In *The 27th PlanSIG Workshop* (PlanSIG 08),
Edinburgh, UK, December 11-12, 2008.

E-HiPPo: Extensions to Hierarchical POMDP-based Visual Planning on a Robot

Mohan Sridharan
Department of Computer Science
Texas Tech University, USA
mohan.sridharan@ttu.edu

Richard Dearden and **Jeremy Wyatt**
School of Computer Science
University of Birmingham, UK
{rwd,jlw}@cs.bham.ac.uk

Abstract

One major challenge to the widespread deployment of mobile robots is the ability to autonomously tailor the sensory processing to the task on hand. In our prior work (Sridharan, Wyatt, and Dearden 2008), we proposed an approach for such general-purpose processing of visual input in an application domain where a robot and a human jointly converse about and manipulate objects on a tabletop by processing the regions of interest (ROIs) in input images. We posed the visual processing management problem as a partially observable Markov decision problem (POMDP), and introduced a hierarchical decomposition to make it tractable to plan with POMDPs. In this paper we analyze and eliminate some of the limitations of the existing approach. First, in addition to tackling visual actions that analyze the state of the world represented by the image, we show how to incorporate actions that can change the state. Secondly, we show how policy caching can be used to speed the planning performance and analyse the tradeoff between planning speed and plan quality.

Introduction

The availability of high-fidelity sensors such as color cameras at moderate costs has led to the use of robots in a range of applications (Minten et al. 2001; Pineau et al. 2003; Thrun 2006), but the ability to accurately sense and interact with the environment is still missing. One major requirement for the widespread deployment of mobile robots is the ability to autonomously tailor the sensory processing to the task on hand. Most current robot systems equipped with visual sensors are designed for specific domains using a manually chosen set of visual operators/processing routines. However, visual processing in robot domains is characterized by non-deterministic actions (visual operators are not completely reliable), high computational complexity and partial observability—the robot cannot observe the true state of the world, but it can update its *belief* about the state of the world by applying actions and observing the outcomes. At the same time the robot has to respond to dynamic environmental changes and operate with a high degree of reliability.

A robot equipped with a visual sensor therefore needs an efficient strategy to decide which visual processing, on which image regions, needs to be performed so as to accomplish the desired task. We investigate visual processing management as a planning problem where given a task the robot has to autonomously infer the sequence of information processing (*what to look for?*) and sensing actions (*where to look?*) that would maximize the reliability while using the available resources in an optimal manner. Though planning of visual operators has been extensively researched (Clouard et al. 1999; Thonnat and Moisan 2000; Moisan 2003; Li et al. 2003), prior work fails to address all the requirements of the problem because they typically are used for single images, require extensive domain knowledge to perform plan repair, have only been extended to robot systems in limited ways, and/or pose the problem in an essentially deterministic or observable framework (Li et al. 2003).

In recent work (Sridharan, Wyatt, and Dearden 2008) we posed the planning problem in a sequential decision making framework, and more specifically as a partially observable Markov decision problem (POMDP) (Kaelbling, Littman, and Cassandra 1998). We explicitly modelled the characteristic features of the problem, including the uncertainty in the visual operator outcomes. We posed the problem as one of applying visual operators to images that had already been segmented into regions of interest (ROIs) that are distinct from the background and hopefully contain the objects in the scene. We introduced a hierarchical decomposition to address the intractability of our POMDP representation. The resulting hierarchical POMDP planning system, called *HiPPo* provided significantly higher reliability in comparison to deterministic planners.

An important characteristic of the domain is that often two or more objects in the images overlap, resulting in visual operators such as shape detectors returning *multiple* as their outcome, instead of the individual shapes they have been trained on. In these cases our previous approach would perform very poorly. We address the problem of overlapping objects by adding an operator that splits a ROI into a set of sub-regions that hopefully contain distinct objects. The challenge with splitting a region is that it results in a change in the size of the state space of the POMDP. In this paper we extend our existing hierarchical POMDP planning framework to handle these splitting operators.

The planning time of our hierarchical approach is somewhat slower than the classical planning approach we compared it with, the Continual Planning (CP) approach of (Brenner and Nebel 2006). This is in part due to the fact that our hierarchical approach involves solving a POMDP

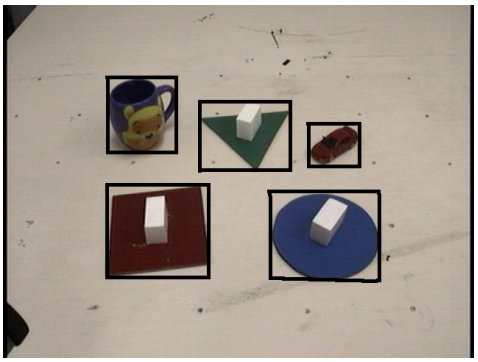


Figure 1: A picture of the typical table top scenario—regions-of-interest (ROIs) bounded by rectangular boxes.

for each ROI in the image, plus one additional POMDP. In this paper we investigate the caching of policies computed for each ROI, and reusing them to reduce the planning time. Policy-caching introduces an error in the computed expected reward for the policy, and we provide a theoretical and empirical evaluation of how much worse than optimal the resulting policy could be.

As our experimental testbed, we use the same domain that was used in our recent work (Sridharan, Wyatt, and Dearden 2008), that of a robot and a human jointly conversing about and manipulating objects on a tabletop, within the EU funded Cognitive Systems (CoSy) project (Hawes et al. 2007). Consider the scene in Figure 1, and the types of visual operations that the robot would need to perform to answer a variety of questions about the scene: "is there a blue triangle in the scene?", "what is the color of the mug?". In order to answer these questions, the robot has at its disposal a range of information processing functions and sensing actions. But it is not feasible for the robot to run all available actions, especially since the cognitive robot system needs to respond to queries/commands dynamically.

The remainder of the paper is organized as follows: we begin with an overview of the hierarchical POMDP framework. Then we introduce the changes required to incorporate region splitting, followed by the evaluation of the approximation errors incurred by policy-caching in the hierarchical decomposition. We conclude with a brief description of related methods and future research directions.

The Hierarchical POMDP Planner

The robot needs to process input images with the regions-of-interest (ROIs) extracted from the background. It maintains a probability distribution (*belief state*) over the true underlying state. For ease of understanding, we use the example of an input image from the table-top scenario that is preprocessed to yield two ROIs, i.e. two rectangular regions that are different from a previously trained background model.

Consider the query: "which objects in the scene are blue?" The goal is to plan a sequence of visual actions that would answer the query with high confidence. 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, based on the SIFT features developed by David Lowe (2004), to detect the presence of one of the previously trained object models. Throughout this paper, we use the following terms interchangeably: visual processing actions, visual actions, and visual operators.

Belief state maintenance requires a suitable model for the action outcomes. In our work, 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. U is an observation, whereas M is part of the underlying state: they are not the same.

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.

For a single ROI in the image, the POMDP 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, the set of states, is a cartesian product of the state spaces of the individual actions. It also includes a *terminal* state (term). $\mathcal{S}_c$: $\{E_c^a, R_c^a, G_c^a, B_c^a, M_c\}$, $\mathcal{S}_s : \{E_s^a, C_s^a, T_s^a, S_s^a, M_s\}$
- $\mathcal{A}$: $\{color, shape, \mathcal{A}_\mathcal{S}\}$ is the set of actions. The first two entries are the processing actions. The rest are special actions ($\mathcal{A}_\mathcal{S} = \{sRed, sGreen, sBlue\}$) that represent query responses such as "say blue", and lead to *term*. Though we only specify "say" actions for color labels, others may be added trivially.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ represents the state transition function. For visual processing actions that do not change the state, such as $color$ and $shape$, it is an identity matrix. 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. It is learned by the robot for the visual actions and it is a uniform distribution for the special actions.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$, specifies the reward, mapping from the state-action space to real numbers. In our case:

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

For visual actions, the cost depends on the size of the ROI and the relative computational complexity ($color$ is twice as costly as $shape$). For special actions, a large positive (negative) reward is assigned for making a right (wrong) decision for a given query. The variable α trades-off between computational costs and reliability of response.

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. But, for a single ROI with m features (color, shape etc.) each with n values (e.g. R, G for color), the POMDP has an underlying space of $n^m + 1$; for k ROIs the overall space is: $n^{mk} + 1$, so the problem soon becomes too large to solve even with state-of-the-art POMDP solvers.

We overcome the exponential state space explosion problem by imposing an intuitive *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. The overall problem is then decomposed into one POMDP with state space $2^k + 1$, and k POMDPs with state space $n^m + 1$. For the two-ROI example with the goal of finding the *blue* objects, the HL-POMDP is given by $\langle \mathcal{S}^H, \mathcal{A}^H, \mathcal{T}^H, \mathcal{Z}^H, \mathcal{O}^H, \mathcal{R}^H \rangle$:

- $\mathcal{S}^H = \{R_1 \wedge \neg R_2, \neg R_1 \wedge R_2, \neg R_1 \wedge \neg R_2, R_1 \wedge R_2\} \cup \text{term}^H$ is the set of states. It represents the presence/absence of the object in one or more of the ROIs, and includes a terminal state ($term^H$).
- $\mathcal{A}^H = \{u_1, u_2, \mathcal{A}_S^H\}$ are the actions. The sensing actions (u_i) denote the choice of executing one of the LL ROIs' policy trees. The special actions ($\mathcal{A}_S^H$) represent "saying" actions, and they lead to $term^H$.
- $\mathcal{T}^H$ is the state transition function that leads to $term^H$ 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, which represents finding/not-finding the desired object when each ROI's (LL) policy is executed.
- $\mathcal{O}^H : \mathcal{S}^H \times \mathcal{A}^H \times Z^H \rightarrow [0, 1]$, the observation function, is an uniform matrix for special actions. For sensing actions, it is learned from the LL-POMDP policy trees.
- $\mathcal{R}^H$ is the reward specification. It is a "cost" for each sensing action, computed from the LL policy trees. For a special action, it is a large positive value if it predicts the state correctly, and a large negative value otherwise.

During the creation of the HL-POMDP (and during plan execution) we control computational complexity by forcing the LL-POMDP's policy tree to terminate after N levels, set heuristically based on the query complexity. Though the LL policies are computed until completion (up to the desired regret bounds), all branches have to take a terminal action after N steps in the tree.

The computation of the observation functions and cost/reward specification for the HL-POMDP, based on the corresponding LL-POMDP policy trees, is an important part of the hierarchical decomposition—details and an experimental comparison with Continual Planning (CP) are in our paper (Sridharan, Wyatt, and Dearden 2008). HiPPo is significantly more reliable (91.67%) than CP (76.67%) or no planning (76.67%) i.e. applying all visual actions on the scene. Since CP does not explicitly model the uncertainty of action outcomes, it cannot do any better than naive processing in terms of reliability.

Handling Regions with Overlapping Objects

As we said above, we use background subtraction to generate the initial ROIs for analysis. In practice, this frequently results in objects that are close together or overlap being placed in a single ROI. In these cases, the vision algorithms are likely to return *unknown*, indicating that multiple classes may be present in the ROI. The question we address in this section is what to do when this occurs.

To handle cases where multiple objects appear in a single ROI we use "region-splitting" actions that segment the ROI based on one of the underlying features, for instance $color$ or $shape$. As an example, consider the splitting action based on the color feature, i.e. $rSplit_{color}$. This action segments the input ROI into one or more ROIs based on color. The difficulty with handling this in our POMDP representation is that creating new ROIs changes the size of the state space of the POMDP. This means that we cannot plan to split a region because we cannot reason about the value of the state that results from doing the split.

To overcome this problem we observe that at most one of the regions that result from splitting are of interest in answering the query. In our ongoing example of finding the blue objects in an image, consider what happens when we split a region based on color, resulting in n new regions. There are two possibilities: Either one of the new regions is blue (there cannot be more than one because all the blue areas of the region are segmented together), or there are no blue regions. In the first case we can ignore the other regions for planning purposes and treat this as if a single blue region was created. In the second case there is no region relevant to the query and we can pick a region at random and plan with that. In either case, we can treat the effect of the splitting action to be to transform the region being operated on into a single "interesting" region, so the state space does not change. However, to make this approach work, we need to identify the interesting region. To do this we assume that every split action on a particular feature is followed by running the feature detector action on each of the regions that results. The $rSplit_{color}$ action, for instance, is followed by the application of $color$ on each resultant ROI. The $rSplit_{color}$ operator can hence be characterized as:

- The number of ROIs created as a result of the split operation is assumed to be distributed as a geometric distribution. The maximum number of possible ROIs equals the number of class labels (excluding *many*) provided by the underlying operator, four for $color$.
- The cost of the operator is the sum of the cost of performing the split operation based on color (i.e. color segmenting the ROI), and the cost of applying $color$ on the expected number of ROIs created by the split.

- The observation function ($\mathcal{O}$) is the same as that of the underlying operator ($color$), and it is used to perform the belief update on each ROI created after the split.
- The transition function is computed as follows: Assume that the feature being split on has n possible classes. Each of the ROIs created by the split has a probability of $\frac{1}{n}$ of having the class label relevant to the query, and the geometric distribution gives a probability of $1/2^{(i-1)}$ for producing i ROIs after the split. The special case of $i = n$ has a probability of occurrence of $1/2^{(n-2)}$ so that the geometric distribution-based probabilities sum to one. The expected probability that one of the ROIs has the appropriate class label is therefore given by:

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

  Then, iff the underlying state is "multiple", with probability p we move to a state where the ROI has the relevant label, and with probability $1 - p$ we move to a random state where the ROI has some other label. The transition matrix for $rSplit_{color}$ while looking for blue objects is given in Table 1.

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

Table 1: Transition function $\mathcal{T}$ for $rSplit_{color}$ with just the $color$ states under consideration.

The approach we have just described allows us to plan split operators in the LL-POMDP, but it relies on the assumption that the observations made of each new ROI are reliable. That is, it assumes that when we split on color and then apply the color operator to each new region, the operator reliably returns the true color of the region. Since this is not always true, we do not want to discard the regions generated in the split that did not have the correct class label. Rather we would like to update our belief about them based on the observation, but still allow them to be searched in the future. To do this, we need to add them to the HL-POMDP and compute policies for all the new regions. Thus, although the procedure above lets us plan split operators and estimate their cost, at execution time when a split action is performed, we need to replan[1].

When a split action is executed we now solve LL-POMDPs for each newly created ROI (these have to be resolved because the size of the regions has changed), use our current beliefs about each region, updated by the observations from the split actions to estimate the cost and observation probabilities for executing the policies, and use these to generate and solve a new HL-POMDP. Figures 2(a)-2(f) present the execution cycle, with the region-splitting actions at the LL, for the query: *Where are the Blue Circles?*

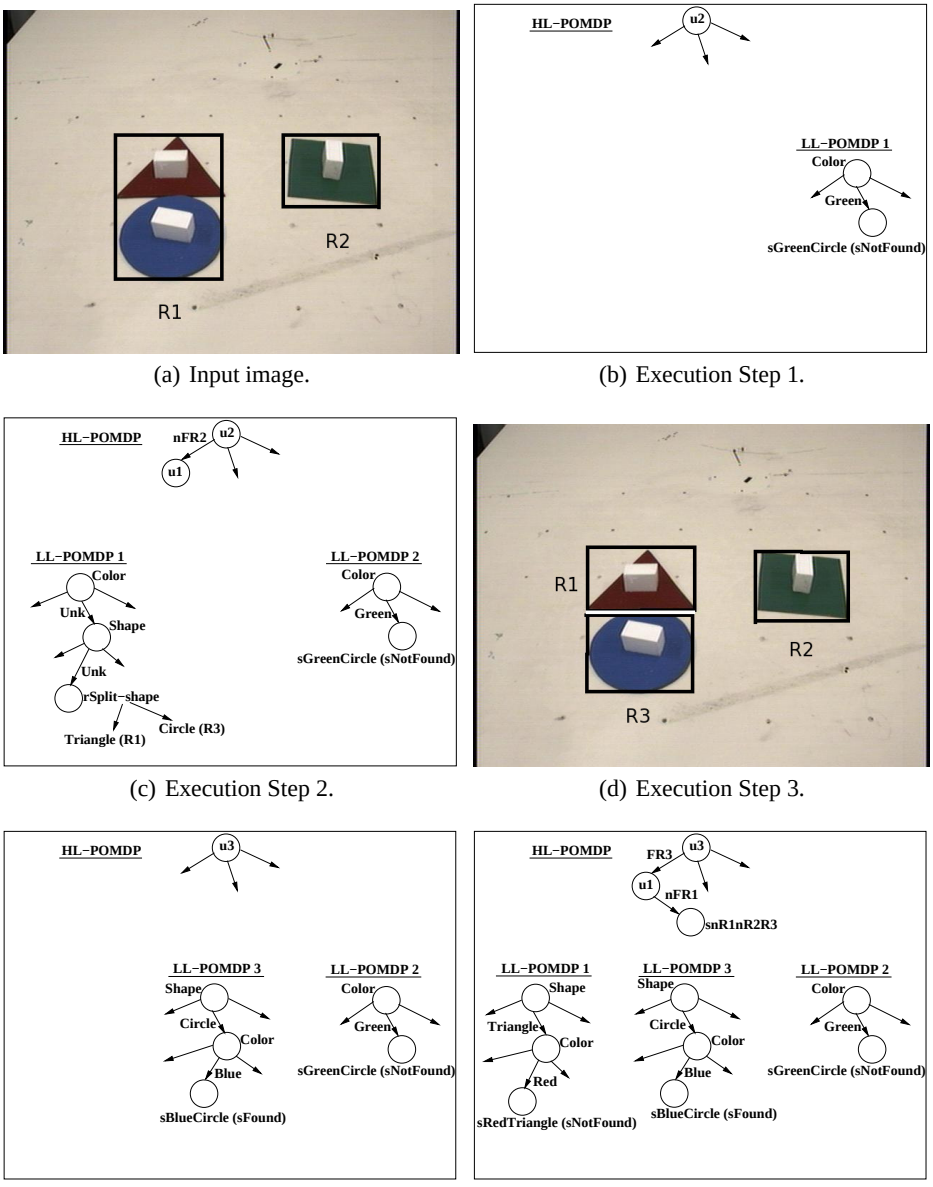
The image shown in Figure 2(a) has three objects, two of which overlap, leading to the creation of two ROIs. The task is to determine the presence and location of one or more *blue circles* in the scene—Fig 2(a). Since both ROIs are equally likely target locations, the HL-POMDP first chooses to execute the policy tree of the second ROI—action u_2 in Fig 2(b)—because of the lower processing cost of the smaller ROI. The corresponding LL-POMDP runs the color operator on the ROI, leading to the outcome of *green*. Finding *green* causes the likelihood of finding a blue circle to be reduced significantly, and the dynamic reward specification ensures that the *best* action chosen at the next level is a terminal action associated with the "Green" property—in this case it is *sGreenCircle*. 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—action u_1 in Fig 2(c). The policy tree of the LL-POMDP of R_1 is invoked, causing the color and shape operators to be applied in turn on the ROI. Both operators come up with outcomes of *Unknown* because of the two different colors and shapes in the ROI. At this point, the $rSplit_{shape}$ operator is chosen as the best action and R_1 is split into R_1 and R_3 on the basis of the shape contours identified in the ROI—Fig 2(d). Our system includes other algorithms that can be invoked, when necessary, to split a ROI on the basis of color (Felzenswalb and Huttenlocher 2004) or clustering of image features (Duda, Hart, and Stork 2000). In the current example, $rSplit_{shape}$ is followed by the application of the shape operator on each sub-region, leading to the observations *triangle* and *circle* in R_1 and R_3 respectively—Fig 2(c). Then the current HL-POMDP beliefs are used to create and solve a new HL-POMDP model for three ROIs. The subsequent action selection in the HL (u_3) results in the execution of the LL-policy of R_3. The ROI's initial state reflects the previous application of the shape operator and hence the color and shape operator are applied just once before the terminal action (*sBlueCircle*) is chosen, as shown in Fig 2(e)—since the shape operator is less reliable than the color operator, normally it takes two applications on the shape operator on two different images of the same scene to accumulate sufficient belief. The update of the beliefs at the HL leads to the processing of R_1 (because the query requires the computation of all locations of blue circles) leading to the terminal action of *sRedTriangle* in R_1 and $s(\neg R_1 \wedge \neg R_2 \wedge R_3)$, i.e. the desired object is found in R_3 but not in R_1 or R_2, at the HL—Fig 2(f).

Policy Caching

In our previous paper we presented a graph (shown here in Figure 3) comparing the planning time for HiPPo with using the CP planner of (Brenner and Nebel 2006). In the figure we claimed that policy caching made the two approaches comparable in terms of plan time, but we gave no details of how caching worked. Here we explain the details of caching and investigate its properties and performance.

For the caching results in Figure 3 we actually assumed that we could solve a single LL-POMDP and reuse the pol-

[1]We are currently investigating whether taking this additional planning time into account in the cost of the split action produces better performance.



(a) Input image. (b) Execution Step 1.

(c) Execution Step 2. (d) Execution Step 3.

(e) Execution Step 4. (f) Execution Step 5.

Figure 2: Example query: "Where is the Blue Circle?" Region-splitting operators allow for the creation of appropriate ROIs to answer the query.

icy for each ROI. While this is true if all ROIs are the same size, if they differ then the reward/cost of each visual action (i.e. not a special action) is a function of the relative time complexity of the action, and the size of the ROI being operated upon. This dependency on the ROI-size, specified as $f(\text{ROI-size})$ in Equation 1, is modeled as:

$$f(r) = a_0 + \sum_{k=1}^{N} a_k \cdot r^k \tag{2}$$

where r is the ROI-size (in pixels) and $N = 3$ i.e. we use a cubic polynomial to approximate the dependency on the size of the ROI being processed by the visual operators—the robot estimates the parameters of the polynomial. In order to make caching possible, we have to discretize the possible ROI sizes and use a single cost estimate for each ROI within a particular ranges of sizes. This approximation introduces an error, which can be estimated and used to perform a trade-off between the computational effort expended in creating and solving the LL-POMDPs and the error incurred by not computing the action costs accurately.

Consider the image shown in Figure 4 with three ROIs extracted from the background. The individual ROI sizes for R_1, R_2, R_3 are 23400, 11050 and 20800 pixels respectively. We have three discretization options here: (1) different action costs for each individual ROI, which would require solving three POMDPs; (2) the same action costs for

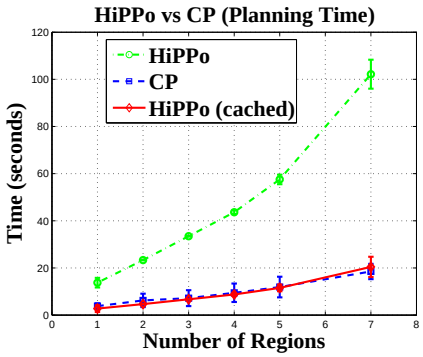


Figure 3: Planning times of HiPPo vs. CP. Policy-caching makes results comparable.

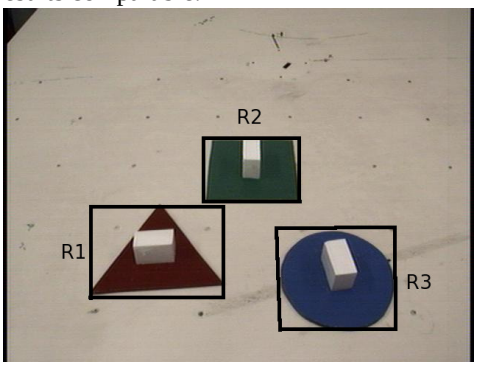


Figure 4: Image with three ROIs extracted from the background.

R_1 and R_3 and a different set of action costs for R_2, leading to the creation and solving of two LL-POMDPs; (3) the same set of action costs for all three ROIs, which would imply that the LL models need to be created and solved just once. In terms of computational costs for creating the LL models and policies:

$$\text{ModelCost}_{Option_2} = \frac{2}{3} \times \text{ModelCost}_{Option_1} \quad (3)$$
$$\text{ModelCost}_{Option_3} = \frac{1}{3} \times \text{ModelCost}_{Option_1}$$

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

$$max_{\substack{a \in \mathcal{A} \\ a \notin \mathcal{A}_S}} |f(r_i) - f(r_{avg})| = \delta \quad (4)$$

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 γ in the POMDP models, the net maximum error due to the ROI-size approximation is:

$$error = \delta + \gamma \cdot \delta + \cdots + \gamma^{N-1} \cdot \delta \quad (5)$$
$$= \delta\{\frac{1-\gamma^N}{1-\gamma}\}$$

where N represents the number of levels that the LL policy tree is allowed to grow to. For $\gamma = 0.9$ and $N = 8$, $error \approx 6\delta$. The upper bounds on the estimation errors in $Option_2$ and $Option_3$, taking into account the actual ROI-sizes in Figure 4, are 0.33 and 2.35 respectively. For $Option_2$, we then compute the actual error, i.e. we estimate two policies for R_1 using models that include actions costs based on r_{avg} and ROI-size(R_1) respectively, and then compute the difference in the *values* of the two policies for the initial state of R_1. The error value is found to be 0.021, and a similar computation for $Option_3$ with R_2 (the ROI where the maximum error is expected to occur) provides an error of 0.24. We observe that the actual error is significantly smaller than the upper bound on error.

The error estimation process described above can be performed automatically by the system (i.e. robot) to determine the ROI-size discretization to use for the given image by trading off the expected error against the reduction in computational effort to be spent determining the models and policies. In the current example, for instance, $Option_2$ may be the better than $Option_3$.

We note in passing that the observation function would also vary for different ROI sizes. We have not considered this as yet due to the difficulty of collecting sufficient data to learn the relationship between ROI sizes and vision algorithm performance.

Related Work

There has been extensive research in the computer vision community on planning a sequence of visual operations in order to perform a specific task. Typically, the user specifies the task/goal, which is used by a classical AI planner to construct a sequence of image processing operations. The planners use deterministic models of the action effects: 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; Chien, Fisher, and Estlin 2000; Thonnat and Moisan 2000; Moisan 2003). Execution monitoring is used to detect unsatisfactory performance and repair the plan by re-planning an action sequence or modifying the parameters of the operators. 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 by Li et al. 2003 has modeled image interpretation as a MDP (Markov Decision Process), using human-annotated images to determine the reward structure and explore the state space to determine dynamic programming-based value functions that are extrapolated to the entire state space through the ensemble technique called leveraging. Online image interpretation involves the choice of an

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 and actions have non-deteministic outcomes. A POMDP formulation provides an elegant means of modeling these features.

Within the planning community, there has recently been some work on relaxing the limiting constraints of classical planning schemes to make them suitable to practical application domains (Petrick and Bacchus 2004; Brenner and Nebel 2006). Petrick and Bacchus proposed the PKS planner 2004, which uses actions 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 domain, we could say that the objects in the query are 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 region, branching based on what is discovered.

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. CP is hence quite similar to PKS in its representation, but works by replanning rather than constructing conditional plans. 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.

The POMDP formulation (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. But, in any practical problem domain, the state space quickly grows too large to be solved by conventional POMDP solvers. Pineau and Thrun 2002 cope with large state spaces in POMDPs, for a nursing assistant robot, through a hierarchical approach 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 et al. 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.

Instead of manually specifying the abstractions at several levels, we propose a simple two-level POMDP hierarchy, where the reward and observation models can be learned autonomously (Sridharan, Wyatt, and Dearden 2008). At the lower level (LL), each ROI is assigned a POMDP, whose state and action 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 with completely different state, action and observation spaces. The HL POMDP maintains the belief over the entire image and chooses the best ROI for further processing, so as to answer the queries posed. This hierarchy structure can be used unmodified for a range of queries in our application domain.

Conclusions and Future Work

Robots operating in real-world application domains need to function autonomously, modifying their sensory processing based on the task to be performed. In a recent paper (Sridharan, Wyatt, and Dearden 2008) we proposed a hierarchical POMDP planning method (HiPPo) that enables a robot to plan a sequence of visual operators so as to accomplish the desired goal with high reliability while still optimizing the computational resources. In a domain where a robot and a human have to jointly converse about and manipulate objects on a tabletop, our probabilistic approach enables the robot to exploit learned models of the uncertainty in the action outcomes to accumulate belief and hence answer user queries with significantly higher reliability than a representative modern planning framework that is non-deterministic.

In this paper we have extended the HiPPo framework to handle overlapping objects in the scene by splitting the region to allow recognition of the individual objects. In the future, we aim to include other visual operators in the analysis, such as a viewpoint change to get a better view of the scene objects. In addition, we aim to tackle the interesting problem of learning object affordances. These additions may require a range of hierarchies in the state and action spaces (Pineau and Thrun 2002), though we would like to learn this hierarchy (Toussaint, Charlin, and Poupart 2008).

Here we have analyzed 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 the computation of action costs for each image region 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 not a problem in our current scenario. However, in more general scenes the approximation error may grow unacceptably and less caching may provide significantly better performance. Ultimately this should be evaluated by including planning time in the performance metric. 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.

There are still several interesting challenges to address, such as the extension of the 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?) that would require reasoning about action affordances. 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

This work was supported by 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 Symposium of Practical Cognitive Agents and Robots*.

Chien, S.; Fisher, F.; and Estlin, T. 2000. Automated software module reconfiguration through the use of artificial intelligence planning techniques. *IEE Proc. Software* 147(5):186–192.

Clouard, R.; Elmoataz, A.; Porquet, C.; and Revenu, M. 1999. Borg: A knowledge-based system for automatic generation of image processing programs. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 21(2):128–144.

Dietterich, T. 1998. The MAXQ Method for Hierarchical Reinforcement Learning. In *International Conference on Machine Learning (ICML)*.

Duda, R. O.; Hart, P. E.; and Stork, D. G. 2000. *Pattern Classification*. Wiley Publishers, 2nd edition.

Felzenswalb, P. F., and Huttenlocher, D. P. 2004. Efficient Graph-Based Image Segmentation. *International Journal of Computer Vision* 59(2).

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.; 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)*.

Kaelbling, L.; Littman, M.; and Cassandra, A. 1998. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence* 101:99–134.

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. *International Journal of Computer Vision (IJCV)* 60(2):91–110.

Minten, B. W.; Murphy, R. R.; Hyams, J.; and Micire, M. 2001. Low-Order-Complexity Vision-Based Docking. *IEEE Transactions on Robotics and Automation* 17(6):922–930.

Moisan, S. 2003. Program supervision: Yakl and pegase+ reference and user manual. Rapport de Recherche 5066, INRIA, Sophia Antipolis, France.

Petrick, R., and Bacchus, F. 2004. Extending the Knowledge-Based approach to Planning with Incomplete Information and Sensing. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2–11.

Pineau, J., and Thrun, S. 2002. High-level Robot Behavior Control using POMDPs. In *The National Conference on Artificial Intelligence (AAAI)*.

Pineau, J.; Montemerlo, M.; Pollack, M.; Roy, N.; and Thrun, S. 2003. Towards Robotic Assistants in Nursing Homes: Challenges and Results. *Robotics and Autonomous Systems, Special Issue on Socially Interactive Robots* 42(3-4):271–281.

Shekhar, C.; Moisan, S.; and Thonnat, M. 1994. Use of a real-time perception program supervisor in a driving scenario. In *Intelligent Vehicle Symposium '94*.

Sridharan, M.; Wyatt, J.; and Dearden, R. 2008. HiPPo: Hierarchical POMDPs for Planning Information Processing and Sensing Actions on a Robot. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Thonnat, M., and Moisan, S. 2000. What can program supervision do for program reuse? *IEE Proc. Software* 147(5):179–185.

Thrun, S. 2006. Stanley: The Robot that Won the DARPA Grand Challenge. *Journal of Field Robotics* 23(9):661–692.

Toussaint, M.; Charlin, L.; and Poupart, P. 2008. Hierarchical POMDP Controller Optimization by Likelihood Maximization. In *Uncertainty in AI (UAI)*.