cell for slow and fast dynamics, and on average for medium dynamics. Thus, the maximum amount of information about cell occupancy is available when the cell is observed when its Markov chain is at its stationary distribution. As the dynamics rate increases, the Markov chain tends faster towards the stationary distribution, and more MI is accumulated. The greater variation for slow dynamics is also due to the greater variance in the occupancy probabilities, which do not quickly return to the stationary distribution.

Table 1: The mean of total mutual information (± its 95% confidence interval) for the reference myopic Monte Carlo method, for each rate of dynamics.

<table>
<thead>
<tr>
<th>Dynamics rate</th>
<th>Slow</th>
<th>Medium</th>
<th>Fast</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total MI [bits]</td>
<td>7360 ± 601</td>
<td>10124 ± 66</td>
<td>12508 ± 78</td>
</tr>
</tbody>
</table>
Fig. 4 shows the improvements over the reference results in Table 1 obtained with the POMCP and SMC planners for each optimization horizon, for each dynamics rate. The mean improvement and its 95% confidence intervals are shown. Looking at $H = 1$ in all cases, we note that POMCP has performance statistically equal to the reference, as expected. SMC performance is similar with the exception of the case of fast dynamics, although the difference even there is slight. Since the SMC method which considers the full continuous action space achieves similar performance as POMCP applying a discretized action space, we can conclude that our discretization of the action space is dense enough for the myopic problem.
Figure 4: Performance of POMCP (left) and SMC (right) planners compared to the reference method, in the case of slow (top), medium (middle), or fast dynamics (bottom). The bars indicate mean improvement over reference level (Table 1). The 95% confidence intervals are shown by the horizontal error bars. The dashed line indicates the 95% confidence interval of the reference level.
In all cases, performance tends to improve as the number of simulations $N$ in the POMCP algorithm or the number of particles $M$ in the SMC algorithm increase. Increasing these parameters improves the coverage over the search space in (5) and the accuracy of the MI approximation (15), helping to find solutions with greater expected MI more reliably.

Considering the cases with $H > 1$ in Fig. 4, we see the greatest improvement over the reference for slow dynamics. For slow dynamics, moving away from the currently sensed area will provide a greater amount of MI than staying in the same area. However, a myopic planner with $H = 1$ was observed to select actions that lead to dead-ends. We observed the relative frequency each cell was observed, and noted that for $H = 1$ most frequently observed cells were at the lower middle or lower left corner of the environment, whereas for $H > 1$ cells around the map were observed equally frequently. Example trajectories illustrating this are shown in Fig. 2 for MC, POMCP ($N = 4000, H = 5$) and SMC ($M = 100, H = 5$). The MC trajectory initially gets stuck at the lower part of the map, while both POMCP and SMC avoid this dead-end. In general, the presence of actions that are informative but ultimately lead to a situation with few options to gain more information can be avoided by multi-step planning.

For $H > 1$ in the case of medium and fast dynamics, Fig. 4 shows mixed results. The rapid decrease of POMCP performance in these cases for $H \geq 4$ is due to insufficient sampling of the search space. The number of possible action sequences increases exponentially as a function of $H$. As $H$ increases, a constant number of samples covers an increasingly smaller fraction of possible action sequences and related possible realizations of the environment state. For example, with 4000 simulations, the typical maximum depth of the search tree generated by the POMCP algorithm was 4 in our experiments. This indicates that more simulations would be needed to find action sequences that cover a longer optimization horizon. The effect is amplified for faster dynamics, since for slow dynamics even a few samples of the environment state provide an accurate prediction. A similar effect is seen also for SMC, although less severe. There, likewise, it is increasingly difficult to cover the exponentially increasing search space by a constant number of particles.

If a long optimization horizon results in trajectories that lead the robot beyond its current sensor range, the evaluated usefulness of such trajectories depends greatly on the prior map information in these areas. Consider for example the case of Fig. 3. For areas outside current sensor range, the planner must sample from the prior distribution of the environment state in these areas. For medium and fast dynamics this distribution was typically
close to \( p^*_c = 0.5 \). On average, sampling from this uniform distribution results in equal estimates for the usefulness of long trajectories. For more heterogeneous prior information, the issue is not as severe. The length of a useful optimization horizon may be estimated by comparing the maximum range of the agent’s sensors, its average velocity and the rate of environment dynamics.

As for the environment dynamics, we may conclude that increasing the optimization horizon is useful up to the order of \( n_0/2 \) time steps. Consider \( n_0 \) in [17] as an inverse exponential decay parameter. After \( n_0 \) steps, the occupancy distribution is again close to the stationary distribution \( p^*_c \) and observing the cell again is beneficial. Specifically in the case of \( p^*_c = 0.5 \), reobserving the cell is maximally useful the closer its occupancy distribution is to \( p^*_c \).

![Figure 5: Average entropy (lines) and its 95% confidence intervals (bars) as a function of the distance travelled in the simulation using POMCP with \( N = 2000 \) simulations (left) and SMC with \( M = 100 \) particles (right), in the case of slow dynamics. The Monte Carlo reference method’s average entropy and its 95% CI are shown by the dashed line and shaded region, respectively.](image)

We finally examined the evolution of the fraction of remaining map entropy as a function of distance travelled by the agent in the simulations with slow dynamics rate. Representative results for both algorithms are shown in Fig. [5]. For clarity of presentation, the results are shown for \( H = 1, 3, 5, 7 \).
and the error bars are not drawn for every data point. We note that like for the total MI, \( H = 1 \) has equal performance compared to the reference MC algorithm. For horizons \( 2 \leq H \leq 4 \), both POMCP and SMC typically achieve a significantly lower entropy than the reference method and thus a more informative belief state about the map. Results are mixed for \( H > 4 \) due to the reasons outlined above. For medium and fast dynamics, the occupancy information quickly returns to the stationary distribution. Due to this, the entropy as a function of time or distance is almost constant and is not shown here.

Applying multi-step planning in the simulation environment studied results in significantly better performance than myopic planning, both measured by the total mutual information collected and by the evolution of entropy of the map information over distance travelled. Multi-step planning avoids actions that seem useful at first but lead to situations where there are few possibilities for further information gain. We however also observed that increasing the optimization horizon does not monotonically improve the results, and noted that this is due to decreasing coverage of the space of possible open loop solutions, and the combined effect of proposed trajectories leading the agent outside current sensor range and environment dynamics.

7 Real-world implementation and experiments

In this section, we present details of the software implementation of the planning approach presented. The implementation is used alongside an existing SLAM algorithm. The experimental setup designed for testing the performance of multi-step planning in a real-world exploration scenario is described, and the results are reviewed.

7.1 Software implementation

Our implementation of the planning approach described in this article is shown in Fig. 6. The goal of the implementation design was to enable use of the planning algorithm together with an existing SLAM system, shown at the top left of the figure, with minimal redesigning of existing software. Thus, we implement the planner as working alongside a module for estimating the state transition probabilities \( p_{o|f} \) and \( p_{o|o} \), a task that is typically not handled by pre-existing SLAM implementations. A static environment may be handled by removing the dynamics estimation module and instead assuming a stationary map in the planning module.
The dynamics estimation module shown on the top right of the figure takes as input the current map and pose information $p(x_t, m_t)$ from SLAM along with the sensor data $z_t$, e.g. odometry and LRF data. We describe a minimal implementation for the dynamic occupancy grid maps introduced earlier, and direct the reader to Meyer-Delius et al (2012); Tipaldi et al (2013) for a more refined solution. First, the joint pdf of the pose and map obtained as SLAM output is marginalised to obtain a marginal pdf of the pose. The maximum likelihood solution $\hat{x}_t$ for the pose is extracted. We then update estimates of map dynamics based on a Bayesian update rule for Binomial parameter estimation, assuming that the data $z_t$ was collected at the maximum likelihood pose. The module outputs the current maximum likelihood estimates $\hat{p}_{c|f}$ and $\hat{p}_{c|o}$ of the state transition probabilities.

The planner module shown on the bottom left of the figure takes as input the current SLAM estimate of the robot’s internal state and the environment state, and the parameter estimates $\hat{p}_{c|f}$ and $\hat{p}_{c|o}$. Based on an implementation of Algorithm 1 or 2, the planner computes the action sequence $u^*_{t:t+H-1}$ that is then output to an executive module.

The implementation as discussed here allows applying existing SLAM algorithms without modification, and only an implementation of a corresponding dynamics estimation module is required besides implementing the planner itself.
7.2 Exploration experiments

An exploration experiment was carried out with a car-like mobile robot platform equipped with a laser range finder. The robot was controlled by a linear velocity $v$ and a steering angle $\omega$. As these quantities reside in continuous spaces and we wished to not limit the space of possible solutions explored, the experiments applied exclusively the SMC algorithm \(^2\). In addition, compared to the POMCP algorithm \(^1\) with a discrete set of actions, the SMC algorithm dynamically varies the action sampling space depending on the situation. The SLAM module in Fig. 6 was the RBPF SLAM algorithm of Grisetti et al. (2007). The dynamics estimation module and planner module were implemented in C++ as components in the Robot Operating System (ROS) framework (Quigley et al., 2009). All computations were performed using the robot’s onboard computer, equipped with an Intel i7 multicore processor, 16GB of RAM and running a Linux operating system.

A map of the experiment area is shown in Fig. 7. The size of the map is approximately 10 by 20 meters, with a cell resolution of 0.1 meters. The map is obtained by SLAM with a human teleoperating the robot and visiting each area in the environment. The trajectory travelled during teleoperation is drawn as a solid line. The environment has several open areas connected together by narrow openings. The walls between the open areas prevent visibility to much of the environment, necessitating exploration to fully reveal the structure of the environment. In addition, there are always at least two optional routes for further advance. Once the robot commits to one of the routes, switching to the other requires manoeuvring that takes time. In the simulation experiment, such features favoured multi-step planning while greedy algorithms performed poorly. Due to difficulty of implementing dynamic features within the environment consistent enough to preserve the repeatability between experiments, we did not attempt to include such elements. Instead, the robot learned the essentially static map of the environment.

The experiments applied an optimization horizon $H$ of 1, 3, 5 or 7 decisions. The other parameters for the planner were selected based on the results of the simulation experiment and to obtain reasonably short computation times. The number of iterations in the SMC algorithm was $l_{\text{max}} = 6$, with a linear schedule for the number of simulations, $\eta = 5 \cdot l$. The number of particles was $M = 25$. A Gaussian kernel with variance decreasing proportional to the inverse square of the iteration $l$ was applied in modifying the solution candidates. While sampling of control sequences, we checked their corresponding trajectories against the current map information and
Figure 7: A map of the experimental area. White, black and gray cells denote free, occupied and unknown areas, respectively. The approximate starting location in the experiments is indicated by a circle marker. The trajectory traveled under human teleoperation is drawn with a solid line. Example trajectories for cases $H = 1$ and $H = 3$ have been drawn with a dashed and dotted lines, respectively.

rejected trajectories that intersected cells where occupancy probability was greater than 0.15.

A car-like dynamic model was applied for the vehicle. As the terrain in the experimental area was flat, the robot could execute single movement actions very accurately. Thus the applied model was assumed noise-free. This coincides with the assumption of reaching the maximum likelihood pose $\hat{x}_{t+1}$ as a result of movement actions, see Section [5]. Thus mutual information of robot pose and observation data, the first term in the sum of (15), is assumed equal to zero.

For computing the mutual information of the map and observation, laser scans were sampled in the frontal sector of the robot. Incidence angles between -45 and 45 degrees were considered. The angular resolution was 0.5 degrees, and the maximum distance considered was 10 meters. The planner thus attempts to maximize mutual information of observations in this sector in front of the robot. The laser observation model was a simple ray-tracing model, with false positive and negative probabilities of 0.05 each. For sampling observations to evaluate (15), we obtained observations by ray tracing
beams from location $x_{i+1}^{(i)}$ in the known map sample $m_{i+1}^{(i)}$. The lazy sampling scheme outlined in Appendix A was applied to improve computational efficiency.

Due to the simplicity of the models applied and by using parallel processing for the independent simulations in the planner, we were able to obtain planning times of less than 1 second for planning horizons 1 and 3, and typical times of less than 4 seconds even for longer planning horizons, as indicated by Fig. 8. Although computation time is strongly dependent on the choice of parameters and computer hardware, we believe this is an indication that the planner may be fast enough for some real-time applications.

![Figure 8: Computation time as a function of optimization horizon $H$. The lower and upper edges of the boxes indicate 25th and 75th percentiles, respectively. The line across the boxes indicates the median. The whiskers cover approximately 99% of the variation in the data. Outliers are drawn with dot markers.](image)

For each optimization horizon $H$, the experiment was repeated for 5 times with the same robot starting pose. Each repetition was terminated at a cut-off time of approximately 4 minutes, or when the robot had completed exploration, as judged by the experimenter monitoring the quality of the map produced, or when a failure state occurred. Failure states include the algorithm getting stuck finding no feasible control trajectories, or, as judged by the experimenter, not making progress in exploration e.g. moving repeatedly around in the same location. A summary of qualitative results
of the experiments is provided in Table 2. We note that most failures are due to no progress being made in exploration, while only 2 failures in the total of 20 experiments occurred due to the planner finding no feasible solutions. We believe this to be an indication of the robustness of the planning algorithm, as the current failure rate was achieved with a straightforward performance-oriented implementation of the planner. A typical cause for the "no progress" type failures was repeatedly moving around in the same area. As concluded by Amigoni and Caglioti (2010), even better exploration performance might be achieved by simultaneously considering the amount of information gathered and the cost of traversing a trajectory, e.g. time or energy required.

Table 2: A qualitative summary of the experimental results. For each horizon the 5 experiments are classified into successes and failures. Failures are further classified by their reason, either due to planning algorithm finding no feasible solutions or no progress being made in exploring the environment (as judged by the experimenter).

<table>
<thead>
<tr>
<th>Horizon $H$</th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>7</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>Failure</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>Planner stuck</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>No progress</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

To evaluate the exploration performance of the algorithm, we compared the fraction of full map entropy remaining as a function of time and distance travelled. The results are shown in Figs. 9 and 10 respectively.

When the remaining entropy value is 100% the occupancy information is $p(c) = 0.5$ for all cells $c$ within the experimental area, and when the value is 0% the occupancy information is $p(c) = 0$ or $p(c) = 1$ for all cells, i.e. entropy is zero. The lines start at approximately 80% remaining entropy, after the robot observes the environment first at its starting pose. A dashed line in each subfigure indicates the performance of the human-operated reference run. The reference run lasted for 150 s and thus the reference line in Fig. 10 is flat after that time. Each of the other lines depicts the result of one of the five repetitions of an experiment. The lines indicate that the durations and lengths varied form experiment to experiment.

The performance of the planner improves significantly when applying multi-step optimization. As in the simulation experiment, a longer opti-
Figure 9: The percentage of entropy of a map of the whole area remaining as a function of the distance traveled, in all five experiments for each optimization horizon length $H$. The dashed line with no markers is the same in each axis, showing the reference result from manual teleoperation.

Figure 10: The percentage of entropy of a map of the whole area remaining as a function of time, in all five experiments for each optimization horizon length $H$. The dashed line with no markers is the same in each axis, showing the reference result from manual teleoperation.
mization horizon helps avoiding actions that seem informative at first but lead to poor information gain later. A typical example observed during the experiments was the robot turning to observe the area in that direction, while ignoring that traveling in the current heading would lead it to a new area with greater potential information gain later. This is seen e.g. from the two example trajectories for $H = 1$ and $H = 3$ shown in Fig. 7. The robot traveling along the dashed line for $H = 1$ turns left back towards the start location instead of proceeding further ahead, shortly after which the experimenter terminated the experiment as no progress was being made. In contrast, for $H = 3$ the algorithm chooses in a similar situation to travel further ahead as indicated by the dotted line, which leads it to a new area to explore on the right hand side of the map.

The performance as a function of the optimization horizon improved consistently, unlike in the simulation experiment. This is seen e.g. from Fig. 9 where longer horizons consistently lead to lower entropy at the same distance traveled. We believe this is primarily due to that in the experiment the sensor range is greater than the length of the trajectories considered by the planner. Thus, the trajectories are within the area currently observable by the robot’s laser range finder. This reduces the need to draw samples from uniform prior map information in so far unobserved areas which was noted in the simulation experiment as one reason for increasingly poorer performance with longer horizons.

Performance as a function of distance was similar to a human operator for all planning horizons up to the first 10 meters travelled. However, beyond this distance performance starts to differ such that $H = 5$ and $H = 7$ have significantly better performance. For $H = 3$, the planner does not consistently outperform the human operator. As a function of time, performance of the planner was closer to that of the human operator. As seen from Fig. 10 the human operator never performs consistently worse than the planner.

The human operator’s performance differed from the planner due to two main factors. First, the human could view the entire experimental area and decide beforehand on a route to travel. This enables the human to easily exceed the performance of a myopic planner. However, the operator’s primary target was to maximize a subjective, qualitative estimate of area coverage rather than quantitative information gain. The route the operator travelled covers the area well, but does not consider e.g. that a certain area could be observed as efficiently along a shorter route or that it might be useful to spend more time in some locations to observe them more accurately. The human operator might perform better by also viewing the current map
online during the teleoperation task. However, dividing attention between multiple tasks may itself degrade performance, or may not always be safe.

The results of the experiments show that the SMC planner can perform comparably to a human operator in a real-world environment and even exceed their performance when applying a long optimization horizon. The SMC planner’s way of sampling control sequences adaptively based on current map information makes it more flexible than approaches relying on a fixed discretization of the action space. Long optimization horizons are useful if there is a strong prior on map information combined with slow dynamics, or for a uniform prior if the sampled trajectories are such that they remain within the current sensor range.

8 Conclusion

In this article, we studied a robotic exploration problem defined as a partially observable Markov decision process (POMDP) with an information-theoretic reward function, the mutual information (MI). Our objective was to simultaneously handle uncertainty in both action effects and sensing outcomes in a dynamic scenario.

We proposed an open-loop approximation to solve the resulting optimal control problem, and combined it with the receding horizon control principle. The approximation is particularly useful in robotics, since the typically large or even uncountable space of observations complicates closed loop planning.

We solved the resulting open-loop control problem applying forward simulation methods. This approach can handle general, non-Gaussian belief states and dynamics and observation models. Two forward simulation algorithms were introduced. The first is suitable for a finite action space, and is based on Monte Carlo tree search. The second algorithm is a sequential Monte Carlo method, which is able to deal with an uncountable action space. Both methods have been shown to converge in probability to the optimal solution, and can be parallelised to decrease computation time.

The cumulative discounted sum of mutual information was considered as the objective function for exploration. We derived an approximation of MI assuming the state separates into an internal state and an environment state that the robot cannot affect. Both algorithms introduced apply the approximation to evaluate the objective function. Effectively, the approximation avoids the need to solve a SLAM problem while solving the control problem.

We demonstrated the utility of multi-step planning in a simulated do-
main and in a real-world experiment. Our results show that a multi-step optimization horizon performs significantly better in exploration tasks than a myopic one-step optimization horizon. Lower environment state entropy and greater total cumulative MI was achieved, as actions that seem to be useful now but lead to situations where there are few possibilities for further information gain are avoided. Minimizing state estimate entropy at end of the optimization horizon is also possible with our approach. Performance was found to be sensitive to the rate of dynamics in the environment state. Multi-step planning was most useful in case environment dynamics were slow and there was a strong prior on the environment state.

Challenges related to application of the algorithms presented include identifying suitable parameters for the exploration task considered. They are strongly affected by both the environment and the abilities of the robot platform and its sensors. We identified suitable parameters by balancing between solution quality and computation speed. Although our approach gains a computational advantage by ignoring the SLAM problem while solving the control problem, if the possibility of losing the consistency of the real process SLAM particle filter cannot be ignored [Blanco et al. 2008; Carlone et al. 2010], our algorithm may select actions that do lead to such a failure state. The SLAM algorithm would then need to be improved, or an alternative approximation of MI considered.

There are several possible directions for further research. In the SMC algorithm, the selection of the sampling kernel is an important choice [Del Moral et al. 2006]. Determining how the environment model and features and robot abilities should be taken into account in this choice remains an open question. As forward simulation methods are applicable to very general underlying models, it might be useful to consider more realistic environment models that lift the assumption of spatial independence between locations inherent to occupancy grid. Examples include e.g. higher order Markov random fields (MRFs), see e.g. Nabbe and Hebert (2007) for an application of forward simulation methods to path planning under a MRF environment model. We believe a more realistic environment model would improve prediction accuracy and further add to the usefulness of non-myopic planning. Applying such environment models requires overcoming the related increase in complexity of inference.
Acknowledgements

The authors thank Mr. Olli Suominen for helpful discussions and comments on the manuscript, and Mr. Joonas Melin for assistance in carrying out the experimental work.

Appendix A  Lazy map and observation sampling

Algorithm 3 presents a lazy sampling method for drawing observation samples. The sampling algorithm can be integrated with Algorithms 1 and 2. It works by constructing step-by-step the observations $z^{(i)}_{t+1:t+H}$ as functions $z^{(i)}_{t} : \tilde{W}(x^{(i)}_{t}) \rightarrow \{1, 0, -1\}$ which assign to each sensed cell one of three values, a hit (1), a miss (0), or indication that the cell was not observed (-1). We also define a persistent map sample $m^p : W \rightarrow \mathbb{N} \times \{0, 1\}$ as a mapping from cells to a pair of an integer $k$ denoting the time up to which the cell’s state has been propagated, and a binary-valued cell state, free (0) or occupied (1).

In a computer implementation, both observations and the persistent map sample can be efficiently represented as hash tables that are constructed and updated piece by piece. Initially, both are defined as empty.

A pose $x^{(i)}_{t}$ is sampled from $p(x_{t})$ obtained from $b_t = p(x_{t}, m_t)$ (Line 2). At each step $k$, a new pose $x^{(i)}_{k+1}$ is sampled from $p(x_{k+1} | x^{(i)}_{k}, u_k)$ (Line 4). The action $u_k$ is obtained either from the tree policy via UCT or randomly from the rollout policy for the POMCP algorithm 1. For the SMC algorithm 2, the action in the sequence of the current particle is applied.

A sensor-dependent method is used (Line 5) to sample an observation. For a laser range finder (LRF) which we applied in our experiments, the sample is drawn by applying the function $\text{raytrace}$. For each incidence angle $\alpha$ in which a LRF beam is sent from the current pose $x^{(i)}_{k+1}$, cells $c$ which are on the path of the LRF beam are found starting from the nearest. If the cell is not already in the persistent map, we set $m^p(c) = (k + 1, o_c)$, sampling $o_c$ from $p(c_{k+1})$ obtained by propagating initial information $p(c_t)$ appropriately via the state transition model (Line 13). If the cell is already in $m^p$, we propagate its state in $m^p$ to time $(k + 1)$ (Line 15).

Once the state of the map in the cell $c$ has been sampled at the current time step, we sample the observation for the particular cell as $z^{(i)}_{k+1}(c) \sim p(z_{k+1}(c) | o_c, x^{(i)}_{k})$ (Line 17). In the case of a LRF sensor, $z^{(i)}_{k+1}(c) = 1$ indicates that the beam hit an object and the ray tracing for the current
Algorithm 3 A lazy sampling scheme.

Parameters: The POMDP model, horizon $H$.

1: function $\text{SAMPLE}(b_t)$
2: Sample $x_t^{(i)} \sim p(x_t)$
3: for $k = t, \ldots, t + H - 1$ do
4: Sample $x_{k+1}^{(i)} \sim p(x_{k+1} \mid x_k^{(i)}, u_k)$
5: Sample $z_{k+1}^{(i)} \sim p(z_{k+1} \mid x_{k+1}^{(i)}, u_k)$ (e.g. $\text{RAYRACE}(x_{k+1}^{(i)})$)
6: Calculate $p(m_{k+1} \mid m_k, x_{k+1}^{(i)}, z_{k+1}^{(i)})$
7: end for
8: end function

9: function $\text{RAYRACE}(x_{k+1}^{(i)})$
10: for all incidence angles $\alpha$ do
11: for all cells $c$ along a beam starting at $x_k^{(i)}$ in direction $\alpha$ do
12: if $c \notin m^p$ then
13: Set $m^p(c) = (k + 1, o_c)$ with $o_c \sim p(c_{k+1})$
14: else
15: Propagate $m^p(c)$ to time $(k + 1)$ via $p(c_{t+1} \mid c_t)$
16: end if
17: Sample $z_{k+1}^{(i)}(c) \sim p(z_{k+1}(c) \mid m^p(c), x_k^{(i)})$
18: if $z_{k+1}^{(i)}(c) = 1$ then
19: break
20: end if
21: end for
22: end for
23: return $z_{k+1}^{(i)}$
24: end function
Finally, once the observation sample is returned, we can compute the posterior belief over the map $m_{k+1}$ (Line 6). At this point, it is also convenient to calculate quantities such as map entropy or mutual information.

References


