

# HiPPo: Hierarchical POMDPs for Planning Information Processing and Sensing Actions on a Robot

Mohan Sridharan and Jeremy Wyatt and Richard Dearden

University of Birmingham, UK  
{mzs,jlw,rwd}@cs.bham.ac.uk

## Abstract

Flexible general purpose robots need to tailor their visual processing to their task, on the fly. We propose a new approach to this within a planning framework, where the goal is to plan a sequence of visual operators to apply to the regions of interest (ROIs) in a scene. We pose the visual processing problem as a Partially Observable Markov Decision Process (POMDP). This requires probabilistic models of operator effects to quantitatively capture the unreliability of the processing actions, and thus reason precisely about trade-offs between plan execution time and plan reliability. Since planning in practical sized POMDPs is intractable we show how to ameliorate this intractability somewhat for our domain by defining a hierarchical POMDP. We compare the hierarchical POMDP approach with a Continual Planning (CP) approach. On a real robot visual domain, we show empirically that all the planning methods outperform naive application of all visual operators. The key result is that the POMDP methods produce more robust plans than either naive visual processing or the CP approach. In summary, we believe that visual processing problems represent a challenging and worthwhile domain for planning techniques, and that our hierarchical POMDP based approach to them opens up a promising new line of research.

## Introduction

Current robot visual systems are designed by hand for specific visual tasks, for specific robots, to run in specific if challenging domains. If we are to achieve more general purpose robots we need to develop methods by which a robot can tailor its visual processing on the fly to match its current task. While there exists a body of impressive work on planning of image processing (Clouard et al. (1999); Thonnat and Moisan (2000); Moisan (2003); Li et al. (2003)), it is largely used for single images, requires specialist domain knowledge to perform re-planning or plan repair, has only been extended to robotic systems in the most limited ways, and poses the problem in an essentially deterministic planning framework or as a MDP (Li et al. (2003)). In this paper we push the field of planning visual processing in a new direction by posing the problem as an instance of probabilistic sequential decision making. We pose it as a Partially Observable Markov Decision Process (POMDP), thereby tak-

ing explicit, quantitative account of the unreliability of visual processing. Our main technical contribution is that we show how to contain one aspect of the intractability inherent in POMDPs for this domain by defining a new kind of hierarchical POMDP. We compare this approach with a formulation based on the Continual Planning (CP) framework of Brenner and Nebel (2006). Using a real robot domain, we show empirically that both planning methods are quicker than naive visual processing of the whole scene, even taking into account the planning time. The key benefit of the POMDP approach is that the plans, while taking slightly longer to execute than those produced by the CP approach, provide significantly more reliable visual processing than either naive processing or the CP approach.

In our domain, both a robot and human can converse about and manipulate objects on a table top (see Hawes et al. (2007)). 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 vision problems is hard in itself, together they are extremely challenging. The challenge is to build a vision system able to tackle all these tasks. One early approach to robot vision was to attempt a general purpose, complete scene reconstruction, and then query this model for each task. This is still not possible and in the opinion of many vision researchers will remain so. An idea with a growing body of evidence from both animals and robots is that some visual processing can be made more effective by tailoring it to the task/environment (Lee (1980); Horswill (1993); Land and Hayhoe (2001)). In this paper we ask how, given a task, a robot could infer what visual processing, of which areas of the scene, should be executed.

Consider the scene in Figure 1, and consider the types of visual operations that the robot would need to perform to answer a variety of questions that a human might ask it about the scene: “is there a blue triangle in the scene?”, “what is the color of the mug?”, “how many objects are there in the scene?”. In order to answer these questions, the robot has at its disposal a range of information processing functions and sensing actions. But, in any reasonably complex sce-



Figure 1: A picture of the typical table top scenario—ROIs bounded by rectangular boxes.

nario (such as the one described above), it is not feasible (and definitely not efficient) for the robot to run all available information processing functions and sensing actions, especially since the cognitive robot system needs to respond to human queries/commands in real-time.

The remainder of the paper is organized as follows: we first pose the problem in two ways: as a probabilistic planning problem, and as a continual planning problem. We show how some visual operators with relatively simple effects can be modeled using planning operators in each framework. We also show how to make the POMDP approach more tractable by defining a hierarchical POMDP planning system called HiPPo. After presenting the results of an empirical study, we briefly describe some related work and discuss future research directions.

## Problem Formulation

In this section, we propose a novel hierarchical POMDP planning formulation. We also briefly present the Continual Planning (CP) framework with which we compare our method. For ease of understanding, we use the example of an input image from the table-top scenario that is pre-processed to yield two regions of interest (ROIs), i.e. two rectangular image regions that are different from a previously trained model of the background. Figure 1 shows an example 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 set of visual actions/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, and a *sift* operator (Lowe (2004)) to detect the presence of one of the previously trained object models. The goal is to plan a sequence of visual actions that would answer the query with high confidence. Throughout this paper, we use the following terms interchangeably: visual processing actions, visual actions, and visual operators.

## A Hierarchical POMDP

In robot applications, typically the true state of the world cannot be observed directly. The robot can only revise its belief about the possible current states by executing actions, for instance one of the visual operators.

We pose the problem as an instance of probabilistic sequential decision making, and more specifically as a Partially Observable Markov Decision Process (POMDP)

where we explicitly model the unreliability of the visual operators/actions. This probabilistic formulation enables the robot to maintain a probability distribution (the *belief state*) over the true underlying state. To do this we need an observation model that captures the likelihood of the outcomes from each action. In this paper, we only consider actions that have purely informational effects. In other words, we do not consider actions such as poking the object to determine its properties, with the consequence that the underlying state does not change when the actions are applied. However, the POMDP formulation allows us to do this, which is necessary if we wish to model the effects of operators that split ROIs, move the camera, or move the objects to gain visual information about them.

Each action considers the true underlying state to be composed of the normal 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 object/valid class—*empty* (*E*), and a label to denote the presence of *multiple* classes (*M*). The observation model for each action provides a probability distribution over the set composed of the normal class labels, the class label *empty* (*E*) that implies that the match probability corresponding to the normal class labels is very low, and *unknown* (*U*) that means that there is no single class label to be relied upon and that multiple classes may therefore be present. Note that *U* is an observation, whereas *M* is part of the underlying state: they are not the same, since they are not perfectly correlated.

Since visual operators only update belief states, we include “special actions” that answer the query by “saying” (not to be confused with language-based communication) which underlying state is most likely to be the true state. Such actions cause a transition to a terminal state where no further actions are applied. In the description below, for ease of explanation (and without loss of generality) we only consider two operators: *color* and *shape*, denoting them with the subscripts *c*, *s* respectively. States and observations are distinguished by the superscripts *a*, *o* respectively.

Consider a single ROI in the scene—it has a POMDP associated with it for the goal of answering a specific query. This POMDP is defined by the tuple  $\langle S, \mathcal{A}, T, \mathcal{Z}, \mathcal{O}, \mathcal{R} \rangle$ :

- $S : \mathcal{S}_c \times \mathcal{S}_s \cup \text{term}$ , the set of states, is a cartesian product of the state spaces of the individual actions. It also includes a *terminal* state (term).  $\mathcal{S}_c : \{E_c^a, R_c^a, G_c^a, B_c^a, M_c\}$ ,  $\mathcal{S}_s : \{E_s^a, C_s^a, T_s^a, S_s^a, M_s\}$
- $\mathcal{A} : \{\text{color}, \text{shape}, s\text{Red}, s\text{Green}, s\text{Blue}\}$  is the set of actions. The first two entries are the visual processing actions. The rest are special actions that represent responses to the query such as “say blue”, and lead to *term*. Here we only specify “say” actions for color labels, but others may be added trivially.
- $T : S \times \mathcal{A} \times S \rightarrow [0, 1]$  represents the state transition function. For visual processing actions it is an identity matrix, since the underlying state of the world does not change when they are applied. For special actions it represents a transition to *term*.
- $\mathcal{Z} : \{E_c^o, R_c^o, G_c^o, B_c^o, U_c, E_s^o, C_s^o, T_s^o, S_s^o, U_s\}$  is the set of observations, a concatenation of the observations for

each visual processing action.

- $\mathcal{O} : \mathcal{S} \times \mathcal{A} \times \mathcal{Z} \rightarrow [0, 1]$  is the observation function, a matrix of size  $|\mathcal{S}| \times |\mathcal{Z}|$  for each action under consideration. It is learned by the robot for the visual actions (described in the next section), and it is a uniform distribution for the special actions.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ , specifies the reward, mapping from the state-action space to real numbers. In our case:

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

For visual actions, the cost depends on the size of the ROI (polynomial function of ROI size) and the relative computational complexity (the *color* operator is twice as costly as *shape*). For special actions, a large +ve (-ve) reward is assigned for making a right (wrong) decision for a given query. For e.g. while determining the ROI's color:

$\mathcal{R}(R_c^a T_s^a, \text{sRed}) = 100 \cdot \alpha$ ,  $\mathcal{R}(B_c^a T_s^a, \text{sGreen}) = -100 \cdot \alpha$  but while computing the location of *red* objects:

$\mathcal{R}(B_c^a T_s^a, \text{sGreen}) = 100 \cdot \alpha$ . The variable  $\alpha$  enables the trade-off between the computational costs of visual processing and the reliability of the answer to the query.

In the POMDP formulation, given the robot's belief state  $b_t$  at time  $t$ , the belief update proceeds as:

$$b_{t+1}(s') = \frac{\mathcal{O}(s', a_t, o_{t+1}) \sum_{s \in \mathcal{S}} \mathcal{T}(s, a_t, s') \cdot b_t(s)}{P(o_{t+1} | a_t, b_t)} \quad (1)$$

where  $\mathcal{O}(s', a_t, o_{t+1}) = P(o_{t+1} | s_{t+1} = s', a_t)$ ,  $b_t(s) = P(s_t = s)$ ,  $\mathcal{T}(s, a_t, s') = P(s_{t+1} = s' | a_t, s_t = s)$ , and  $P(o_{t+1} | a_t, b_t) = \sum_{s' \in \mathcal{S}} \{P(o_{t+1} | s', a_t) \cdot \sum_{s \in \mathcal{S}} P(s' | a_t, s) b_t(s)\}$  is the normalizer.

Our visual planning task for a single ROI now involves solving this POMDP to find a policy that maximizes reward from the initial belief state. Plan execution corresponds to traversing 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 getting a particular observation. In order to ensure that the observations are independent (required for Eqn 1 to hold), we take a new image of the scene if an action is repeated on the same ROI.

Actual scenes will have several objects and hence several ROIs. Attempting to solve a POMDP in the joint space of all ROIs soon becomes intractable due to an exponential state explosion, even for a small set of ROIs and actions. For a single ROI with  $m$  features (color, shape, etc.) each with  $n$  values, the POMDP has an underlying space of  $n^m$ ; for  $k$  ROIs the overall space becomes:  $n^{mk}$ . Instead, we propose a *hierarchical decomposition*: we model each ROI with a lower-level (LL) POMDP as described above, and use a higher-level (HL) POMDP to choose, at each step, the ROI whose policy tree (generated by solving the corresponding LL-POMDP) is to be executed. This decomposes the overall problem into one POMDP with state space  $k$ , and  $k$  POMDPs with state space  $n^m$ . For the example of two

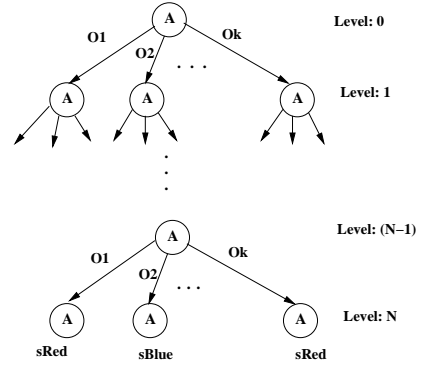


Figure 2: Policy Tree of an ROI—each node represents a belief state and specifies the action to take.

ROIs and the goal of finding the *blue* objects, the two LL-POMDPs are defined as above, and the HL-POMDP is given by:  $\langle \mathcal{S}^H, \mathcal{A}^H, \mathcal{T}^H, \mathcal{Z}^H, \mathcal{O}^H, \mathcal{R}^H \rangle$ , where:

- $\mathcal{S}^H = \{R_1 \wedge \neg R_2, \neg R_1 \wedge R_2, \neg R_1 \wedge \neg R_2, R_1 \wedge R_2\} \cup \text{term}^H$  is the set of states. It represents the presence/absence of the object in one or more of the ROIs, i.e.  $R_1 \wedge \neg R_2$  means the object is really exists in  $R_1$  but not in  $R_2$ . It also includes a terminal state ( $\text{term}^H$ ).
- $\mathcal{A}^H = \{u_1, u_2, A_S\}$  are the actions. The sensing actions ( $u_i$ ) denote the choice of executing one of the LL ROIs' policy trees. The special actions ( $A_S$ ) represent the fact of "saying" that one of the entries of  $\mathcal{S}^H$  is the answer, and they lead to  $\text{term}^H$ .
- $\mathcal{T}^H$  is the state transition function, which leads to  $\text{term}^H$  for special actions and is an identity matrix otherwise.
- $\mathcal{Z}^H = \{FR_1, \neg FR_1, FR_2, \neg FR_2\}$  is the set of observations. It represents the observation of finding/not-finding the object when each ROI's policy tree is executed.
- $\mathcal{O}^H : \mathcal{S}^H \times \mathcal{A}^H \times \mathcal{Z}^H \rightarrow [0, 1]$ , the observation function of size  $|\mathcal{S}^H| \times |\mathcal{Z}^H|$ , is an uniform matrix for special actions. For sensing actions, it is obtained from the policy trees for the LL-POMDPs as described below.
- $\mathcal{R}^H$  is the reward specification. It is a "cost" for each sensing action, which is computed as described below. For a special action, it is a large +ve value if it predicts the true underlying state correctly, and a large -ve value otherwise. For e.g.  $\mathcal{R}(R_1 \wedge R_2, sR_1 \wedge R_2) = 100$ , while  $\mathcal{R}(R_1 \wedge \neg R_2, sR_1 \wedge R_2) = -100$ .

The computation of the observation function and the cost/reward specification for each sensing action is based on the policy tree of the corresponding LL-POMDP. As seen in Figure 2, the LL-POMDP's policy tree has the root node representing 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 object being searched for (a *blue* object in this example) is "found" in  $R_1$  on executing the LL-POMDP's policy tree. The leaf nodes corresponding to the desired terminal action

(*sBlue*) are determined. The probability of ending up in each of these leaf nodes is computed by charting a path from the 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. For our example of searching for *blue* objects, this can be formally stated as:

$$P(FR_1) = \sum_{i_n \in \text{LeafNodes} | \text{action}(i_n) = s\text{Blue}} P(i_n | \pi_1, b_0) \quad (2)$$

$$P(i_n | \pi_1, b_0) = \Pi_{k=n}^1 P(i_k | \text{Parent}(i)_{k-1})$$

where,  $i_k$  denotes the node  $i$  at level  $k$ ,  $\text{Parent}(i)_{k-1}$  is the parent node of node  $i$  at level  $k-1$ , and  $\pi_1$  is the policy tree corresponding to the ROI  $R_1$ . We control computation by forcing the LL-POMDP's policy tree to terminate after  $N$  levels, set heuristically based on the query complexity ( $x = \text{no. of object features being analyzed}$ ):  $N_{\min} + k \cdot x$ , where all branches have to take a terminal action. The entries within the product term ( $\Pi$ ) are the normalizers of the belief update (Eqn 1):

$$P(i_k | \text{Parent}(i)_{k-1}) = \sum_{s' \in \mathcal{S}} \left\{ P(o_{\text{Parent}(i)_{k-1}}^{i_k} | s', a_{\text{Parent}(i)_{k-1}}) \cdot \sum_{s \in \mathcal{S}} P(s' | a, s) b_{\text{Parent}(i)_{k-1}}(s) \right\} \quad (3)$$

where  $o_{\text{Parent}(i)_{k-1}}^{i_k}$  is the observation that transitions to node  $i_k$  from its parent  $\text{Parent}(i)_{k-1}$ ,  $a_{\text{Parent}(i)_{k-1}}$  represents the action taken from the parent node, and  $b_{\text{Parent}(i)_{k-1}}(s)$  is the belief state at  $\text{Parent}(i)_{k-1}$ .

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

$$C_{\text{root}} = \sum_j P(j_1) \cdot C_{\text{root}}^{j_1} \cdot C_{j_1} \quad (4)$$

where  $C_{\text{root}}^{j_1}$  is the cost of performing the action at the root node that created the child node ( $j$ ) and  $P(j_1)$  is the transition probability from the root node to the child node at level 1 (Equation 3).  $C_{j_1}$  is the cost of the child node, computed by analyzing its children in a similar manner.

The traversal of the LL-POMDP's policy tree for the transition probability (and cost computation) for the HL-POMDP model is different from the belief update when generating/executing the policy at the LL. When  $\pi_1$  is evaluated for  $FR_1$ , we are computing the probability of finding the *blue* object in  $R_1$  *conditioned on the fact that a blue object actually exists in the ROI*, information that the LL-POMDP does not normally have. The observation functions are hence modified such that for each action, each row of the matrix is the weighted average of the rows corresponding to the states that can predict the target ("blue") property, the weights being the likelihood of the corresponding states in the initial belief. In our example the states:  $B_c^a E_s^a, B_c^a C_s^a, B_c^a T_s^a, B_c^a S_s^a, B_c^a M_s$  are equally

likely, while other states have zero probability in the initial belief. This change is *only* for building the HL-POMDP model—belief update in the LL-POMDPs proceeds with an uniform initial belief and an unmodified  $\mathcal{O}$ .

Once the observation functions and costs are computed, the HL-POMDP model can be built and solved to yield the HL policy. During execution, the HL-POMDP's policy is queried for the best action choice, which causes the execution of one of the LL-POMDP policies, resulting in a sequence of visual operators being applied on one of the image ROIs. The answer provided by the LL-POMDP's policy execution causes a belief update in the HL-POMDP, and the process continues until a terminal action is chosen at the HL, thereby answering the query posed. Here it provides the locations of all *blue* objects in the scene. For simpler occurrence queries (e.g. "Is there a blue object in the scene?") the execution can be terminated at the first occurrence of the object in a ROI. Both the LL and HL POMDPs are query dependent; we cannot solve them in advance and execute the policies during user interactions. Solving the POMDPs efficiently is hence crucial to overall performance.

## Continual Planning

The Continual Planning (CP) approach of Brenner and Nebel (2006) interleaves planning, plan execution and plan monitoring. Unlike classical planning approaches that require prior knowledge of state, action outcomes, and all contingencies, an agent in CP postpones reasoning about unknowable or uncertain states until more information is available. It achieves this 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, and if those preconditions are not met during execution (or are met earlier), replanning is triggered. But there is *no* representation of the uncertainty/noise in the observation/actions. It uses the PDDL (McDermott (1998)) syntax and is based on the FF planner of Hoffmann and Nebel (2001). Consider the example of a *color* operator:

```
(: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 parameters are a color-property (e.g. *blue*) being searched for in a particular ROI. It can be applied on any ROI that satisfies the precondition i.e. it has not already been analyzed. The expected result is that the desired color is found in the ROI. The "replan:" condition ensures that if the robot observes the ROI's color by another process, replanning is triggered to generate a plan that excludes this action. This new plan will use the *containsColor* fact from the new state instead. In addition, if the results of executing a plan step are not as expected, replanning (triggered by execution monitoring) ensures that other ROIs are considered. Other operators are defined similarly, and based on the goal state definition

the planner chooses the sequence of operators whose effects provide parts of the goal state—the next section provides an example. The CP approach to the problem is more responsive to an unpredictable world than a non-continual classical planning approach would be, and it can therefore reduce planning time in the event of deviations from expectations. But, while actions still have non-deterministic effects, there is no means for accumulating belief over successive applications of operators. We show that the HiPPo formulation provides significantly better performance than CP in domains with uncertainty.

## Experimental Setup and Results

In this section, we describe how the LL-POMDP observation functions are learned, followed by an example of the planning-execution cycle and a quantitative comparison of the planning approaches.

### Learning Observation Functions

The model creation in the LL-POMDPs requires observation and reward/cost functions for the sensing/information actions. Unlike POMDP-based applications where the reward and observation matrices are manually specified (Pineau and Thrun (2002)), we aim to model the actual conditions by having the robot learn them in advance, i.e. before actual planning and execution. Objects with known labels (“red circular mug”, “blue triangle” etc) are put in front of the robot, and the robot executes repeated trials to estimate the probability of various observations (*empty*, class labels and *unknown*) for each action, given the actual state information (*empty*, class labels and *multiple*). We assume here that the observations are independent, and are produced by different actions. The costs of the operators capture the relative run-time complexity of the operators computed through repeated trials over ROIs. The cost is also a function of the ROI size but the objects are approximately the same size in our experiments. In addition, a model is learned for the scene background in the absence of the objects to be analyzed. This background model is used during online operation to generate the ROIs by background subtraction. Other sophisticated techniques such as saliency computation (Itti, Koch, and Niebur (1998)) may be used to determine the ROIs, but the background subtraction method suffices for our problem domain and it is computationally more efficient.

### Experimental Setup and Example

The experimental setup is as follows. The camera mounted on a robot captures images of a tabletop scene. Any change from the known background model is determined and the ROIs are extracted by background subtraction. The goal is to choose a sequence of operators that when applied on the scene can provide an answer to the query. For these experiments we assume the robot can choose from *color*, *shape* and *sift*, and we explain the execution with the sample query: *Where is the blue circle?*

In the HiPPo approach, the robot creates a LL-POMDP model file for each ROI, based on the available visual operators and the query that has been posed. The model file

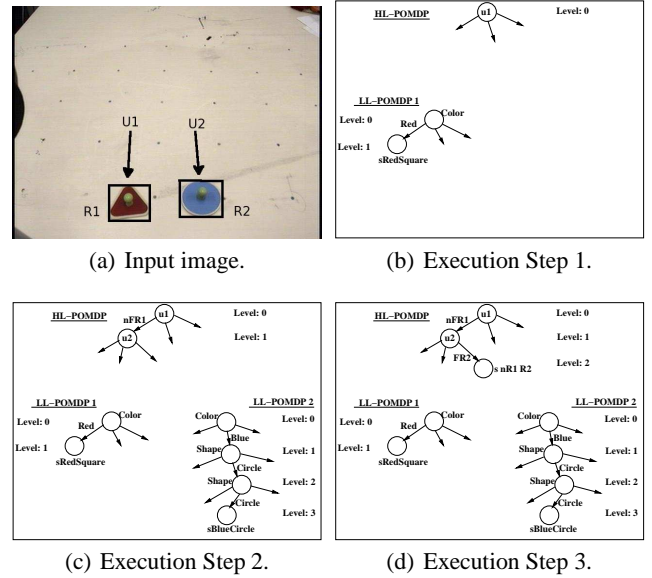


Figure 3: Example query: “Where is the Blue Circle?” Dynamic reward specification in the LL-POMDP allows for early termination when negative evidence is found.

is in the format required by the ZMDP planning package<sup>1</sup>. The point-based solver of Smith and Simmons (2005) in the same package is used to determine the LL-policies. As described earlier, the policy trees of the ROIs are evaluated with the modified belief vector and observation functions to determine the observation functions and costs of the HL-POMDP. The resulting HL model file is in turn solved to get the HL policy. Figs 3(a)-3(d) show one execution example for an image with two ROIs.

The example query is to determine the presence and location of one or more *blue circles* in the scene (Fig 3(a)). Since both ROIs are equally likely target locations, the HL-POMDP first chooses to execute the policy tree of the first ROI (action  $u_1$  in Fig 3(b)). The corresponding LL-POMDP runs the color operator on the ROI. The outcome of applying any individual operator is the observation with the maximum probability, which is used to update the subsequent belief state—in this case the answer is *red*. Even though it is more costly, the color operator is applied before shape because it has a higher reliability, based on the learned observation functions. When the outcome of *red* increases the likelihood (belief) of the states that represent the “Red” property as compared to the other states, the likelihood of finding a blue circle is reduced significantly. The dynamic reward specification ( $\alpha = 0.2$ ) ensures that without further investigation (for instance with a shape operator), the *best* action chosen at the next level is a terminal action associated with not finding the target object—here it is *sRedSquare*. The HL-POMDP receives the input that a red square is found in  $R_1$ , leading to a belief update and subsequent action selection (action  $u_2$  in Fig 3(c)). Then the policy tree of the LL-POMDP of  $R_2$  is invoked, causing the color and shape operators to be

<sup>1</sup>See [www.cs.cmu.edu/~trey/zmdp/](http://www.cs.cmu.edu/~trey/zmdp/)

applied in turn on the ROI. The higher noise in the shape operator is the reason why it has to be applied twice before the uncertainty is sufficiently reduced to cause the choice of a terminal action (*sBlueCircle*)—the increased reliability therefore comes at the cost of execution overhead. This results in the belief update and terminal action selection in the HL-POMDP—the final answer is  $(s \neg R_1 \wedge R_2)$ , i.e. that a *blue circle* exists in  $R_2$  and not  $R_1$  (Fig 3(d)).

In our HiPPo representation, each HL-POMDP action chooses to execute the policy of one of the LL-POMDPs until termination, instead of performing just one action. The challenge here is the difficulty of translating from the LL belief to the HL belief in a way that can be planned with. The execution 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 can only increase the posterior of the ROI currently being explored, so if the HL-POMDP were to choose the next action, it would choose to explore the same ROI again.

If the same problem were to be solved with the CP approach, the goal state would be 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 a ROI which has the color *blue* and shape *circle*. The planner must then find a sequence of operators to satisfy the goal state. In this case it leads to the creation of the plan:

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

i.e. the robot 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 (due to image noise) an operator determines a wrong class label as the closest match with a low probability, there is no direct mechanism to incorporate the knowledge. Any thresholds will have to be carefully tuned to prevent mis-classifications. Assuming that the color operator works correctly in this example, it would classify the ROI as being *red*, which would be determined in the plan monitoring phase. Since the desired outcome (finding *blue* in the first ROI) was not achieved, replanning is triggered to create a new plan with the same steps, but to be applied on the second ROI. This new plan leads to the desired conclusion of finding the *blue circle* in  $R_2$  (assuming the operators work correctly).

## Quantitative comparison

The hypotheses we aim to test are as follows:

- Hierarchical-POMDP planning (HiPPo) formulation is more efficient than the standard POMDP formulation.
- HiPPo and CP have comparable plan-time complexity.
- Planning is significantly more efficient than blindly applying all operators on the scene.
- HiPPo has higher execution time than CP but provides more reliable results.

In order to test these hypotheses we ran several experiments on the robot in the tabletop scenario. Objects were

placed on the table and the robot had to analyze the scene to answer user-provided queries. Query categories include:

- Occurrence queries: Is there a red mug in the scene?
- Location queries: Where in the image is the blue circle?
- Property queries: What is the color of the box?
- Scene queries: How many green squares are there in the scene?

For each query category, we ran experiments over  $\sim 15$  different queries with multiple trials for each such query, thereby representing a range of visual operator combinations in the planning approaches. We also repeated the queries for different numbers of ROIs in the image. In addition, we also implemented the naive approach of applying all available operators (color, shape and sift in our experiments) on each ROI, until a ROI with the desired properties is found and/or all ROIs have been analyzed.

Unlike the standard POMDP solution that considers the joint state space of several ROIs, the hierarchical representation does not provide the optimal solution (policy). Executing the hierarchical policy may be arbitrarily worse than the optimal policy. For instance, in the search for the *blue* region, the hierarchical representation is optimal *iff* every ROI is blue-colored. But as seen in Figure 4(a) that compares the planning complexity of HiPPo with the standard POMDP solution, the non-hierarchical approach soon becomes intractable. The hierarchical approach provides a significant reduction in the planning time and (as seen below) still increases reliability significantly.

Next, we compare the planning times of HiPPo and CP approaches as a function of the number of ROIs in the scene—Figure 4(b). The standard hierarchical approach takes more time than CP. But, the computationally intensive part of HiPPo is the computation of the policies for the ROIs. Since the policies computed for a specific query are essentially the same for all scene ROIs, they can be cached and not repeated for every ROI. This simple adjustment drastically reduces the planning time and makes it comparable to the CP approach.

Figure 4(c) compares the execution time of the planning approaches against applying all the operators on each ROI until the desired result is found. The HiPPo approach has a larger execution time than CP because it may apply the same operator multiple times to a single ROI (in different images of the same scene) in order to reduce the uncertainty in its belief state. In all our experiments the algorithms are being tested on-board a cognitive robot which has multiple modules to analyze input from different modalities (vision, tactile, speech) and has to bind the information from the different sources. Hence, though the individual actions are optimized and represent the state-of-the-art in visual processing, they take some time to execute on the robot.

A key goal of our approach is not only to reduce overall planning and execution time, but to improve the reliability of the visual processing. In these terms the benefits are very clear, as can be seen in Table 1. The direct application of the actions on all the ROIs in the scene results in an average classification accuracy of 76.67%, i.e. the sensing actions



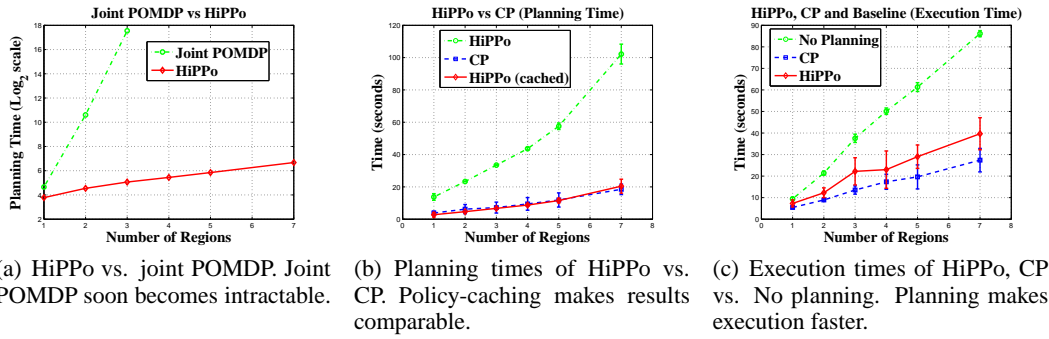


Figure 4: Experimental Results—Comparing planning and execution times of the planners against no planning.

Approach	% Reliability
Naive	76.67
CP	76.67
HiPPo	91.67

Table 1: Reliability of visual processing

misclassify around one-fourth of the objects. Using CP also results in the same accuracy of 76.67%, i.e. it *only reduces the execution time* since there is no direct mechanism in the non-probabilistic planner to exploit the outputs of the individual operators (a distribution over the possible outcomes). The HiPPo approach is designed to utilize these outputs to reduce the uncertainty in belief, and though it causes an increase in the execution time, it results in much higher classification accuracy: 91.67%. It is able to recover from a few noisy images where the operators are not able to provide the correct class label, and it fails only in cases where there is consistent noise. A similar performance is observed if additional noise is added during execution. As the non-hierarchical POMDP approach takes days to compute the plan for just two ROIs we did not compute the optimal plan for scenes with more than two ROIs, but for the cases where a plan was computed, there was no significant difference between the optimal approach and HiPPo in terms of the execution time and reliability, even though the policy generated by HiPPo can be arbitrarily worse than that generated by the non-hierarchical approach.

A significant benefit of the POMDP approach is that it provides a ready mechanism to include initial belief in decision-making. For instance, in the example considered above, if  $R_2$  has a higher initial belief that it contains a *blue circle*, then the cost of executing that ROI's policy would be lower and it would automatically get chosen to be analyzed first leading to a quicker response to the query.

Figure 5 shows a comparison of the combined planning and execution times for HiPPo, CP, and the naive approach of applying all actions in all ROIs (no planning). As the figure shows, planning is worthwhile even on scenes with only two ROIs. In simple cases where there are only a couple of operators and/or only one operator for each feature (color, shape, object recognition etc) one may argue that rules may be written to decide on the sequence of operations. But as soon as the number of operators increase and/or there

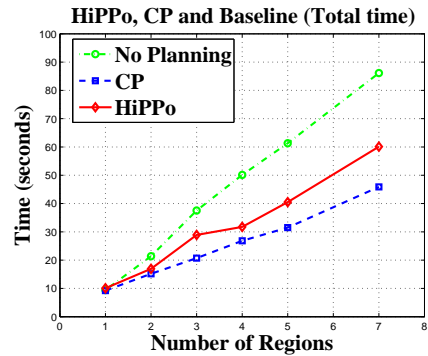


Figure 5: Planning+execution times of HiPPo, CP vs. No planning. Planning approaches reduce processing time.

is more than one operator for each feature (e.g. two actions that can find color in a ROI, each with a different reliability), planning becomes an appealing option.

## Related Work

### Previous work on Planning Visual Processing

There is a significant body of work in the image processing community on planning sequences of visual operations. The high level goal is specified by the user, and is used by a classical AI planner to construct a pipeline of image processing operations. The planners use deterministic models of the effects of information processing: handling the pre-conditions and the effects of the operators using propositions that are required to be true a priori, or are made true by the application of the operator. Uncertainty is handled by evaluating the output images using hand-crafted evaluation rules (Clouard et al. (1999); Thonnat and Moisan (2000); Moisan (2003)). If the results are unsatisfactory, execution monitoring detects this and the plan is repaired. This either involves re-planning the type and sequence of operators or modification of the parameters used in the operators. Example domains include astronomy (Chien, Fisher, and Estlin (2000)) and bio-medical applications (Clouard et al. (1999)). There has also been some work on perception for autonomous object detection and avoidance in vehicles (Shekhar, Moisan, and Thonnat (1994)) but extensions to more general computer vision has proven difficult. Recent work (Li et al. (2003)) has mod-

eled image interpretation as a MDP (Markov Decision Process). Here, human-annotated images are used to determine the reward structure, and to explore the state space to determine dynamic programming based value functions that are extrapolated (to the entire state space) using the ensemble technique called leveraging. Online image interpretation involves the choice of action that maximizes the value of the learned value functions at each step.

In real-world applications, the true state of the system is not directly observable. We model the probabilistic effects of operators on the agent's beliefs, and use the resulting probability distributions for a more generic evaluation.

### Observation Planners

The PKS planner (Petrick and Bacchus (2004)) uses actions which are described in terms of their effect on the agent's knowledge, rather than their effect on the world, using a first order language. Hence the model is 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 will not know if it is broken, and 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 problem domain, we could say that the objects in the query are in one of the regions of interest, but that we do not know which one. The planner will then plan to use the observational actions to examine each region, branching based on what is discovered.

The Continual Planning (CP) approach of Brenner and Nebel (2006) that we have compared against is quite similar to PKS in its representation, but works by replanning rather than constructing conditional plans. As we have said, in applications where observations are noisy, the optimal behaviour may be to take several images of a scene and run the operators more than once to reduce uncertainty. This cannot be represented in either PKS or CP.

### POMDP Solvers

The POMDP formulation of Kaelbling, Littman, and Cassandra (1998) is appropriate for domains where the state is not directly observable, and the agent's actions update its belief distribution over the states. Our domain is a POMDP where the underlying state never changes; actions only change the belief state. But the state space quickly grows too large to be solved by conventional POMDP solvers as the number of ROIs increases. To cope with large state spaces in POMDPs, Pineau and Thrun (2002) propose a hierarchical POMDP approach for a nursing assistant robot, similar to the MAXQ decomposition for MDPs of Dietterich (1998). They impose an action hierarchy, with the top level action being a collection of simpler actions that are represented by smaller POMDPs and solved completely; planning happens in a bottom-up manner. Individual policies are combined to provide the total policy. When the policy at the top-level task is invoked, it recursively traverses the hierarchy invoking sequence of local policies until a primitive action is reached. All model parameters at all levels are defined over the same

state-action-observation space, but the relevant space is abstracted for each POMDP using a dynamic belief network. In the actual application, a significant amount of data for the hierarchy and model creation is hand-coded.

Hansen and Zhou (2003) propose a manually specified task hierarchy for POMDP planning. Though similar to Pineau's work in terms of the bottom-up planning scheme, each policy is represented as a finite-state controller (FSC), and each POMDP in the hierarchy is an indefinite-horizon POMDP that allows FSC termination without recognition of the underlying terminal state. In addition, they use policy iteration instead of value iteration to solve POMDPs. They show that this representation guarantees policy quality. More recent work by Toussaint, Charlin, and Poupart (2008) proposes maximum likelihood estimation for hierarchy discovery in POMDPs, using a mixture of dynamic Bayesian networks and EM-based parameter estimation.

We propose a hierarchy in the (image) state and action space. Instead of manually specifying the hierarchy, abstractions and the model parameters across multiple levels, our hierarchy only has two levels. At the lower level (LL), each ROI is assigned a POMDP, whose action (and state) space depends on the query posed. The visual processing actions are applied in the LL. The approximate (policy) solutions of the LL-POMDPs are used to populate a higher level (HL) POMDP that has a completely different state, action and observation space. The HL POMDP acts as the controller: it maintains the belief over the entire scene/image and chooses the best ROI for further processing by executing the corresponding LL policy, thereby providing answers to the queries posed. Hence a simple hierarchy structure can be used unmodified for a range of queries in our application domain. Furthermore, all reward and observation models are learned: in the LL the robot autonomously collects statistics based on repeated applications of the visual operators, and in the HL it is learned based on the LL policies.

### Conclusions and Future Work

Robots working on cognitive tasks in the real world need the ability to tailor their visual 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, which when applied on an input scene enable it to determine the answer to a user-provided query with high confidence. We have compared the performance of our approach with a representative modern planning framework (continual planning) on a real robot application. Both planning approaches provide significant benefits over direct application of all available visual operators. In addition, the probabilistic approach is better able to exploit the output information from individual actions in order to reduce the uncertainty in decision-making. Still we have but opened up an interesting direction of further research, and there are several challenges left to address.

Currently we are dealing with a relatively small set of operators and ROIs in the visual scene. With a large number of operators the solution to the LL-POMDPs may prove to be expensive. Similarly, once the number of regions increase, finding the solution to the HL-POMDP may also become



computationally expensive. In such cases, it may be necessary to implement a range of hierarchies (e.g. Pineau and Thrun (2002)) in both the action and state spaces. A further extension would involve learning the hierarchy itself (Toussaint, Charlin, and Poupart (2008)).

In our current implementation, the LL-POMDP policies are evaluated up to a certain number of levels before control returns to the HL-POMDP. One direction of further research is to exploit the known information to make the best decision at every step. This would involve learning (or evaluating during run-time) the observation functions for the HL-POMDP for partial executions of the LL policies.

We would also like to extend this planning framework for more complex queries, such as “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?). For such queries, we aim to investigate the use a combination of a probabilistic and a deterministic planner. For known facts about the world (e.g. relationships such as “left of”) and binding information across different modalities (say tactile and visual), we could use the deterministic planning approach, while the probabilistic framework could be employed for sensory processing. Furthermore, the current planning framework can be extended to handle actions that change the visual input, for instance a viewpoint change to get more information or an actual manipulation action that grasps and moves objects from one point to another (and hence changes the state). Eventually the aim is to enable robots to use a combination of learning and planning to respond autonomously and efficiently to a range of tasks.

### Acknowledgements

Special thanks to Nick Hawes for providing feedback on an initial draft. We are grateful that this work was supported by the EU FP6 IST Cognitive Systems Integrated Project (CoSy) FP6-004250-IP, the Leverhulme Trust Research Fellowship Award Leverhulme RF/2006/0235 and the EU FP7 IST Project CogX FP7-IST-215181.

### References

- Brenner, M., and Nebel, B. 2006. Continual Planning and Acting in Dynamic Multiagent Environments. In *The International PCAR Symposium*.
- Chien, S.; Fisher, F.; and Estlin, T. 2000. Automated software module reconfiguration through the use of artificial intelligence planning techniques. *IEE Proc. Software* 147.
- Clouard, R.; Elmoataz, A.; Porquet, C.; and Revenu, M. 1999. Borg: A knowledge-based system for automatic generation of image processing programs. *PAMI* 21.
- Dietterich, T. 1998. The MAXQ Method for Hierarchical Reinforcement Learning. In *ICML*.
- Hansen, E. A., and Zhou, R. 2003. Synthesis of Hierarchical Finite-State Controllers for POMDPs. In *ICAPS*, 113–122.
- Hawes, N.; Sloman, A.; Wyatt, J.; Zillich, M.; Jacobsson, H.; Kruiff, G.-J. M.; Brenner, M.; Berginc, G.; and Skocaj, D. 2007. Towards an Integrated Robot with Multiple Cognitive Functions. In *The Twenty-second National Conference on Artificial Intelligence (AAAI)*.
- Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research* 14:253–302.
- Horswill, I. 1993. Polly: A Vision-Based Artificial Agent. In *AAAI*, 824–829.
- Itti, L.; Koch, C.; and Niebur, E. 1998. A Model of Saliency-Based Visual Attention for Rapid Scene Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20(11):1254–1259.
- Kaelbling, L.; Littman, M.; and Cassandra, A. 1998. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence* 101:99–134.
- Land, M. F., and Hayhoe, M. 2001. In What Ways do Eye Movements Contribute to Everyday Activities. *Vision Research* 41:3559–3565.
- Lee, D. N. 1980. The Optical Flow-field: The Foundation of Vision. *Philosophical Transactions of the Royal Society London B* 290:169–179.
- Li, L.; Bulitko, V.; Greiner, R.; and Levner, I. 2003. Improving an Adaptive Image Interpretation System by Leveraging. In *Australian and New Zealand Conference on Intelligent Information Systems*.
- Lowe, D. 2004. Distinctive Image Features from Scale-Invariant Keypoints. *IJCV*.
- McDermott, D. 1998. PDDL: The Planning Domain Definition Language, Technical Report TR-98-003/DCS TR-1165. Technical report, Yale Center for Computational Vision and Control.
- Moisan, S. 2003. Program supervision: Yakl and pegase+ reference and user manual. Rapport de Recherche 5066, INRIA, Sophia Antipolis, France.
- Petrack, R., and Bacchus, F. 2004. Extending the Knowledge-Based approach to Planning with Incomplete Information and Sensing. In *ICAPS*, 2–11.
- Pineau, J., and Thrun, S. 2002. High-level Robot Behavior Control using POMDPs. In *AAAI*.
- Shekhar, C.; Moisan, S.; and Thonnat, M. 1994. Use of a real-time perception program supervisor in a driving scenario. In *Intelligent Vehicle Symposium*.
- Smith, T., and Simmons, R. 2005. Point-based POMDP Algorithms: Improved Analysis and Implementation. In *UAI*.
- Thonnat, M., and Moisan, S. 2000. What can program supervision do for program reuse? *IEE Proc. Software*.
- Toussaint, M.; Charlin, L.; and Poupart, P. 2008. Hierarchical POMDP Controller Optimization by Likelihood Maximization. In *UAI*.