Artificial Intelligence 174 (2010) 704–725

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

# Planning to see: A hierarchical approach to planning visual actions on a robot using POMDPs

Mohan Sridharan [a,*], Jeremy Wyatt [b,1], Richard Dearden [b,2]

[a] Department of Computer Science, Texas Tech University, Lubbock, TX 79409, USA
[b] School of Computer Science, University of Birmingham, Birmingham, B152TT, UK

A R T I C L E   I N F O

*Article history:*
Received 23 April 2009
Received in revised form 23 April 2010
Accepted 23 April 2010
Available online 29 April 2010

*Keywords:*
Hierarchical POMDPs
Visual processing
Probabilistic planning
Robotics

A B S T R A C T

Flexible, general-purpose robots need to autonomously tailor their sensing and information processing to the task at hand. We pose this challenge as the task of planning under uncertainty. In our domain, the goal is to plan a sequence of visual operators to apply on regions of interest (ROIs) in images of a scene, so that a human and a robot can jointly manipulate and converse about objects on a tabletop. We pose visual processing management as an instance of probabilistic sequential decision making, and specifically as a Partially Observable Markov Decision Process (POMDP). The POMDP formulation uses models that quantitatively capture the unreliability of the operators and enable a robot to reason precisely about the trade-offs between plan reliability and plan execution time. Since planning in practical-sized POMDPs is intractable, we partially ameliorate this intractability for visual processing by defining a novel hierarchical POMDP based on the cognitive requirements of the corresponding planning task. We compare our hierarchical POMDP planning system (HiPPo) with a non-hierarchical POMDP formulation and the Continual Planning (CP) framework that handles uncertainty in a qualitative manner. We show empirically that HiPPo and CP outperform the naive application of all visual operators on all ROIs. The key result is that the POMDP methods produce more robust plans than CP or the naive visual processing. In summary, visual processing problems represent a challenging domain for planning techniques and our hierarchical POMDP-based approach for visual processing management opens up a promising new line of research.

## 1. Introduction

One significant challenge on the path towards widespread deployment of flexible general purpose robots is the lack of general purpose vision systems. This inability to exploit the rich information encoded in visual input is primarily due to the fact that work in machine vision has predominantly focused *not* on the design of visual systems, but on solutions to specific sub-problems such as object categorization or surface reconstruction. There is hence no convincing candidate theory that explains how to put together the pieces of a flexible and robust machine vision system. In the past, the goal of robot vision was seen by some as being the production of a general purpose world model that is suitable to support a variety of tasks [1]. This approach of creating a world model and querying it for specific tasks, has largely been abandoned due to the unreliability of visual operators and the infeasible computational load associated with producing such a model. Roboticists have instead built small special-purpose visual systems that make strong assumptions about the task and the environment

* Corresponding author. Tel.: +1 806 742 3527; fax: +1 806 742 3519.
*E-mail addresses:* mohan.sridharan@ttu.edu (M. Sridharan), jlw@cs.bham.ac.uk (J. Wyatt), rwd@cs.bham.ac.uk (R. Dearden).
[1] Tel.: +44 121 414 4788; fax: +44 121 414 4281.
[2] Tel.: +44 121 414 6687; fax: +44 121 414 4281.

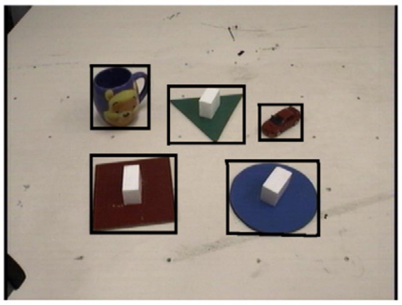


**Fig. 1.** Image of the typical tabletop scenario—salient regions of interest (ROIs) bounded by rectangular boxes.

[2–4]. This approach of creating task-specific vision systems has been advocated by some as *the* desirable way to build robot vision systems [5]. The downside is that the goal of creating general-purpose visual systems has been abandoned. Our work is a significant step towards answering the following question: *how can we create robot vision systems that are computationally lightweight, robust and flexible, all at the same time?*

One solution is to enable a robot to retain visual operators to support a variety of tasks, and choose a subset that is relevant to a given task. We call this approach of enabling a robot to autonomously tailor visual processing to the task at hand as *planning to see*. Such a formulation is influenced by the following features and requirements of robot vision systems:

- Features:
  - ▷ *Non-deterministic actions*: visual operators are unreliable.
  - ▷ *Partial observability*: the true state of the world is not directly observable. The robot can only update its *beliefs* about the state of the world by applying operators and observing the outcomes.
  - ▷ *Computationally intensive*: state of the art visual processing operators are often computationally expensive.
- Requirements:
  - ▷ *Performance*: robots interacting with a changing environment or humans need to respond in real-time.
  - ▷ *Reliability*: the visual system has to operate with a high degree of overall reliability despite the unreliability of individual visual operators.

These features and requirements suggest that visual processing on a robot should take into account the reliability of the operators, the accumulation of evidence from several operators to provide high overall reliability, and the computational complexity of the operators. An approach based on sequential decision making under uncertainty captures most of these features and requirements [6]. We therefore formulate *planning to see* as a probabilistic sequential decision making problem and enable a robot to simultaneously decide *where to look*, i.e., which region in the scene or image to focus on, and *what to look for*, i.e., what processing to perform on the chosen region.

## 1.1. Our domain

We study visual planning in a specific domain that captures many of the characteristics we would ultimately like to address. In our domain, a robot and human jointly converse about and manipulate objects on a tabletop [7]. Though seemingly simple, the domain represents the state of the art in cognitive robotics [8]. Several modules operating in parallel process the visual and speech inputs to create goals that are achieved by other modules such as manipulation. Typical visual processing tasks in this domain require the ability to find the color, shape, identity or category of objects in the scene to support dialogues about their properties; to see where to grasp an object; to plan an obstacle-free path to do so and then move it to a new location; to identify groups of objects and understand their spatial relations; and to recognize actions the human performs on the objects. Each of these tasks is a hard problem in itself, but we are faced with the formidable challenge of building a vision system capable of performing all of them.

Consider the scene in Fig. 1 with rectangular regions of interest (ROIs) extracted from the background. The robot has at its disposal a range of visual processing and sensing actions, in order to execute commands and answer a variety of queries about the scene: "is there a blue triangle in the scene?", "what is the color of the mug?", "move the mug to the right of the circle". However, in a complex scenario such as the one described above, it is neither feasible nor efficient to run all the operators on an image, especially since the robot has to respond dynamically to queries and commands.

## 1.2. Contributions

Though there exists a body of impressive work on planning of visual processing [9–11], AI planning [12] and probabilistic planning on robots [3,13,14], none accommodates all the features and requirements of the problem identified above. Most such methods are used for image analysis, require specialist domain knowledge to perform re-planning and plan repair, or have only been extended to robot systems in limited ways.

We push the field of planning of visual processing in a new direction by posing it as an instance of partially observable probabilistic sequential decision making. Specifically, we pose visual processing management as a partially observable Markov decision process (POMDP) [6] thereby incorporating quantitative models of the unreliability of visual operators. We represent the visual operators as actions that produce stochastic observations. Most actions considered in this paper gather information but do not change the actual contents of the visual scene, i.e., they do not change the state. However, in order to address scenes with partially overlapping or occluding objects (a common occurrence in practical scenes), we incorporate actions that change the perceived state of the system during planning. In other words, we demonstrate the ability of the proposed system to represent actions that change the state. The initial uncertainty about the scene is captured by the POMDP's belief state, a probability distribution over the underlying state space. A reward function is used to quantify the trade-off between processing time and reliability, by penalizing wrong answers and assigning costs to operators based on their run time complexity.

The POMDP formulation however quickly grows too large to be tractable, even with state of the art approximate solution methods. We therefore reduce the complexity of planning by defining a novel hierarchical POMDP that decomposes the visual planning problem into two sub-problems: choosing which region-of-interest (ROI) to perform processing on; and deciding what processing to perform on the chosen ROI. This decomposition is achieved by defining one POMDP at the higher level (HL) where actions consist of choosing the ROI to process, and one POMDP per ROI at the lower level (LL) for choosing the operators to apply on the corresponding ROI. The LL-POMDPs are treated as actions at the HL and return observations to the HL-POMDP. Many of the LL-POMDPs can also be cached and reused in different scenes, resulting in a significant speed up.

In addition to the POMDP formulation, we present a formulation based on the Continual Planning (CP) framework [15]. CP can be thought of as a contingency planning algorithm where only a single branch of the contingency tree is fully planned, and replanning is triggered if the execution finds itself on an unplanned branch. CP provides significant improvements over classical planning methods, and it has been implemented successfully on real robot domains [7]. We show empirically that incorporating either planning method significantly speeds up the performance in comparison to the naive approach of applying all visual operators on all ROIs, even if the planning times are taken into account. The key benefit of the hierarchical POMDP planning approach is that the plans, while taking slightly longer to generate and execute than those produced by CP, result in more reliable performance than CP or naive visual processing.

POMDP planning requires probabilistic models of action effects. We use learning with minimal human feedback to acquire these models offline. During online execution, the user-specified queries are converted into query-specific hierarchical POMDPs. An off-the-shelf POMDP solver [16] is used to solve these POMDPs to produce policies that provide the sequence of operators best suited to address a given query. Policies are executed without re-planning, except when an action changes the underlying state space, as described later in the paper. The goal is not to create a POMDP solver or improve the visual processing operators. Instead, the key consequence of the work described here is a hierarchical probabilistic planning scheme that automatically tailors visual processing to the task at hand.

### 1.3. Structure of the paper

The remainder of the paper is organized as follows. Section 2 briefly reviews a representative set of modern planning methods, including those designed specifically for visual processing. We then pose planning of visual processing in two ways: as a probabilistic planning problem (Section 3.1), and as a continual planning problem (Section 3.4). We show how visual operators can be modeled using planning operators in each framework. Next, we define our hierarchical planning system (HiPPo) that makes the POMDP formulation more tractable (Section 3.3), and differentiate it from other approaches that impose a hierarchy in POMDP formulations (Section 3.2). We then present detailed experimental results in our test domain (Section 4) and conclude with a discussion of future research directions (Section 5).

## 2. Related work

AI planning and cognitive planning architectures have been extensively researched [12,17–19]. Most classical planning methods however require prior knowledge of states, action outcomes, and all contingencies [12], which cannot be provided for robot systems that interact with the real-world through noisy sensors and effectors. The focus here is on a specific sub-category of the planning problem: the joint planning of sensing (where to look) and information processing (what to look for) actions to achieve a desired goal. Here, we review a set of methods for planning of visual processing, including general planning algorithms. These methods are evaluated in terms of their ability to satisfy the requirements of the visual planning task, as described in Section 1.

### 2.1. Visual planning

There is a significant body of work in the image processing community on planning sequences of visual operations [9,20–23]. These methods typically use a classical AI planner that takes a user-specified high-level goal and constructs a pipeline of image processing operations. The planners use deterministic models of information processing, handling operator preconditions and effects using propositions that are required to be true a priori or are made true by the application of the operator. Unsatisfactory results are detected by evaluating output images using hand-crafted rules [9,10,23] and addressed by re-planning the operator sequence or modifying the operators' parameters [10,23].

The representation of the actions (i.e., the image processing operators) has received considerable attention. One approach is to use a modeling language similar to STRIPS operators [24], where the preconditions and effects of operators are pre-specified [10]. The rules mainly decide the legal sequences of image processing operators—there is typically little or no attempt to predict the non-deterministic characteristics of output images because it is very difficult to estimate the likely output of an operator [9]. Another challenge is to create a domain-specific set of rules for evaluating the output of each operator, a process that requires expert knowledge. These planning and program supervision methods have been used for automating image processing in fields such as astronomy [10,20] and medicine [9]. There has also been work on perception for autonomous object avoidance in vehicles [25], and interpretation of 3D object structure [26]. Extensions to general computer vision tasks have however proven difficult.

In the field of computer vision, probabilistic sequential decision processes, i.e., Markov Decision Processes (MDPs) and Partially Observable MDPs (POMDPs), have been used for image interpretation. Darrell use memory-based reinforcement learning and POMDPs to learn to foveate salient body parts in an active gesture recognition system [27]. The problem is formulated as a POMDP using an action set consisting of foveation actions and a special recognition action. During the learning phase, execution of the recognition action is followed by manual feedback on the target object's presence in the scene, so that each action sequence can be assigned a reward. Instance-based hidden-state reinforcement learning is used to learn what foveation actions to execute, and when to execute the terminal recognition action. Since the learned recognition policies can consume significant memory, the action-selection policy is transformed into an augmented Finite State Machine for online recognition. Such approaches that require manual feedback to generate the reward signal on each trial, are too time consuming to use on a robot operating in a dynamic domain. It would instead be desirable to autonomously generate the models and policies.

More recently, Li et al. [11] posed image interpretation as an MDP. All possible sequences of image operators are applied on human-annotated images in an offline process to determine the reward structure. The states are abstracted into a set of image features. Dynamic programming methods are used to determine the value function for the explored parts of the state space, which is then extrapolated to the entire state space using an ensemble learning technique. Online execution consists of repeatedly extracting image features and choosing an action that maximizes the learned value functions. Similar approaches using MDP-based approximations such as the most-likely-state MDP [28] have been proposed for tasks such as robot navigation [29]. However, such an approximation is not suitable for information gathering actions that do not change the physical state.

Sequential decision processes have also been used for gaze planning, i.e., to plan a sequence of gaze locations (image ROIs) to analyze in order to identify the desired target. In recent work, Vogel and de Freitas [30] have posed gaze sequence selection as a finite-horizon sequential decision process that elegantly combines bottom–up saliency, top–down target knowledge and spatial target context. The proposed gaze planning strategy was tested on image databases to determine the location of computer monitors. Open-loop feedback control was used for re-planning based on observations. Though a prior distribution over object locations is difficult to compute for multiple objects in practical scenes, this method can be incorporated in our POMDP framework as a processing routine for tasks such as viewpoint planning.

In addition to computer vision tasks, POMDP-based approaches have been utilized for tasks such as preference elicitation, i.e., to infer user preferences through repeated interactions with the system [31,32]. Though similar in some aspects to the visual processing management task considered here, preference elicitation assumes that user preferences do not change as a result of the interaction, i.e., that actions do not change the state of the system: this assumption is not valid in human–robot interaction scenarios.

A POMDP can be solved by ignoring the observation model and finding the $Q$-function for the underlying MDP. This QMDP approach yields a set of action values that can be weighted by the belief to estimate the action values in the belief states [33]. However, this method assumes that the uncertainty in the underlying state disappears in one step. QMDP policies will not take actions to gain information and are hence poorly suited for domains where the information gain of actions is the source of reward.

There has been considerable work in the related field of active sensing, where sensor placement and information processing are decided based on the task at hand [34,35]. Kreucher et al. [34] presented an active sensing approach that combines particle filtering, predictive density estimation and relative entropy maximization for scheduling sensors in order to learn the number and states of a group of moving targets. The particle filter estimates the joint multitarget probability density. A sensing action is chosen based on the Renyi divergence entropy measure, following which the system updates the probability density on the number and states of the targets. Though estimating the joint probability density requires considerable prior information, this approach can be used to determine the region in space where our planning approach should focus during a surveillance task.

Recently, submodular functions have been used for sensor placements in spatial phenomena modeled as Gaussian processes [36,37]. In domains where the objective function can be represented using submodular functions, the greedy policy provides performance that is at least 63% of optimal performance, thereby providing a strong lower bound [36]. Krause and Guestrin have shown that many observation planning problems are submodular, and can therefore be efficiently solved approximately [37]. A problem is submodular if it exhibits diminishing returns—each additional observation is worth less than the previous one. Unfortunately, this does not apply in our domain. Consider the query "where is the red circle?" If we run the color algorithm on an ROI and it returns the value *red*, the expected value of running the shape detection algorithm on the same ROI is higher than it was previously. Hence the problem is not submodular. In general, submodularity applies in domains where the value is in making the observation regardless of its result. In our domain, the eventual reward received depends on the result of the observation so submodularity is not present.

Our POMDP-based approach does have similarities with the existing schemes for visual planning in that we are computing a sequence of operations for a specific task. The difference lies in the fact that our approach encapsulates the features and requirements of the visual planning task in robot domains (Section 1). We automate belief propagation and the computation of the associated models, thereby making it easier to add more actions and address a range of queries in the domain.

### 2.2. Observation planners

Several modern planning schemes relax some of the constraints of classical planning methods in order to apply the machinery of classical planning to practical tasks, and we review a few methods below.

Draper et al. proposed C-BURIDAN, a planning scheme that incorporates a probabilistic model of the noisy sensors and effectors, while retaining a symbolic STRIPS-like representation of action effects [38]. The plan-assessment phase treats actions as probabilistic state transitions, while the plan-refinement phase links the symbolic action effects to the symbolic sub-goals of the desired goal state. Similar to our POMDP-based approach, they reason about the best action to perform based on prior belief about the world and the observations made by executing actions. However, their approach: requires the preconditions and effects of actions to be manually specified; does not incorporate action costs; and requires a manual ordering of actions to accumulate belief from repeated execution of the same action.

In contrast to the C-BURIDAN system, Petrick and Bacchus's PKS planner describes actions in a first order language, in terms of their effect on the agent's knowledge rather than their effect on the world [39]. The model is hence non-deterministic in the sense that the true state of the world may be determined uniquely by the actions performed, but the agent's knowledge of that state is not. For example, dropping a fragile item will break it, but if the agent does not know that the item is fragile, it must use an observational action to determine its status. PKS captures the initial state uncertainty and constructs conditional plans based on the agent's knowledge. In our domain, we can say that the target object is in one of the ROIs, but that we do not know which one. The planner will then plan to use the observational actions to examine each ROI, branching based on what is discovered.

More recently, Brenner and Nebel proposed the Continual Planning (CP) approach, which interleaves planning, plan execution and plan monitoring [40]. Unlike classical planning schemes that require prior knowledge of state, action outcomes and all contingencies, an agent in CP postpones reasoning about uncertain states until more information is available. The agent allows each action to assert that its preconditions will be met when the agent reaches that point in the execution of the plan. If these preconditions are not met during execution, or are met earlier, replanning is triggered. CP is therefore quite similar to PKS in representation but works by replanning rather than constructing conditional plans. There is however *no* representation of the uncertainty in the observations and actions. CP is based on the FF planner [41] and uses the PDDL syntax introduced by McDermott [42]. Section 3.4 describes how our domain can be modeled in the CP framework.

In applications with noisy observations, the optimal behavior may be to increase confidence in the image interpretation by accumulating the evidence obtained by running the operators more than once on several images of the scene. None of the approaches mentioned in this section can readily represent this belief accumulation.

## 3. Probabilistic problem formulation

As mentioned in Section 1, visual processing on robots is characterized by partial observability and non-determinism. We capture these features by posing the problem as an instance of probabilistic sequential decision making. Specifically, we pose the planning of visual sensing and information processing as a partially observable Markov decision process (POMDP) and explicitly model the unreliability of the visual operators. This formulation enables the robot to maintain a probability distribution over the true underlying state: the *belief state*. The belief state maintenance in turn requires an observation model that predicts the likelihood of all possible action outcomes. The robot can learn this observation model, as described in Section 4.1. Our POMDP formulation models the effects of actions that gather information and actions that change the system state. This ability to model actions that change the state is essential for achieving a long-term goal of our work: to enable a robot to interleave perception with environmental object interactions (e.g., pushing an object) in order to obtain new views.

For ease of understanding, we use the example of an input image from the tabletop scenario that is pre-processed to yield two regions of interest (ROIs), i.e., two salient image regions that are different from the immediate background—see rectangular bounding boxes in Fig. 2. Fig. 1 shows additional examples of ROIs extracted from the background.

Consider the query: "which objects in the scene are blue?" Without loss of generality, assume that the robot has the following visual operators at its disposal: a *color* operator that classifies the dominant color of the ROI it is applied on; a *shape* operator that classifies the dominant shape within the ROI; a *sift* operator that uses SIFT features [43] to detect objects whose feature models have been trained; and a set of region-split operators that split an ROI into smaller regions based on each of the operators listed above: $rSplit_{color}$, $rSplit_{shape}$ and $rSplit_{sift}$. The *color* operator characterizes ROIs based

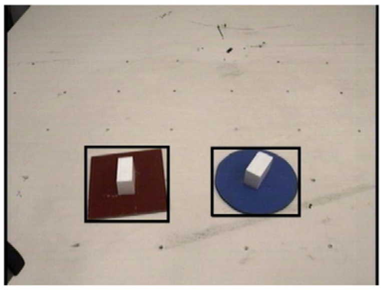


**Fig. 2.** Image of a tabletop scenario with two objects that result in two ROIs.

on color-space histograms, while the *shape* operator characterizes the dominant contour within the ROI using invariant moments. The *sift* operator characterizes target objects with local image gradients that are robust to scale, orientation and viewpoint changes. The region-split operators split the ROIs and change the *perceived* state of the system during planning, and are used to tackle scenes with partially overlapping objects. The goal is to plan a sequence of operators that would answer queries with high confidence. Throughout this paper, we use the following terms interchangeably: visual processing actions, visual actions and visual operators.

### 3.1. POMDP formulation

As mentioned in Section 1.2, our hierarchical POMDP planning system (HiPPo) consists of two levels. Given a specific query about a scene represented by image ROIs, the POMDP at the higher level chooses the ROI to process next. One POMDP per ROI at the lower level chooses the sequence of operators to apply on the ROI in order to reliably predict its contents. We begin with the POMDP framework at the lower level.

The POMDP framework at the lower level requires a suitable representation of action outcomes. Each action considers the true underlying state to be composed of the class labels (e.g., *red*($R$), *green*($G$), *blue*($B$) for color; *circle*($C$), *triangle*($T$), *square*($S$) for shape; *picture, mug, box* for sift); a label to denote the absence of any valid object—*empty* ($\phi$); and a label to denote the presence of *multiple* classes ($M$). The corresponding observation function is a probability distribution over the set of possible action outcomes. The set of action outcomes consists of the class labels; the label *empty* ($\phi$) which implies that the match probability corresponding to the class labels is very low; and *unknown* ($U$) which implies that multiple classes are equally likely and the ROI may therefore contain multiple objects. While $U$ is an observation, $M$ is part of the underlying state: they are not the same since they are not perfectly correlated. In the formal definition, we make this distinction clear for the other class and observation variables.

Since visual operators only update belief states, we include "special actions" that cause a transition to a terminal state where no further actions are applied, i.e., these query-specific actions terminate processing to answer the query. The answer could report or "say" (not to be confused with language-based communication) which underlying state is most likely to be the true state, or it could simply state the presence or absence of the target object. In the description below, without loss of generality and for ease of explanation, we only consider two operators: *color* and *shape*, each of which provides three class labels. The operators are denoted with the subscripts $c$, $s$ respectively. The approach generalizes to *sift*, other vision algorithms and more outcomes. True states and observations are distinguished by the superscripts $a$, $o$ respectively.

Consider a single ROI in the scene. The POMDP that answers specific queries about the ROI is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{Z}, \mathcal{O}, \mathcal{R} \rangle$:

- $\mathcal{S} : \mathcal{S}_c \times \mathcal{S}_s \cup term$ is the set of states. It is a Cartesian product of the variables describing different aspects of the underlying state (e.g., color, shape) and includes a *terminal* state (*term*). $\mathcal{S}_c : \{\phi_c^a, R_c^a, G_c^a, B_c^a, M_c^a\}$, $\mathcal{S}_s : \{\phi_s^a, C_s^a, T_s^a, S_s^a, M_s^a\}$. The subscript denotes the aspect of the state being modeled, and the superscript denotes that it is the *actual* state.
- $\mathcal{A} : \{color, shape, \mathcal{A}_{sp}\}$ is the set of actions. The first two entries are the visual processing actions. The rest are query-specific special actions that represent responses to the queries. For a query such as "is there a circle in the scene?" $\mathcal{A}_{sp} = \{sFound, sNotFound\}$ describes the presence or absence of the target object, while a query such as "what is the color of the ROI?" would lead to $\mathcal{A}_{sp} = \{sRed, sGreen, sBlue\}$, i.e., actions such as "say blue". All the special actions lead to *term*.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ represents the state transition function. It is an identity matrix for visual processing actions such as *color* and *shape* whose application does not change the underlying state of the world. For special actions, it represents a transition to *term*. For actions that change the state, the transition function needs to be suitably modified—an example is described later in this section.
- $\mathcal{Z} : \{\phi_c^o, R_c^o, G_c^o, B_c^o, U_c^o, \phi_s^o, C_s^o, T_s^o, S_s^o, U_s^o\}$ is the set of observations, a union of the observations of each visual action, i.e., $\mathcal{Z} = \mathcal{Z}_c \cup \mathcal{Z}_s$. The use of the superscript $o$ denotes an observation.
- $\mathcal{O} : \mathcal{S} \times \mathcal{A} \times \mathcal{Z} \to [0, 1]$ is the observation function, a matrix of size $|\mathcal{S}| \times |\mathcal{Z}|$ for each action. It is learned offline by the robot for each visual action (Section 4.1) and it is a uniform distribution for the special actions.

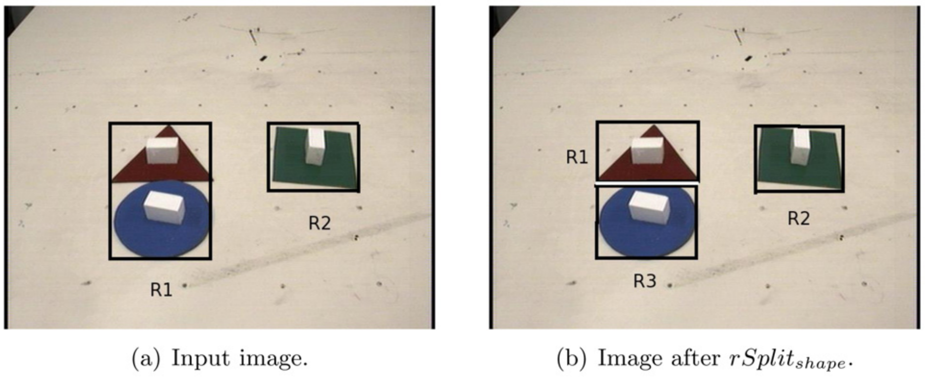


**Fig. 3.** Example of splitting an input ROI to analyze overlapping objects.

- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ specifies the reward, mapping from the state-action space to real numbers. In our case:

$$\begin{aligned}
&\forall s \in \mathcal{S}, \quad \mathcal{R}(s, \mathit{shape}) = -1.25 \cdot f_s(\text{ROI-size}) \\
&\mathcal{R}(s, \mathit{color}) = -2.5 \cdot f_c(\text{ROI-size}) \\
&\mathcal{R}(s, \text{special actions}) = \pm 100 \cdot \alpha
\end{aligned} \tag{1}$$

The cost for visual actions depends on the relative computational complexity of the operator and the size of the ROI. For instance, the *color* operator is twice as costly as *shape*, and this information is used to experimentally assign a cost factor such that the least expensive operator is assigned a value close to one. The dependence on ROI-size is captured using a polynomial function—see Eq. (8) in Section 4.1. For special actions, a large positive (negative) reward is assigned for making a correct (incorrect) decision for a given query. For instance, for the query "what is the color of the ROI?": $\mathcal{R}(R_c^a T_s^a, \text{sRed}) = 100 \cdot \alpha$ and $\mathcal{R}(B_c^a T_s^a, \text{sGreen}) = -100 \cdot \alpha$, i.e., we receive a reward of $100 \cdot \alpha$ for detecting the color correctly and a punishment for answering incorrectly. On the other hand, for the query "is there a red object in the scene?": $\mathcal{R}(R_c^a T_s^a, \text{sFound}) = 100 \cdot \alpha$ and $\mathcal{R}(B_c^a T_s^a, \text{sFound}) = -100 \cdot \alpha$. The variable $\alpha$ trades-off the computational costs against the reliability of query responses. When $\alpha$ is large, the special action is taken after executing a larger number of visual operators, resulting in higher reliability.

In the POMDP formulation, given the probability distribution over the underlying state at time $t$: the belief state $b_t$, the belief update proceeds as:

$$b_{t+1}(s_{t+1}) = \frac{\mathcal{O}(s_{t+1}, a_t, o_{t+1}) \sum_{s \in \mathcal{S}} \mathcal{T}(s, a_t, s_{t+1}) \cdot b_t(s)}{P(o_{t+1}|a_t, b_t)} \tag{2}$$

where $P(o_{t+1}|a_t, b_t) = \sum_{s' \in \mathcal{S}} \{P(o_{t+1}|s', a_t) \cdot \sum_{s \in S} P(s'|a_t, s)\, b_t(s)\}$ is the normalizer, $\mathcal{O}(s_{t+1}, a_t, o_{t+1}) = P(o_{t+1}|s_{t+1}, a_t)$, $b_t(s) = P(s_t = s)$, and $\mathcal{T}(s, a_t, s_{t+1}) = P(s_{t+1}|a_t, s_t = s)$. Our visual planning task for a single ROI now involves solving this POMDP to find a policy that maximizes reward over a range of belief states. Plan execution corresponds to using the policy to traverse a specific path of a *policy tree*, repeatedly choosing the action with the highest value at the current belief state, and updating the belief state after executing that action and receiving an observation. Fig. 4 shows a policy tree pictorially: it represents the result of applying all possible visual operator sequences on an ROI. In order to ensure that the observations are conditionally independent of each other given different snapshots of the same scene from the same view, a new image of the scene is taken if an action is to be repeated on an image ROI. This independence assumption is required for the belief update in Eq. (2). Though the images are not strictly independent, the assumption works in practice to account for small changes in environmental factors such as illumination. In situations where this independence assumption is violated, the POMDP will need to be reformulated before the appropriate models can be learned.

In most real-world scenes, overlap between objects is a common occurrence as a result of occlusion or camera viewpoint. Two or more objects can hence be contained in a single ROI. An ROI may also contain an object with multiple feature values (e.g., multiple colors). Such ROIs need to be split into smaller ROIs for analysis, based on one or more features of the ROI such as color or shape. Fig. 3(a) shows an example of an ROI containing two objects ($R_1$), which can be split into two ROIs based on the shape feature to result in ROIs $R_1$ and $R_3$ in Fig. 3(b).

Such region-splitting actions pose a challenge because they modify the perceived state space of the POMDP. The new state space has additional state variables for the new ROIs that have been created. It is not computationally feasible to plan through a split in advance, since each application of a split operator will lead to several other POMDPs. Each of these POMDPs must be solved, but crucial information such as the size of the new ROIs will not be known in advance. It is also not easy to estimate the transition probabilities for the new POMDPs. Our approach uses an analytic model that approximates the likelihood of transitioning to each new POMDP, and the value to be garnered from each such transition.

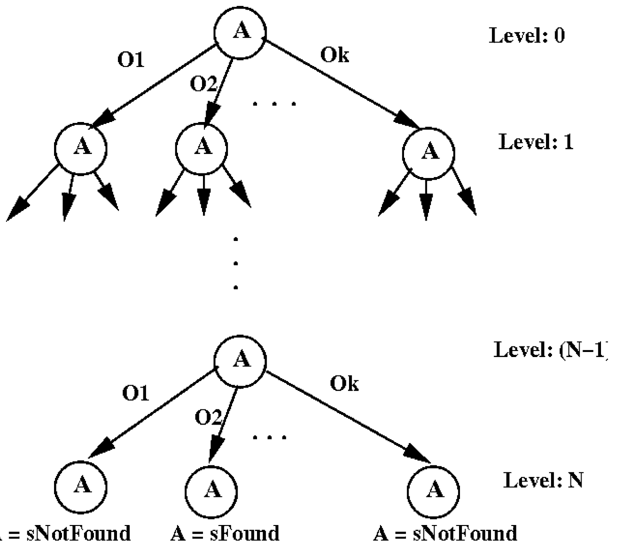


**Fig. 4.** Policy Tree for processing of an ROI—each node represents a belief state and specifies the action to take. All leaf nodes specify a terminal action.

Our strategy is explained using the example of the region-splitting action that segments the input ROI into one or more ROIs based on color: $rSplit_{color}$. If the true state of the ROI consists of just one color, $rSplit_{color}$ should not change the state. On the other hand, if there is more than one color in the ROI, we make the optimistic assumption that the action $rSplit_{color}$ will isolate the color we are looking for in the ROI (*blue* for the query: "which objects in the scene are blue?"). In addition, the execution of an *rSplit* operator is accompanied by an application of the operator corresponding to the feature used to perform the split, i.e., the execution of $rSplit_{color}$ is followed by the application of *color* on each resultant ROI. The $rSplit_{color}$ operator can hence be characterized as:

- The number of ROIs resulting from a split action is assumed to follow a geometric distribution. The maximum number of possible ROIs is equal to the number of labels provided by the underlying feature (five for *color*).
- The operator cost is the sum of three values. The first is the cost of performing $rSplit_{color}$, i.e., segmenting the ROI based on color. The second is the cost of re-applying the color operator on the expected number ($d$) of ROIs created by the split. Each of the $d$ ROIs is only $1/d$ the size of the ROI being split: for linear action costs, this cost is equivalent to applying the action once on the original ROI. The third value is the cost of solving the POMDPs corresponding to the new ROIs.
- The observation function ($\mathcal{O}$) is the same as that of the underlying operator (*color*). It is used to perform the belief update on each ROI resulting from the execution of $rSplit_{color}$.
- The transition function ($\mathcal{T}$) needs to ensure that the split operation is feasible only in the state where it is justified, i.e., $M_c^a$. We assume that applying the split operator in $M_c^a$ results in well-separated ROIs, i.e., we cannot transition back to $M_c^a$ after the split. Assume that the feature being split on has $n$ possible labels. Each of the ROIs created by the split can have a label relevant to the query with probability $\frac{1}{n}$. The geometric distribution sets the probability of producing $i$ ROIs after the split to $1/2^{(i-1)}$, assuming that a split action results in at least two ROIs. The special case of $i = n$ occurs with probability: $1/2^{(n-2)}$ so that the geometric distribution-based probabilities sum to one. The expected probability that one of the ROIs has the appropriate label, is then given by:

$$p = \sum_{i=2}^{n-1} \frac{1}{2^{i-1}} \frac{i}{n} + \frac{1}{2^{n-2}} \tag{3}$$

  The transition is hence as follows: if (and only if) the underlying state has multiple colors (i.e., $M_c^a$), with probability $p$ we move to a state where one ROI has the relevant label, and with probability $(1 - p)$ we move to one of the states where all of the ROIs have some other label. For the current example of searching for *blue* objects, Table 1 shows $\mathcal{T}$ for $rSplit_{color}$ while considering just the *color* states.

This formulation of region-splitting allows us to plan the effects of the split operators in the current POMDP model even though a split will increase the number of ROIs and change the model. We are replacing the solution of several new POMDPs with one in which we compute the likelihood that one of the resultant ROIs has the desired feature value. During execution, we will have to replan with a new POMDP after a split action (see Section 4.2).

**Table 1**
Transition function for $rSplit_{color}$ while considering just the *color* states.

| $p(init \mid fin)$ | $\phi_c^a$ | $R_c^a$ | $G_c^a$ | $B_c^a$ | $M_c^a$ |
|---|---|---|---|---|---|
| $\phi_c^a$ | 1.0 | 0 | 0 | 0 | 0 |
| $R_c^a$ | 0 | 1.0 | 0 | 0 | 0 |
| $G_c^a$ | 0 | 0 | 1.0 | 0 | 0 |
| $B_c^a$ | 0 | 0 | 0 | 1.0 | 0 |
| $M_c^a$ | $\frac{1-p}{3}$ | $\frac{1-p}{3}$ | $\frac{1-p}{3}$ | $p$ | 0 |

When we generalize this POMDP formulation to scenes with multiple ROIs, the state space grows exponentially. For a single ROI with $m$ features (e.g., color, shape) each with $n$ values (e.g., $R_c^a$, $G_c^a$, $B_c^a$, $\phi_c^a$ and $M_c^a$ for color), the POMDP has an underlying space of size $n^m + 1$. Actual scenes will contain several objects and hence several ROIs—for $k$ ROIs we have $n^{mk} + 1$ states. Solving a POMDP in the joint space of all ROIs soon becomes intractable even for a small set of ROIs and actions—with three ROIs and two actions we get $\approx 15\,000$ states. The rest of this section focuses on possible solutions to this problem.

## 3.2. Prior work on leveraging structure

Our visual processing problem when modeled as a POMDP has a state space whose dimensionality rises exponentially in the product of the number of features and the number of ROIs. Since the worst case time complexity of POMDPs is exponential in the number of dimensions in the state space, our problem becomes particularly formidable. In this paper, we propose a hierarchical decomposition to partially ameliorate this intractability. However, we first review recent approaches that exploit the structure in POMDP formulations in order to make such formulations more tractable.

Pineau et al. [3,44] proposed a hierarchical POMDP approach for high-level behavior control on a robot, similar to the MAXQ decomposition for MDPs [45]. They imposed an action hierarchy, with the top level action being a collection of simpler actions represented by smaller POMDPs. The planning algorithm operates in a bottom–up manner such that complete solutions for the smaller POMDPs are combined to provide the policy for the entire problem. The execution proceeds in a top–down manner: invoking the top-level policy traverses the hierarchy invoking a sequence of local policies until a primitive action is reached. Model parameters at all levels are defined over the same space of states, actions and observations, but the relevant space is abstracted for each POMDP using a dynamic belief network. Similar systems have been proposed for autonomous robot navigation [13,14,46]. In the actual application, however, a significant amount of data for the hierarchy and model creation has to be hand-coded.

Hansen et al. [47] proposed a programmer-defined Task Hierarchy (TH) for planning with POMDPs, where the policies are defined as finite-state controllers (FSCs). It is similar to Pineau's work in terms of the hierarchy, with exact dynamic programming (DP) solutions for each sub-problem and a bottom–up traversal of the TH. The DP policy of a sub-problem is treated as an abstract action in the next higher level POMDP. The difference lies in the fact that each policy is represented as an FSC to build exact models for each abstract action. In addition, each POMDP in the hierarchy is an indefinite-horizon POMDP in order to allow FSC termination without recognition of the underlying terminal state. Furthermore, policy iteration is used instead of value iteration to solve POMDPs. As a result these changes, the representation provides guarantees on policy quality.

There has been considerable work on exploring representations for hierarchical POMDPs that allow for tractable performance in practical applications. Theocharous et al. [48] represented hierarchical POMDPs as dynamic Bayesian networks (DBNs) for the specific task of using multi-resolution spatial maps for indoor robot navigation. They have shown that the DBN representation can train faster and with fewer samples than the hierarchical POMDP or the joint POMDP, and that their formulation extends to factoring of the underlying variables. More recent work by Toussaint et al. [49] aims to learn the hierarchical representation of a POMDP based on maximum likelihood estimation. It uses a mixture of DBNs and parameter estimation based on Expectation-Maximization.

The proposed hierarchical POMDP approach has some similarities with the existing approaches for hierarchical decomposition in POMDPs. Similar to [13,44], the model parameters at the higher level of our proposed hierarchy are functions of the policies computed at the lower level. The key difference lies in the fact that we do not require manual encoding of domain knowledge to abstract the relevant state space at different levels. The automatic belief propagation is achieved by having the different levels of our hierarchy operate over different state-action spaces that are based on a *functional* decomposition. Images of specific regions in space, and the corresponding image ROIs, are considered at different levels. Analyzing additional operators or image ROIs is hence significantly easier. In addition, the lessons learned from the hierarchical decomposition of the specific visual processing domain are applicable to other domains. Furthermore, the proposed hierarchy enables an automatic trade-off between reliability and efficiency (see Appendix A), which is very useful in robot applications.

## 3.3. Hierarchical POMDP planning: HiPPo

In the proposed *hierarchical decomposition*, each ROI is modeled with a lower-level (LL) POMDP as described in Section 3.1, and a higher-level (HL) POMDP chooses, at each step, the ROI whose policy is to be executed. This decomposes the overall problem into one POMDP with state space $2^k + 1$ and $k$ POMDPs with state space $n^m + 1$ (as before, $m$ is the number of features, each with $n$ values). The problem of what information to process is hence separated from how to process it. For the example of two ROIs and the query: "which objects in the scene are blue?", two LL-POMDPs are defined as in Section 3.1 and the HL-POMDP is given by $\langle \mathcal{S}^H, \mathcal{A}^H, \mathcal{T}^H, \mathcal{Z}^H, \mathcal{O}^H, \mathcal{R}^H \rangle$:

- $\mathcal{S}^H = \{R_1 \wedge \neg R_2, \neg R_1 \wedge R_2, \neg R_1 \wedge \neg R_2, R_1 \wedge R_2\} \cup \mathit{term}^H$ is the set of states. It represents the presence or absence of an object satisfying the query in one or more of the ROIs: $R_1 \wedge \neg R_2$ means the object exists in ROI $R_1$ but not in $R_2$. It also includes a terminal state: $\mathit{term}^H$.
- $\mathcal{A}^H = \{u_1, u_2, \mathcal{A}^H_{sp}\}$ are the actions. The sensing actions ($u_i$) denote the choice of executing one of the LL ROIs' policy trees. Similar to the LL-POMDPs, the special actions ($\mathcal{A}^H_{sp}$) are query-specific. For queries such as "is there a circle in the scene?", $\mathcal{A}^H_{sp} = \{\mathit{sFound}^H, \mathit{sNotFound}^H\}$. However, for queries such as "where is the blue circle?", $\mathcal{A}^H_{sp}$ identifies an entry of $\mathcal{S}^H$ as the answer. All special actions lead to $\mathit{term}^H$.
- $\mathcal{T}^H$ is the state transition function, which leads to $\mathit{term}^H$ for special actions and is an identity matrix for other actions.
- $\mathcal{Z}^H = \{FR_1, \neg FR_1, FR_2, \neg FR_2\}$ is the set of observations. It represents the observation of finding or not-finding the target object when each ROI's policy is executed.
- $\mathcal{O}^H : \mathcal{S}^H \times \mathcal{A}^H \times Z^H \to [0,1]$ is the observation function, which is a uniform matrix for special actions. For sensing actions, it is obtained from the policies of the LL-POMDPs as described below.
- $\mathcal{R}^H$ is the reward specification. For each sensing action, it is the "cost" of running the policy of the corresponding LL-POMDP, computed as described below. For a special action, it is a large positive (negative) value if it predicts the true underlying state correctly (incorrectly): $\mathcal{R}(R_1 \wedge R_2, sR_1 \wedge R_2) = 100$ and $\mathcal{R}(R_1 \wedge \neg R_2, sR_1 \wedge R_2) = -100$.

An important aspect of our hierarchical formulation is the relationship between the HL-POMDP and the LL-POMDPs. The observation function and reward specification for each HL sensing action are defined as a function of the corresponding LL-POMDP's policy. As seen in Fig. 4, root node of the LL-POMDP's policy tree represents the initial belief. At each node, the LL-POMDP's policy is used to determine the best action, and all possible observations are considered to determine the resultant beliefs and hence populate the next level of the tree.

Consider the computation of $FR_1$, i.e., the probability that the target object is "found" in $R_1$ on executing the LL-POMDP's policy. The probability of ending up in a leaf node corresponding to the desired terminal action (*sFound*) is computed by charting a path from each such leaf node to the root node and computing the product of the corresponding transition probabilities (the edges of the tree). These individual probabilities are summed up to obtain the total probability of obtaining the desired outcome in the HL-POMDP. While looking for *blue* objects ("which objects in the scene are blue?") let $\mathcal{L}$ be the set of leaf nodes in $R_1$'s LL-POMDP policy tree $\pi_1$ with the chosen action of *sFound*. Then:

$$P(FR_1) = \sum_{i_n \in \mathcal{L}} P_{\pi_1}(i_n)$$
$$P_{\pi_1}(i_n) = \prod_{k=n}^{1} P\big(i_k | \mathit{Parent}(i)_{k-1}\big) \tag{4}$$

where $i_k$ denotes the node $i$ at level $k$, $\mathit{Parent}(i)_{k-1}$ is the parent of node $i$ at level $k-1$ and $\pi_1$ is the policy tree corresponding to the ROI $R_1$. The entries within the product term in Eq. (4) are the normalizers of the belief update in Eq. (2):

$$P\big(i_k | \mathit{Parent}(i)_{k-1}\big) = \sum_{s' \in \mathcal{S}} \Big\{ P\big(o^{i_k}_{\mathit{Parent}(i)_{k-1}} | s', a_{\mathit{Parent}(i)_{k-1}}\big) \cdot \sum_{s \in \mathcal{S}} P\big(s'|a, s\big)\, b_{\mathit{Parent}(i)_{k-1}}(s) \Big\} \tag{5}$$

where $o^{i_k}_{\mathit{Parent}(i)_{k-1}}$ is the observation that transitions to node $i_k$ from its parent $\mathit{Parent}(i)_{k-1}$, while $a_{\mathit{Parent}(i)_{k-1}}$ is the action taken from the parent node and $b_{\mathit{Parent}(i)_{k-1}}(s)$ is the belief state at $\mathit{Parent}(i)_{k-1}$. The other entries of the HL observation function are computed by parsing the corresponding LL policy trees with appropriate initial beliefs.

During execution, we restrict the action choice to the set of terminal actions after $N$ steps in order to speed up plan execution on the robot. It is possible to pose the problem as a finite-horizon POMDP by including the time step constraint in the state description. However, we use an off-the-shelf infinite-horizon POMDP solver that is efficient [16] but only provides the belief space discretization as a control parameter, i.e., the LL policies can be computed up to a desired regret bound. Hence, we use the infinite-horizon solver with a reasonable regret bound and restrict the depth of the policy tree for online execution on the robot. The LL actions are therefore not necessarily optimal, but reasonable solutions are obtained if $N$ is set heuristically based on the query complexity:

$$N = N_{min} + k \cdot x \tag{6}$$

where $x$ represents the number of object features being analyzed. In our experiments, we set $k = 2$ and $N_{min} = 2$ because we consider at least two visual actions and allow for each visual action to be applied twice to improve the reliability of the response. This constraint is not an essential component of our approach: it is a robot-specific trade-off for efficient operation.

The cost of an HL sensing action, say $u_1$, is the average cost of executing the actions represented by the corresponding LL-POMDP's policy tree: $\pi_1$. The cost is computed by a recursive process starting from the root node:

$$\begin{aligned} C_{root} &= \sum_{n_1 \in \mathcal{N}_{root,1}} C_{n_1} \\ C_{i_k} &= \sum_{j_{k+1} \in \mathcal{N}_{i_k,k+1}} P(j_{k+1}|i_k)\left\{C_{i_k}^{j_{k+1}} + C_{j_{k+1}}\right\} \end{aligned} \tag{7}$$

where $n_1$ is the $n$th node of the $\mathcal{N}_{root,1}$ nodes at level 1 that are the children of the root node; $C_{i_k}$ is the cost of the node $i$ at level $k$; while $C_{j_{k+1}}$ is the cost of the $j$th node among the $\mathcal{N}_{i_k,k+1}$ nodes at level $k+1$ that are children of node $i_k$. The term $P(j_{k+1}|i_k)$ is the transition probability from node $i_k$ to child node $j_{k+1}$ (Eq. (5)). The term $C_{i_k}^{j_{k+1}}$ is the cost of performing the action at node $i_k$ that created the child node $j_{k+1}$—it is the *reward* or *cost* of the action, as given by Eq. (1).

An observation function has to be computed conditioned on the underlying state, and this must be done for every possible underlying state. Parsing the LL-POMDP's policy tree for computing the observation function and reward specification for the HL-POMDP model is hence different from the normal belief update performed when computing or executing the LL policy. For instance, when $\pi_1$ is evaluated for computing the probability of $FR_1$ for the query: "which objects in the scene are blue?", we are computing the probability of finding a blue object in $R_1$ *conditioned on the fact that it exists in the ROI*, information that is not available when the LL policy is computed or executed. The initial beliefs of the LL-POMDPs are hence modified (based on the query) while generating the HL-POMDP model. For instance, while searching for "blue" objects, states $B_c^a\phi_s^a$, $B_c^aC_s^a$, $B_c^aT_s^a$, $B_c^aS_s^a$, $B_c^aM_s^a$ are equally likely in the initial belief, while other states have zero probability. The LL observation function is re-weighted based on the modified initial belief. The HL observation function and reward specification are then computed by parsing the corresponding LL policy trees with the belief states corresponding to the elements of the HL observation set. However, these LL changes are used *only* for building the HL-POMDP. Normal belief updates in the LL-POMDPs use an unmodified $\mathcal{O}$ and an appropriate initial belief: for e.g., it is uniform if nothing is known about the contents of the ROI.

Once the HL-POMDP's model parameters are computed, it can be solved to yield the HL policy for a specific query. During execution, the HL-POMDP's policy is queried for the best action choice. The chosen action causes the execution of one of the LL policies, resulting in a sequence of visual operators being applied on one of the ROIs. The answer provided by the LL policy execution causes a belief update in the HL-POMDP and the process continues until a terminal action is chosen in the HL to answer the query. Here it identifies all *blue* objects in the image.

The transfer of control between the HL and the LL is different when the LL actions change the state. Such a change in the state may require the HL model to be reformulated. For instance, the execution of *rSplit$_{color}$* splits the target ROI into smaller ROIs by color segmentation (i.e., by clustering image regions with similar color) and the operator *color* is applied on each of these ROIs. However, each of these ROIs is not evaluated up to $N$ levels. Instead, the current belief states of all the LL ROIs, including those created as a result of the split, are used to create a new HL-POMDP model with appropriate parameters. The new HL model is solved to get the new HL policy, which is used to select subsequent HL actions. Section 4.2 provides an illustrative example. A similar procedure can be used when the robot changes the state by interacting with the objects (e.g., pushing an object), but our current system does not yet incorporate such actions.

Algorithm 1 describes the overall planning algorithm for our domain. Based on the given query, the available visual actions and the number of ROIs, the LL-POMDPs are created and solved (line 1). The LL policies are parsed to build the HL-POMDP and solve it (line 2). During execution, invoking the HL-Policy (line 3) results in the analysis of the corresponding ROI (line 5). Under normal circumstances, the ROI is analyzed until a terminal action is reached in the LL (lines 7–8, 15–18). However, the execution of some LL actions can change the underlying state space dimensions (line 9), leading to a change in the execution cycle. For instance, if *rSplit$_{color}$* is executed and new ROIs are created, new POMDP models are created and solved for the resulting ROIs (line 11). The color operator is applied on all the ROIs created as a result of *rSplit$_{color}$* in order to update their beliefs (line 12), and the processing is terminated in the LL (line 13).

In the HL, the beliefs are typically updated based on the LL response (line 24) and a new action is chosen. However, if the most recent LL action to be executed causes a state change, a new HL-POMDP is created based on all existing LL-POMDPs and the current beliefs (line 21). This HL-POMDP is solved to obtain the updated HL policy. A new HL action is chosen using the modified HL policy (line 26) and the process continues until a terminal action is executed in the HL to answer the input query (line 28).

In summary, we propose a hierarchy in the (image) state and action space. Instead of manually encoding the hierarchy, abstractions and model parameters across multiple levels, our hierarchy only has two levels whose models are generated automatically. In the LL, each ROI is assigned a POMDP, whose states, actions and observations depend on the query and the available visual operators. The approximate (policy) solutions of the LL-POMDPs are used to populate an HL-POMDP that has completely different states, actions and observations. The HL-POMDP maintains the belief over the entire image

**Algorithm 1** HiPPo algorithm for the tabletop scenario.

```
Require: $Q$ = Input query/command, visual actions to apply on the ROIs, learned observation and reward functions for each action.
Require: $M$ = number of ROIs in image after initial visual processing.
 1: Create and solve POMDPs for each ROI (i.e., LL-POMDPs) based on input query, and learned observation and reward functions.
 2: Create and solve HL-POMDP based on query and LL policy trees.
 3: $a_{HL} = bestAction$(HL-Policy). {Choose HL action based on HL policy}
 4: while !terminalAction($a_{HL}$) do
 5:   $R_i$ = ROI corresponding to $a_{HL}$.
 6:   $a_{LL} = bestAction$(LL-Policy$_{R_i}$). {Choose visual operator based on LL policy of $R_i$}
 7:   while !terminalAction($a_{LL}$) do
 8:     Execute $a_{LL}$. {Apply visual operator on $R_i$}
 9:     if stateChanged then
10:       {Region-splitting may create new LL ROIs and change state dimensions}
11:       Create and solve new LL-POMDPs.
12:       Update beliefs of all relevant POMDPs.
13:       break;
14:     else
15:       Update belief of $R_i$ based on observation obtained by executing $a_{LL}$.
16:       $a_{LL} = bestAction$(LL-Policy$_{R_i}$).
17:     end if
18:   end while
19:   if stateChanged then
20:     {The new LL ROIs created by region-splitting need to be included in the HL}
21:     Create and solve HL-POMDP.
22:     Reset HL beliefs based on the belief state that existed before region-splitting.
23:   else
24:     Update HL belief based on LL response.
25:   end if
26:   $a_{HL} = bestAction$(HL-Policy).
27: end while
28: Answer input query.
```

and chooses the ROI that is most appropriate for further processing. The execution of the HL and LL policies eventually provides an answer to the input query. As a result of the automatic belief propagation, the proposed hierarchy can be used unmodified for a range of queries. Furthermore, all reward and observation models are learned: in the LL they are modeled based on the collected statistics (see Section 4.1), and in the HL they are inferred from the LL policies.

### 3.4. *Continual planning*

As described in Section 1, we compare our hierarchical POMDP approach against a modern planner that handles uncertainty in a qualitative manner: Continual Planning (CP) [40]. This section describes how the visual planning task can be formulated in the CP approach. As mentioned in Section 2.2, CP interleaves planning, plan execution and plan monitoring. It postpones reasoning about uncertain states by allowing actions to assert that the preconditions for the action will be met when the agent reaches that point in the execution of the plan. Replanning is triggered if these preconditions are not met during execution, or are met earlier. As an example, consider the specification of a *color* operator in the CP framework:

```
(:action colorDetector
:agent (?a - robot)
:parameters (?vr - visRegion ?colorP - colorProp )
:precondition (not (applied-colorDetector ?vr) )
:replan (containsColor ?vr ?colorP)
:effect (and
   (applied-colorDetector ?vr)
   (containsColor ?vr ?colorP) ) )
```

The operator aims to determine the color label of the input ROI, the same function performed by the *color* operator in the POMDP framework described in Section 3.1. The variable names are prefixed by '?' and each variable's *type* or *category* is specified when the variable is used for the first time. For instance, ?*vr* is a variable of type "visual-region" (*visRegion*), which can be bound to an existing image ROI at run-time. The parameters of the operator are an ROI that is currently being analyzed and a color-property (*colorProp*) that can take any of the possible labels provided by the operator (e.g., *red*, *green*, *blue* for *color*). The operator can be applied on any ROI that satisfies the "precondition", i.e., any ROI that has not already been analyzed by this operator. The expected result ("effect:") is that the color of the ROI is found. The "replan:" condition ensures that if the effect is produced by another process (for e.g., a human or another visual operator adds the ROI's color to the known state information), the current plan is terminated. Replanning generates a new plan that includes the *containsColor* fact in the known state information—*colorDetector* will therefore not be applied on this ROI again. In addition, if the results of executing a plan step are not as desired (for e.g., the color of an ROI is found to be *red* while searching for *blue* objects), execution monitoring triggers replanning to ensure that other ROIs are considered.

Other operators for determining shape or object category (equivalents of *shape* and *sift* in the POMDP formulation) are defined in a similar manner. Based on the goal state definition, the planner chooses the sequence of operators whose effects provide parts of the desired goal state—an example of the planning and execution cycle in CP is provided in Section 4.2.

The CP approach to the problem is more responsive to an unpredictable world than a non-continual classical planning approach—it can significantly reduce planning time in the event of deviations from expected performance. CP provides an efficient and appealing option because it does not require an analysis of all possible contingencies in advance. In addition, it has been used successfully for the tabletop domain (Section 1.1) and other human robot interaction scenarios [8]. Furthermore, the key contribution of the proposed work is a hierarchical planning framework for efficient and reliable visual processing—the goal is *not* to develop a POMDP planner or improve the visual operators. In the next section, the proposed HiPPo approach is hence compared experimentally against CP in the tabletop domain. We show that the ability to model action outcomes and accumulate belief enables HiPPo to perform more reliably than CP in domains with uncertainty, while still having comparable computational efficiency.

## 4. Experimental setup and results

In this section, we first describe the learning of LL-POMDP observation functions and then summarize the experimental results.

### 4.1. Learning observation functions and reward specifications

As described in Section 3.3, building the LL-POMDP models requires the observation functions and reward specifications of the visual operators. Unlike existing POMDP-based applications where these model parameters are manually specified [13, 44], we explicitly model the uncertainty in the performance of the different visual operators. The observation and reward functions are hence learned by the robot before planning and execution.

Objects with known labels (“red circular mug”, “blue triangle” etc.) are put in front of the robot. The robot applies each operator several times to estimate the probability of obtaining different outcomes ($\phi$, class labels and $U$) given the actual state information ($\phi$, class labels and $M$). The collected statistics are used to populate the observation functions of the individual visual operators, assuming that the observations produced by different actions are mutually independent.

In parallel, the reward specifications are also learned. The rewards (i.e., costs) of the operators are a function of two factors: the relative run-time complexity of the operators and the size of the visual region (ROI) that the operator is applied on—Eq. (1) in Section 3.1. The relative run-time complexity is determined experimentally based on the operator run-time statistics collected during observation function learning. The least expensive operator is assigned a run-time complexity value close to one and the values for other operators are multiples of this base value. The dependence on ROI-size is modeled as:

$$f(r) = a_0 + \sum_{k=1}^{N} a_k \cdot r^k \tag{8}$$

where $r$ is the ROI-size (in pixels), i.e., a polynomial of a specific degree is used to approximate the dependence on the size of the ROI being processed. The coefficients of the polynomial are estimated by performing Gaussian elimination [50] on the statistics of operator performance collected by the robot while learning the observation functions. The robot chooses the degree of the polynomial such that it best fits the collected statistics—for e.g., values of $N = 3, 1, 5$ were estimated for *color*, *shape* and *sift* respectively. The robot is hence able to learn the action costs used in Eq. (1).

During the initial learning phase, the robot also learns a model for the scene background in the absence of the target objects. This background model is represented as a Gaussian mixture model whose parameter values are estimated based on images of the background. During online operation, this background model is used to generate contours corresponding to the objects introduced in the scene. The contours are enveloped within rectangular boundaries to obtain the salient ROIs. Our system includes techniques such as saliency computation [51] and sophisticated image segmentation [52], which may be used to generate the ROIs. However, background subtraction is computationally efficient and suffices for our experimental domain.

### 4.2. Experimental setup and examples

The experimental setup is as follows. The robot uses a color camera to observe the tabletop scene. Any change from the learned model of the background is identified as a salient region, and all such regions of interest (ROIs) are extracted. The goal is to answer queries by applying a sequence of operators on one or more of the ROIs. For the first illustrative example, assume that there is no overlap between objects in the image, and that the robot can choose from *color*, *shape* and *sift*. Consider the query: “where are the blue circles?”, i.e., the robot has to determine the location of one or more blue circles in the image.

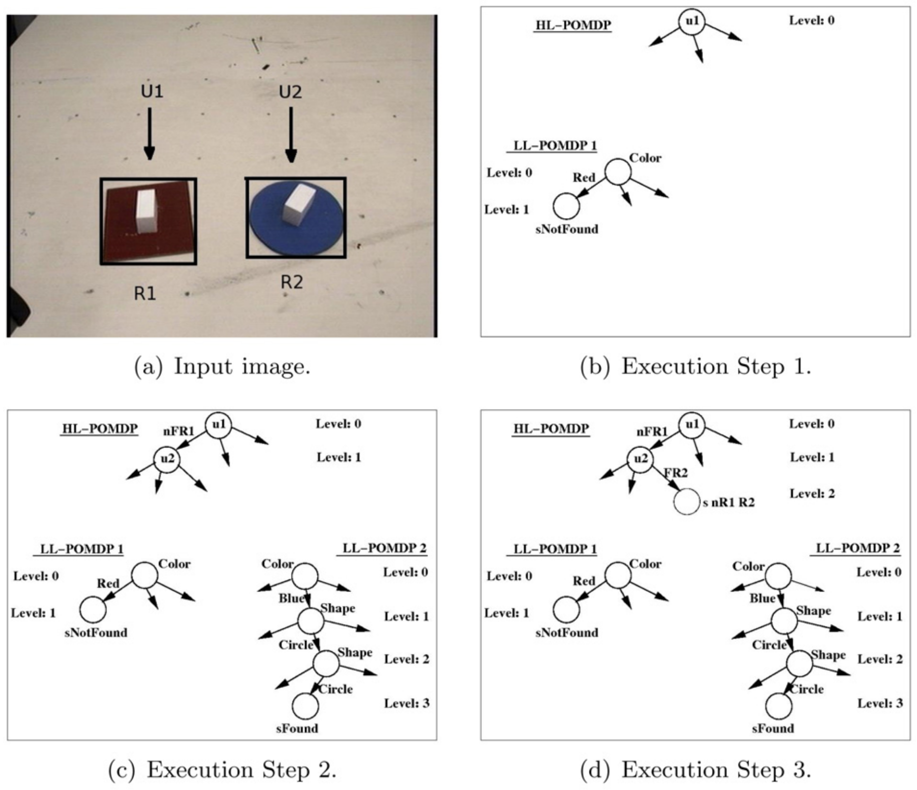


**Fig. 5.** Example query: "Where are the Blue Circles?" Appropriate LL-POMDP reward specification results in early termination when negative evidence is found.

In the HiPPo approach, the robot creates an LL-POMDP model file for each ROI based on the query and the available visual operators. The model file is in the format required by the ZMDP planning package.[3] A point-based solver in the same package is used to determine the LL-policies [16]. The LL policies are used to create policy trees, which are parsed to determine the observation functions and costs for the HL-POMDP model. The HL model is solved to get the HL policy. Figs. 5(a)–(d) show intermediate steps in the execution of the proposed method on an image with two ROIs.

The task is to determine the location of one or more *blue circles* in Fig. 5(a). No prior information is available about either ROI, i.e., both ROIs have uniform initial belief. Hence, the HL-POMDP first chooses to execute the policy of the smaller ROI $R_1$ because it has smaller action costs associated with it: action $u_1$ in Fig. 5(b). The corresponding LL-POMDP runs the color operator on the ROI. The outcome of applying an operator is one of the possible observations: $\phi_c^o, R_c^o, G_c^o, B_c^o, U_c^o$ for *color*. Here, the outcome is *red* ($R_c^o$), which is used to update the subsequent belief state. Despite being more costly than the shape operator, the color operator is applied first because the observation function of *color* indicates a higher likelihood of success in comparison to *shape*. When the outcome increases the belief of the states representing the "red" property, the likelihood of finding a blue circle is reduced significantly. The reward specification ($\alpha = 0.2$ in Eq. (1)) ensures a trade-off between computation and reliability, and there is no further analysis of $R_1$. In the next step, the *best* action chosen for analyzing $R_1$ is a terminal action: *sNotFound*. The HL-POMDP obtains an observation that the target object (blue circle) is not found in $R_1$, leading to a belief update and subsequent action selection: $u_2$ in Fig. 5(c). Then $R_2$'s LL policy is invoked, causing *color* and *shape* to be applied in turn on $R_2$. The higher uncertainty of *shape* is the reason why it is applied twice, and a different image used for each execution to ensure failure independence. Once the uncertainty in the LL belief for ROI $R_2$ is reduced sufficiently, a terminal action (*sFound*) is chosen—the increased reliability therefore comes at the cost of execution overhead. This response from $R_2$'s LL-POMDP is used to update the HL belief. In the next step, a terminal action is chosen in the HL-POMDP: ($s\neg R_1 \wedge R_2$), thereby stating that a *blue circle* exists in $R_2$ and not $R_1$—Fig. 5(d).

One significant advantage of the POMDP-based approach is that it provides the means to include prior knowledge in the decision-making, whenever such knowledge is made available. Consider the scene in Fig. 5(a) and the query: "where is the blue circle?" For instance, if there is some prior bias for the existence of the *blue circle* in $R_2$, the initial belief distribution of $R_2$ can be modified to represent this information. Then, the cost of executing $R_2$'s policy will be lower in the HL-POMDP—$R_2$ will hence be analyzed first, thereby leading to a quicker response to the query.

[3] See http://www.cs.cmu.edu/~trey/zmdp/.

In the HiPPo framework, each HL-POMDP action chooses to execute the policy of one of the LL-POMDPs until termination, instead of executing just one action. It can be argued that the best option is to choose a new HL action at each step based on the current belief, instead of waiting for the LL-POMDP to terminate. However, the challenge is to develop a scheme that translates from the LL belief to the HL belief and allows for planning. In addition, this belief propagation has to take into account the different states, actions and observations at the LL and the HL. The proposed hierarchy addresses this challenge and automates the belief propagation between the LL and HL for a range of queries and commands. Furthermore, the example above shows that our approach still *does the right thing*, i.e., it stops early if it finds negative evidence for the target object. Finding positive evidence only increases the posterior of the ROI being explored—even if the HL-POMDP chooses the next action, it will choose to process the same ROI.

Next, consider posing and answering the query: “where is the blue circle?” using the CP framework. The states and actions are defined in the appropriate format, as described in Section 3.4. The goal state is defined as the PDDL string:

```
  (and (exists ?vr - visRegion) (and (containsColor ?vr Blue)(containsShape ?vr Circle) ))
```

i.e., the goal is to determine the existence of an ROI which has the color *blue* and shape *circle*. The state of the system consists of:

```
  (newDeclaration “vr0”, “visRegion”)
  (newDeclaration “vr1”, “visRegion”)
```

i.e., there are two ROIs in the scene. The *types* are defined as:

```
(:types
  ;; Types for general variables...
  agent boolean - object
  ;; Define visual regions and their properties...
  visRegion colorProp objectType shapeProp - object
)
```

where the *agent*, ROI and its properties such as color are defined as valid variables (i.e., objects). The variables can take the following values:

```
(:constants
  ;;;Possible color labels...
  red green blue - colorProp
  ;;; Possible shape labels...
  circle triangle square - shapeProp
  ;;; Possible object types detectable by SIFT...
  picture mug box - objectType
  ;;; Boolean values that variables can take...
  true false - boolean
  ;;; Dummy agent to use for the operators...
  robot - agent
)
```

where the valid outcomes of each action are defined, along with the fact that a *robot* is a valid agent in the system. There is no representation of uncertainty in the definition of actions or observations.

The task of the planner is to find a sequence of operators to satisfy the goal state. In the current example, the following plan is created:

```
  (colorDetector robot vr0 blue)
  (shapeDetector robot vr0 circle)
```

i.e., the robot (the *agent* in the system) is to apply the color operator followed by the shape operator on the first ROI. There is a single execution of each operator on the ROI. Even if an operator determines an incorrect class label as the closest match with a low probability, there is no mechanism to incorporate the uncertainty. Any thresholds will have to be carefully tuned to prevent mis-classifications. If the color operator works correctly, it would classify the ROI’s color as *red*. Since the desired outcome (*blue*) is not achieved, the plan monitoring phase triggers replanning to create a new plan. The state after the execution of the first plan includes the fact that the first ROI’s color has been examined:

```
  (vr0 visRegion)
  (vr1 visRegion)
  (applied-colorDetector vr0)
  (containsColor vr0 red)
```

Though the new plan consists of the same visual operators as before, it directs the robot to apply the operators on the second ROI. Executing this plan results in the recognition of a *blue circle* in $R_2$, assuming the operators work correctly. The CP framework hence provides a mechanism to plan visual actions on a robot platform. It is representative of a modern planning scheme that has been used in human robot interaction scenarios [7,40]. It is possible to incorporate a notion of probability in the definitions of operators. However, CP does not have the ability to exploit the probabilistic action outcomes or to propagate beliefs. HiPPo is therefore able to provide significantly more reliable performance, as described in Section 4.3.

Most real-world scenes involve overlap between objects as a result of occlusion or the camera viewpoint. In such cases it makes sense to split an ROI into smaller ROIs based on one or more of the ROI’s features. Such a region-splitting operation would result in a modified state space for the HL-POMDP. We present an illustrative example that includes such actions. Consider the image shown in Fig. 6(a) with three objects, two of which overlap to create two ROIs. The query is: “where are the blue circles?” Since both ROIs are equally likely target locations, the HL-POMDP first chooses to execute the policy

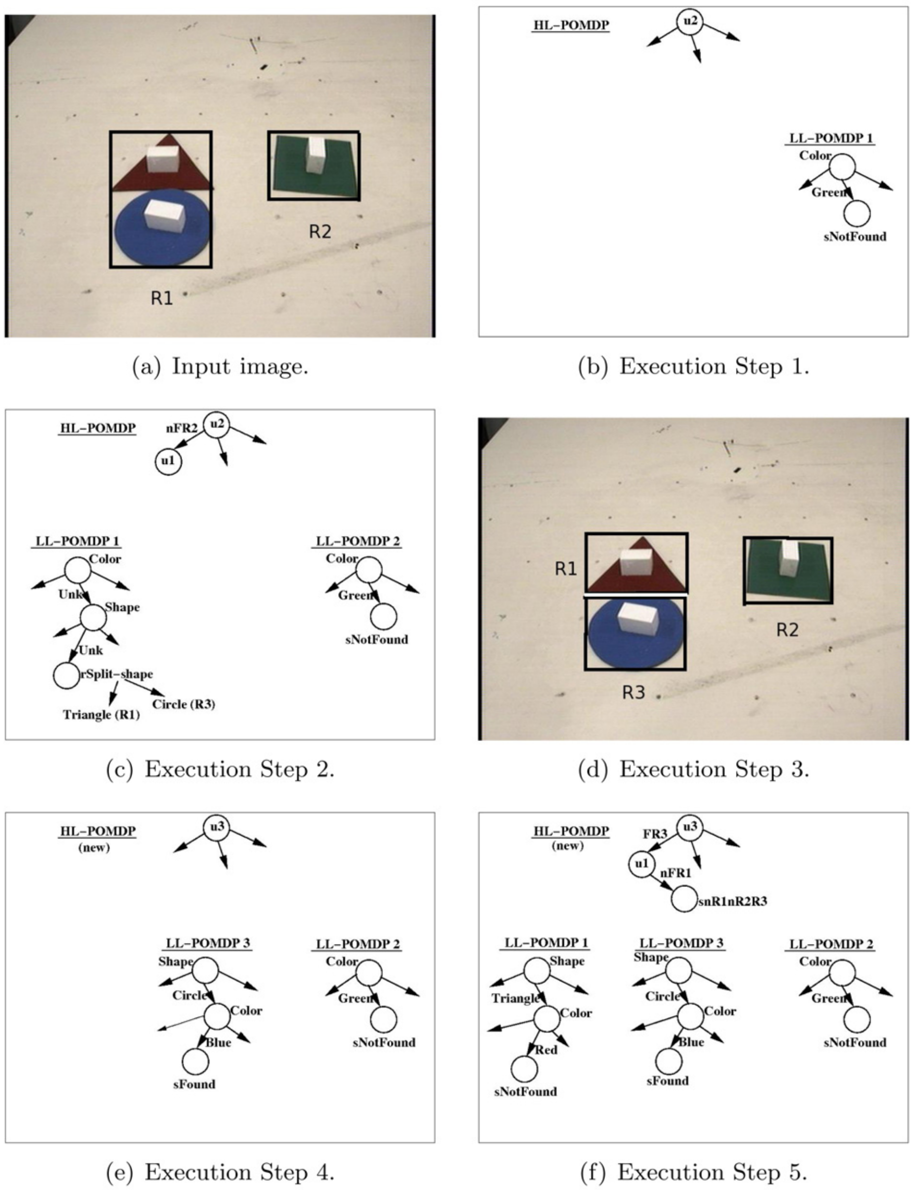


**Fig. 6.** Example query: "Where are the Blue Circles?" Region-splitting operators allow for the creation of appropriate ROIs to answer the query.

of the smaller ROI $R_2$ because of its smaller action costs: action $u_2$ in Fig. 6(b). The corresponding LL-POMDP applies the color operator on the ROI, leading to the outcome of *green*. As in the previous example, the likelihood of finding a blue circle is reduced significantly and the *best* action chosen at the next step is a terminal action: *sNotFound*. The HL-POMDP receives the input that the target object was not found in $R_2$, leading to a belief update and subsequent action selection: $u_1$ in Fig. 6(c). The LL policy of $R_1$ is invoked, causing *color* and *shape* to be applied in turn on the ROI. Both operators come up with outcomes of *unknown* because the ROI has two different colors and shapes. At this point, $rSplit_{shape}$ is chosen as the *best* action and $R_1$ is split into $R_1$ and $R_3$ on the basis of the shape contours within the ROI—Fig. 6(d). Our system includes other algorithms that can be invoked to split an ROI on the basis of color [52] or clustering of image features [50]. As described earlier, $rSplit_{shape}$ is followed by the application of *shape* on each sub-region, leading to the observations *triangle* and *circle* in $R_1$ and $R_3$ respectively—Fig. 6(c). The current beliefs are used to create and solve a new HL-POMDP for three ROIs. The subsequent action selection in the HL ($u_3$) results in the execution of the LL-policy of $R_3$, whose initial belief reflects the previous application of the shape operator. Hence, *color* and *shape* are applied just once before a terminal action is chosen—*sFound* in Fig. 6(e). The subsequent belief update and action choice in the HL leads to the processing of $R_1$ since the goal is to find the locations of *all* blue circles. The terminal action *sNotFound* in $R_1$ results in the terminal action $s\neg R_1 \wedge \neg R_2 \wedge R_3$ in the HL—Fig. 6(f).

Similar actions that change the state (for e.g., camera zoom) can be modeled using the HiPPo framework. It is also possible to use visual routines that characterize objects containing multiple colors and shapes. Posing and solving such problems in the CP or other non-probabilistic frameworks would be difficult. Even with probabilistic and other POMDP-based methods, achieving the automatic belief propagation of HiPPo would be a challenge.

## *4.3. Quantitative comparison*

In this section, we provide a quantitative comparison between CP and HiPPo. The goal is to test the following hypotheses:

1. HiPPo is more efficient than the (standard) POMDP formulation without the proposed hierarchical decomposition.
2. Using policy-caching to trade-off accuracy and efficiency (Appendix A) makes HiPPo's plan-time complexity comparable to CP.
3. HiPPo and CP are significantly more efficient than the naive approach of applying all available visual operators on the images.
4. HiPPo has higher execution time than CP but provides significantly more reliable performance.
5. HiPPo can represent operators that change the state, and hence handle scenes with partially overlapping objects.

These hypotheses were subjected to thorough experimental evaluation on the robot in the tabletop scenario. Objects of interest were placed on the table, and the robot had to analyze images of the scene in order to answer input queries and execute commands. The robot is equipped with a manipulator arm that can be used to grasp and move objects placed on the tabletop. The execution of input commands typically involves the robot identifying and moving the target objects to desired positions. The following categories of queries and commands were considered:

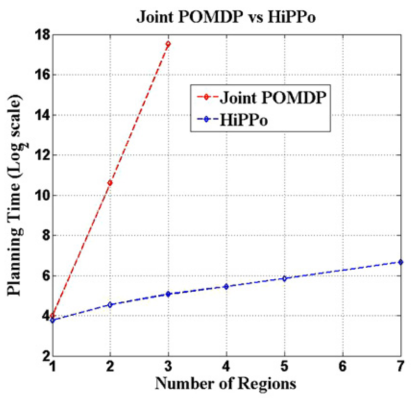
- *Occurrence queries*: the robot has to determine the presence or absence of the desired object(s) in the image. Examples include: "is there a blue mug in the scene?" and "is there a red triangle on the table?"
- *Location queries*: the robot has to determine the location of the target object. Examples include: "where is the blue circle in the image?" and "what is the location of the green object?"
- *Property queries*: the robot has to compute specific properties of specific objects. Examples include: "what is the color of the mug?" and "what is the shape of the red object?"
- *Scene queries*: the robot has to answer queries about the scene that could be a combination of the queries listed above. For instance: "How many green squares are there in the scene?"

Unlike our prior work on HiPPo [53], the experiments reported here included images of scenes with partially overlapping objects, which required the robot to plan with several additional operators that change the state dimensionality (see Section 3.1). In addition, the computed policies were tested on the robot, while our initial results involved several tests that were conducted offline. Furthermore, we included queries and commands of the form: "move the red circle to the left" that require the robot to confirm the occurrence of the target object and compute its location before performing the action. The translation of queries and commands to an internal representation is done using an existing natural language processing system [54]. As a result of these additional capabilities, we had to repeat our earlier experiments and run additional experiments to test the hypotheses listed above.

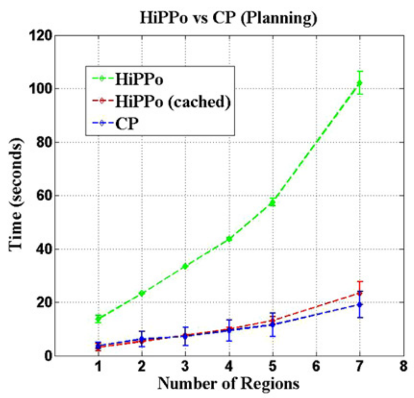
For each query category, we ran experiments over 20 different queries, with 20 trials for each such query. The tabletop scenario consisted of objects of different colors, shapes and other properties of interest (e.g., different object categories such as *box*, *mug* for *sift*). Example objects on the tabletop include *red square*, *blue mug*, *green triangle*, *red box*—some of these objects are shown in Fig. 1. Each query focused on an appropriate combination of the object properties—for instance, *red circle*, *blue*, *green triangle*, *red square box* etc. The set of queries hence represents different combinations of the object properties under consideration. In order to test the ability to tackle operators that change the state, some of the scenes were specifically designed to contain overlapping objects. The object arrangement was also modified to conduct experiments for different number of ROIs in the image (ranging from one to seven). These experiments and the tabletop domain may appear simplistic, and one could argue for running experiments with more natural scenes. However, the application domain represents the state of the art in cognitive robotics [8]. In addition, the goal is to develop an approach that automatically tailors visual processing to the task at hand. The lessons learned in this domain can hence be applied to more complex domains with other sophisticated operators.

In addition to HiPPo and CP, we also implemented the naive approach of applying all the visual processing operators on each image ROI. The naive approach trusts the operator outputs and does not propagate beliefs or uncertainty. It terminates when an ROI with the desired properties (to match the query) is found or all image ROIs are analyzed. The results of these experiments are summarized in Figs. 7(a)–(d), where the naive approach is denoted by the term "no planning".

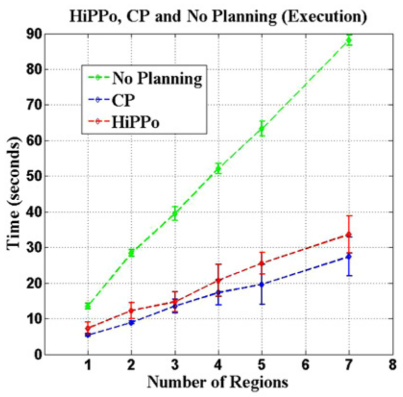
Fig. 7(a) compares the planning complexity of HiPPo against the standard POMDP approach that operates in the joint space of all the ROIs and actions. The "joint POMDP" suffers from the exponential state explosion problem and soon becomes intractable, even with the state of the art approximate solvers. The hierarchical approach significantly reduces the planning time, as observed in Fig. 7(a). However, the hierarchical representation does not provide the optimal solution (i.e., policy). Executing the hierarchical policy involves the use of approximate (i.e., heuristic) solvers that compute policy solu-



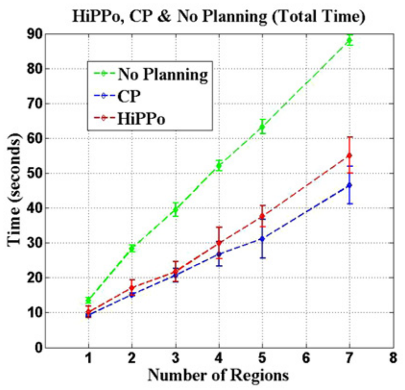
(a) HiPPo vs. joint POMDP. Joint POMDP soon becomes intractable.



(b) Planning times of HiPPo vs. CP. Policy-caching makes results comparable.



(c) Execution times of HiPPo, CP vs. No planning. Planning makes execution faster.



(d) Planning+execution times of HiPPo, CP vs. No planning.

**Fig. 7.** Experimental results: comparing planning and execution times of HiPPo and CP against no planning. HiPPo and CP reduce processing time.

tions only up to the desired regret bounds or number of levels in the policy tree. As a result of these approximations, the computed policies are not optimal except in some limited cases. For instance, while searching for a *red* region, the hierarchical representation is optimal if (and only if) every image ROI is colored red. Though the solutions are not necessarily optimal, the hierarchical decomposition provides reasonable policies that result in reliable and efficient performance.

Fig. 7(b) compares the planning times of HiPPo and CP as a function of the number of ROIs in the image. The default hierarchical approach takes more time than CP. The computationally intensive part of HiPPo is the computation of the policies for all the image ROIs at the LL of the hierarchy. However, the LL-POMDP models typically differ only in terms of the operator costs, and this difference in operator costs is primarily due to the difference in the ROI-sizes (Eqs. (1), (8)). If all the image ROIs were of the same size, a specific query would result in similar LL policies for all the ROIs, and the policies can hence be cached and reused after being computed once. Though the image ROIs, as expected, are not of the same size, several image ROIs have similar sizes. In order to reduce the computational load, the ROI sizes are discretized and all ROIs whose sizes fall within a particular size range are assigned the same operator costs. This discretization allows us to cache computed policies and re-use them for similar ROIs, thereby drastically reducing the planning time of HiPPo. With the cached policies, HiPPo has planning times comparable to CP, as seen in Fig. 7(b). However, such an approximation introduces value estimation errors. Appendix A describes how this error can be estimated theoretically and experimentally in order to trade-off accuracy against efficiency.

Next, Fig. 7(c) compares the execution time of HiPPo and CP against the naive approach of applying all the visual operators on each ROI until the desired result is found or all image ROIs are analyzed. HiPPo has a larger execution time than CP because it occasionally applies the same operator multiple times on an ROI, using different images of the same scene in order to reduce the uncertainty in its belief. However, the execution time of HiPPo is still comparable to that of CP.

Finally, Fig. 7(d) compares the planning approaches against the naive approach in terms of their combined planning and execution times. Planning provides benefits even on scenes with just two ROIs, and the benefits are much more significant as the number of ROIs increase. In simple cases with a small number of ROIs and visual operators, it may be possible to write simple rules that determine the operator sequence for each task. However, for a reasonable number of visual operators and

**Table 2**
Reliability of visual processing.

| Approach | Reliability |
|---|---|
| Naive | 76.67 |
| CP | 76.67 |
| HiPPo | 90.75 |

in situations where there are multiple operators for a feature (e.g., two actions that can find color), planning is an appealing option.

In all these experiments, the algorithms were tested on-board a robot that has multiple modules operating in parallel to analyze the inputs from different modalities (vision, tactile, speech etc.) and set up goals for other modules to achieve. Hence, though the individual operators represent optimized state of the art visual processing routines, recognizing changes and triggering appropriate operators may take some time. This parallelism partially accounts for the times reported in Figs. 7(a)–(d).

As mentioned earlier, the proposed approach aims to perform visual processing efficiently and reliably. HiPPo was therefore compared against CP and the naive approach in terms of its reliability. Over the range of queries and commands, the ground-truth labels of the target objects were provided manually and compared against the answers provided by HiPPo. Similar experiments were conducted using CP and the naive approach. The results are summarized in Table 2. The naive approach results in an average classification accuracy of 76.67%, i.e., the visual operators mis-classify approximately one-fourth of the objects. The naive approach accepts the operator outputs (most likely outcomes) and does not accumulate beliefs. In addition, it is computationally expensive and infeasible in many robot applications.

Using CP to plan the sequence of visual operators for a given query results in an accuracy of 76.67%, i.e., CP significantly reduces the execution time but can do no better than the naive approach in terms of reliability. CP does not improve the reliability because it fails to exploit the probabilistic outputs of the operators and handles uncertainty qualitatively. The CP framework is not well-suited for propagating beliefs and uncertainty. HiPPo, on the other hand, is designed to exploit the probabilistic outcomes of the operators in order to accumulate belief and reduce the uncertainty. The significantly higher reliability of 90.75% is a direct reflection of this capability. Based on these experiments, we observe that HiPPo is able to recover from instances where the operators do not provide the correct classification. The proposed approach fails only when the noise in information processing is consistent over several images of the scene. As the non-hierarchical POMDP takes a significantly larger amount of planning time for simple scenes, the optimal plan was not computed for scenes with more than two ROIs. However, for the cases where a plan was computed, there was no significant difference between the optimal approach and HiPPo in terms of reliability.

## 5. Conclusions and future work

Robots working on cognitive tasks in the real world require the ability to operate autonomously, tailoring sensing and information processing to the task at hand. In this paper, we have proposed a probabilistic planning framework that enables the robot to plan a sequence of visual operators in order to accomplish a given task with high reliability while still using the available resources in an optimal manner. In a representative domain where a robot and a human jointly converse about and manipulate objects on a tabletop, our hierarchical POMDP approach enables the robot to efficiently use learned models of the uncertainty in action outcomes to accumulate belief. Our approach hence responds to queries or commands with comparable efficiency and significantly higher reliability than a representative modern planning framework that had been applied to similar domains.

In our problem domain, we are still dealing with a relatively small set of operators and image ROIs. We have analyzed actions that observe the state of the ROI, and actions that modify the state—for e.g., region-splitting operators to analyze overlapping objects. With a large number of operators, solving the LL-POMDPs may become expensive. Similarly, solving the HL-POMDP may be computationally expensive for images with a large number of ROIs. In addition, we are interested in using the lessons learned in the tabletop scenario to apply the proposed approach to other complex domains. In such cases, it may be necessary to use a range of hierarchies [44] in the state and action spaces. A further extension would involve learning the hierarchy using approximations of recent theoretical developments [49].

The hierarchical decomposition incorporates approximations to speed up the planning. We have presented an analysis of the approximation error involved in policy-caching, which involves a trade-off between the computational effort involved in creating and solving the LL-POMDPs, and the accuracy of action costs computed for each ROI being analyzed. We have shown that the experimental error incurred by the approximation is significantly smaller than the theoretical estimate, and that the error incurred is offset by the reduction in the computational effort involved in creating and solving the POMDPs. A direction of further research is to compute the observation functions at different ROI-sizes so that the *value* and cost of each action can be computed more accurately. We also aim to analyze other approximations in the hierarchical decomposition.

In this paper, we have looked at analyzing different images of the same scene. When there are multiple objects in the scene, it may be required to move the camera in order to get a better view of the objects and to eliminate occlusions.

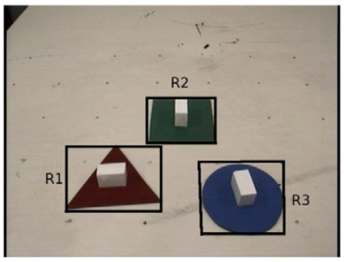


**Fig. 8.** Image with three ROIs extracted from the background.

In addition, the desired object may be present at a location that the robot cannot observe without first moving to an appropriate location. In such situations, the robot should be able to first choose a specific scene or viewpoint to observe the environment (based on the task), and then analyze images of the scene using a range of suitable visual operators. One direction of further research is the formulation of scene analysis as a POMDP that forms a higher level of abstraction over the existing hierarchy. The goal at the top layer of the hierarchy will be to choose actions (i.e., scenes to examine) that maximize the information gain, i.e., reduce the entropy in the belief distribution [55]. The challenge will be the efficient and automatic belief propagation and transfer of control between the levels of the hierarchy. We have performed proof of concept experiments that extend the hierarchy for vision-based high-level scene analysis [56]. The results show that the existing hierarchy holds promising potential for such extensions.

Currently the system can handle queries on object location, existence and identity. In the future, we plan to extend the proposed planning framework to more complex “relationship queries” (e.g., “is the red triangle to the left of the blue circle?”), and “action queries” (e.g., “can the red mug be grasped from above?”). We also plan to investigate the interaction of a mobile robot with the objects in the environment, and use the interactions (e.g., pushing or grasping the objects) to reason about action affordances.

The key contribution of this article is a hierarchical planning framework whose levels match the cognitive requirements of visual processing on a robot. The framework can jointly tailor sensing and information processing to the task at hand. The proposed system is able to automatically plan a sequence of visual operators that are individually unreliable, in order to achieve high overall reliability and efficiency. A similar framework can be developed for other application domains that use visual or non-visual sensors, by designing hierarchical decompositions that match the corresponding sensing and information processing requirements. Eventually the aim is to enable mobile robots to use a combination of learning and planning to respond autonomously, reliably and efficiently to a range of tasks, thereby interacting with and assisting humans in real-world applications.

## Acknowledgements

This work was supported in part by the EU FP6 IST Cognitive Systems Project (CoSy) FP6-004250-IP; the Leverhulme Trust Research Fellowship RF/2006/0235; the EU FP7 IST Project CogX FP7-IST-215181; and the ONR Science of Autonomy award N00014-09-1-0658.

## Appendix A. Approximation error estimation

During the creation of the LL-POMDPs for the image ROIs, we discretize the ROI-sizes and cache the policies in order to speed up the planning—Fig. 7(b). However, the POMDP models for each ROI are not identical since the cost of each visual operator is a function of the size of the ROI being operated upon—Eqs. (1), (8). In this section, we estimate the error introduced by policy caching. We show how to trade-off the computational effort expended in creating and solving the LL-POMDPs, against the error incurred by not computing the action costs accurately.

Consider the image shown in Fig. 8 with three ROIs extracted from the background. The individual ROI-sizes for $R_1$, $R_2$, $R_3$ are 23 400, 11 050 and 20 800 pixels respectively. We have three discretization options: (1) different action costs for each individual ROI, which would require the LL models and policies to be computed thrice; (2) same action costs for $R_1, R_3$ and a different set of action costs for $R_2$, leading to the creation of LL models and policies twice; (3) same set of action costs for all three ROIs, i.e., the LL models need to be created and solved just once. In terms of computational costs for creating the LL models and policies:

$$\begin{aligned} \text{ModelCost}_{Option_2} &= 0.667 \times \text{ModelCost}_{Option_1} \\ \text{ModelCost}_{Option_3} &= 0.333 \times \text{ModelCost}_{Option_1} \end{aligned} \tag{9}$$

On the other hand, the discretization of ROI-sizes causes an approximation error. We compute a theoretical upper bound first. The maximum approximation error in the action costs, over all visual operators and ROIs whose sizes fall within the discretization range, is given by:

$$max_{\substack{a\in\mathcal{A}\\ a\notin\mathcal{A}_{\mathcal{S}}}} \left|f(r_i) - f(r_{avg})\right| = \delta \tag{10}$$

For instance, in $Option_2$:

$$r_i = \{\text{ROI-size}(R_1), \text{ROI-size}(R_3)\}$$
$$r_{avg} = (\text{ROI-size}(R_1) + \text{ROI-size}(R_3))/2$$

whereas in $Option_3$:

$$r_i = \{\text{ROI-size}(R_1), \text{ROI-size}(R_2), \text{ROI-size}(R_3)\}$$
$$r_{avg} = (\text{ROI-size}(R_1) + \text{ROI-size}(R_2) + \text{ROI-size}(R_3))/3$$

Then, for a discount factor of $\gamma$ in the POMDP models, the net maximum error due to the ROI-size approximation is given by:

$$\begin{aligned} error &= \delta + \gamma \cdot \delta + \cdots + \gamma^{N-1} \cdot \delta \\ &= \delta \left\{ \frac{1-\gamma^N}{1-\gamma} \right\} \end{aligned} \tag{11}$$

where $N$ represents the number of steps over which an ROI's policy is executed, or equivalently it is the number of levels for which the corresponding policy tree is explored. For $\gamma = 0.9$ and $N = 8$, the upper bound (on the approximation error) computed in Eq. (11) is $\approx 6\delta$. Taking into account the ROI-sizes in Fig. 8, the upper bounds on the estimation errors in $Option_2$ and $Option_3$ are 0.33 and 2.35 respectively. We then compute the actual error for $Option_2$ by computing two policies for $R_1$ based on models that estimate actions costs using $r_{avg}$ and ROI-size$(R_1)$ respectively. The difference in the *values* of the two policies for the initial state of $R_1$ results in an experimental error of 0.021. A similar computation for $Option_3$ with $R_2$, the ROI where the maximum error is expected to be observed, results in an error of 0.232. We observe that the actual approximation error is significantly smaller than the upper bound on error.

The error estimation process described above can be used to autonomously determine the ROI-size discretization appropriate for a given image. Different discretization options can be compared by trading off the expected approximation error against the corresponding reduction in the computational effort involved in determining the models and policies. The observation functions would also vary for different ROI-sizes. However, a large amount of data would be required to learn the relationship between ROI-sizes and the performance of visual operators. We therefore do not (as yet) use different observation functions for different ROI-sizes.

## References

[1] N.J. Nilsson, Shakey the robot, Technical Report, AI Center Technical Note 323, SRI International, 1984.
[2] B.W. Minten, R.R. Murphy, J. Hyams, M. Micire, Low-order-complexity vision-based docking, IEEE Transactions on Robotics and Automation 17 (6) (2001) 922–930.
[3] J. Pineau, M. Montemerlo, M. Pollack, N. Roy, S. Thrun, Towards robotic assistants in nursing homes: Challenges and results, in: Special Issue on Socially Interactive Robots, Robotics and Autonomous Systems 42 (3–4) (2003) 271–281.
[4] S. Thrun, Stanley: The robot that won the DARPA grand challenge, Field Robotics 23 (9) (2006) 661–692.
[5] I. Horswill, Polly: A vision-based artificial agent, in: AAAI, 1993, pp. 824–829.
[6] L. Kaelbling, M. Littman, A. Cassandra, Planning and acting in partially observable stochastic domains, Artificial Intelligence 101 (1998) 99–134.
[7] N. Hawes, A. Sloman, J. Wyatt, M. Zillich, H. Jacobsson, G.-J. Kruiff, M. Brenner, G. Berginc, D. Skocaj, Towards an integrated robot with multiple cognitive functions, in: The Twenty-second National Conference on Artificial Intelligence (AAAI), 2007.
[8] CoSy, Cognitive systems for cognitive assistants, http://www.cognitivesystems.org/, 2008.
[9] R. Clouard, A. Elmoataz, C. Porquet, M. Revenu, Borg: A knowledge-based system for automatic generation of image processing programs, IEEE Transactions on Pattern Analysis and Machine Intelligence 21 (2) (1999) 128–144.
[10] S. Moisan, Program supervision: YAKL and PEGASE+ reference and user manual, Rapport de Recherche 5066, INRIA, Sophia Antipolis, France, 2003.
[11] L. Li, V. Bulitko, R. Greiner, I. Levner, Improving an adaptive image interpretation system by leveraging, in: Australian and New Zealand Conference on Intelligent Information Systems, 2003.
[12] M. Ghallab, D. Nau, P. Traverso, Automated Planning: Theory and Practice, Morgan Kaufmann, San Francisco, CA, 2004.
[13] A.F. Foka, P.E. Trahanias, Real-time hierarchical POMDPs for autonomous robot navigation, in: IJCAI Workshop on Reasoning with Uncertainty in Robotics, 2005.
[14] J.M. Porta, M.T.J. Spaan, N. Vlassis, Robot planning in partially observable continuous domains, in: Robotics: Science and Systems, 2005.
[15] M. Brenner, B. Nebel, Continual planning and acting in dynamic multiagent environments, in: The International Symposium of Practical Cognitive Agents and Robots, 2006.
[16] T. Smith, R. Simmons, Point-based POMDP algorithms: Improved analysis and implementation, in: UAI, 2005.
[17] R.A. Brooks, A robust layered control system for a mobile robot, Robotics and Automation 2 (1986) 14–23.
[18] J.E. Laird, A. Newell, P. Rosenbloom, SOAR: An architecture for general intelligence, Artificial Intelligence 33 (3) (1987) 1–64.
[19] J.R. Anderson, D. Bothell, M.D. Byrne, S. Douglass, C. Lebiere, Y. Qin, An integrated theory of the mind, Psychological Review 111 (4) (2004) 1036–1060.
[20] S. Chien, F. Fisher, T. Estlin, Automated software module reconfiguration through the use of artificial intelligence planning techniques, IEE Proc. Software 147 (5) (2000) 186–192.
[21] L. Gong, Composition of image analysis processes through object-centred hierarchical planning, Ph.D. thesis, Rutgers University, 1992.
[22] S. Chien, H. Mortensen, Automating image processing for scientific data analysis of a large image database, IEEE Transactions on Pattern Analysis and Machine Intelligence 18 (8) (1997) 854–859.

[23] M. Thonnat, S. Moisan, What can program supervision do for program reuse?, IEE Proc. Software 147 (5) (2000) 179–185.
[24] S.J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, 2nd ed., Prentice Hall, Upper Saddle River, New Jersey, 2003.
[25] C. Shekhar, S. Moisan, M. Thonnat, Use of a real-time perception program supervisor in a driving scenario, in: Intelligent Vehicle Symposium '94, Paris, France, 1994.
[26] X. Jiang, H. Bunke, Vision planner for an intelligent multisensory vision system, Technical Report, University of Bern, Switzerland, 1994.
[27] T. Darrell, Reinforcement learning of active recognition behaviors, Technical Report, Interval Research Technical Report 1997-045, 1997.
[28] R.L. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, USA, 1998.
[29] D. Nikovski, I. Nourbakhsh, Learning probabilistic models for decision-theoretic navigation of mobile robots, in: The International Conference on Machine Learning (ICML), 2000.
[30] J. Vogel, N. de Freitas, Target-directed attention: Sequential decision-making for gaze planning, in: The International Conference of Robotics and Automation (ICRA), 2008, pp. 2372–2379.
[31] C. Boutilier, K. Regan, P. Viappiani, Preference elicitation with subjective features, in: The 3rd ACM Conference on Recommender Systems (RecSys-09), 2009.
[32] C. Boutilier, A POMDP formulation of preference elicitation problems, in: The National Conference on Artificial Intelligence (AAAI), 2002, pp. 239–246.
[33] M.L. Littman, A.R. Cassandra, L.P. Kaelbling, Learning policies for partially observable environments: Scaling up, in: Proceedings of the Twelfth International Conference on Machine Learning, Morgan Kaufmann, 1995, pp. 362–370.
[34] C. Kreucher, K. Kastella, A. Hero, Sensor management using an active sensing approach, IEEE Transactions on Signal Processing 85 (3) (2005) 607–624.
[35] A.O.I. Hero, D.A. Castanon, D. Cochran, K. Kastella, Foundations and Applications of Sensor Management, Springer, 2008.
[36] A. Krause, A. Singh, C. Guestrin, Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies, Technical Report, Carnegie Mellon University CMU-ML-07-108, 2007.
[37] A. Krause, A. Singh, C. Guestrin, Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies, Journal of Machine Learning Research 9 (2008) 235–284.
[38] D. Draper, S. Hanks, D. Weld, A probabilistic model of action for least-commitment planning with information gathering, in: Uncertainty in AI (UAI), 1994.
[39] R. Petrick, F. Bacchus, Extending the knowledge-based approach to planning with incomplete information and sensing, in: International Conference on Automated Planning and Scheduling (ICAPS), 2004, pp. 2–11.
[40] M. Brenner, B. Nebel, Continual planning and acting in dynamic multiagent environments, Journal of Autonomous Agents and Multiagent Systems 19 (3) (2009) 297–331.
[41] J. Hoffmann, B. Nebel, The FF planning system: Fast plan generation through heuristic search, Journal of Artificial Intelligence Research 14 (2001) 253–302.
[42] D. McDermott, PDDL: The planning domain definition language, Technical Report TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.
[43] D. Lowe, Distinctive image features from scale-invariant keypoints, International Journal of Computer Vision (IJCV) 60 (2) (2004) 91–110.
[44] J. Pineau, S. Thrun, High-level robot behavior control using POMDPs, in: The National Conference on Artificial Intelligence (AAAI), 2002.
[45] T. Dietterich, The MAXQ method for hierarchical reinforcement learning, in: International Conference on Machine Learning (ICML), 1998.
[46] J. Pineau, G. Gordon, POMDP planning for robust robot control, in: The Twelfth International Symposium on Robotics Research, 2005.
[47] E.A. Hansen, R. Zhou, Synthesis of hierarchical finite-state controllers for POMDPs, in: ICAPS, 2003, pp. 113–122.
[48] G. Theocharous, K. Murphy, L.P. Kaelbling, Representing hierarchical POMDPs as DBNs for multi-scale robot localization, in: International Conference on Robotics and Automation (ICRA), 2004.
[49] M. Toussaint, L. Charlin, P. Poupart, Hierarchical POMDP controller optimization by likelihood maximization, in: Uncertainty in AI (UAI), 2008.
[50] R.O. Duda, P.E. Hart, D.G. Stork, Pattern Classification, 2nd ed., Wiley Publishers, 2000.
[51] L. Itti, C. Koch, E. Niebur, A model of saliency-based visual attention for rapid scene analysis, IEEE Transactions on Pattern Analysis and Machine Intelligence 20 (11) (1998) 1254–1259.
[52] P.F. Felzenswalb, D.P. Huttenlocher, Efficient graph-based image segmentation, International Journal of Computer Vision (IJCV) 59 (2) (2004) 167–181.
[53] M. Sridharan, J. Wyatt, R. Dearden, HiPPo: Hierarchical POMDPs for planning information processing and sensing actions on a robot, in: International Conference on Automated Planning and Scheduling (ICAPS), 2008.
[54] G.-J.M. Kruijff, H. Zender, P. Jensfelt, H.I. Christensen, Situated dialogue and spatial organization: What, where… and why?, in: Special Issue on Human–Robot Interaction, International Journal of Advanced Robotic Systems 4 (1) (2007) 125–138.
[55] N.J. Butko, J.R. Movellan, I-POMDP: An infomax model of eye movement, in: The IEEE International Conference on Development and Learning (ICDL), 2008.
[56] S. Zhang, M. Sridharan, Vision-based scene analysis on a mobile robot using layered POMDPs, in: International POMDP Practitioners Workshop, Toronto, Canada, 2010.