In *The Fifth International Cognitive Vision Workshop* (ICVW 09),
St. Louis, USA, October 11 2009.

POMDP-based Planning for Visual Processing Management on a Robot

M. Sridharan[1], R. Dearden[2], and J. Wyatt[3]

[1] Department of Computer Science, Texas Tech University, USA
mohan.sridharan@ttu.edu
http://www.cs.ttu.edu/~smohan
[2] School of Computer Science, University of Birmingham, UK
rwd@cs.bham.ac.uk
http://www.cs.bham.ac.uk/~rwd
[3] School of Computer Science, University of Birmingham, UK
jlw@cs.bham.ac.uk
http://www.cs.bham.ac.uk/~jlw

Abstract. Recent progress in sensor technology [10, 24], and the use of state of the art algorithms to process the input from a variety of sensors, has resulted in the deployment of mobile robots in several specific applications [2, 17, 22]. A key requirement for the widespread deployment of mobile robots is the ability to autonomously tailor the sensory processing to the task at hand. Our work represents a significant effort towards such general-purpose processing of visual input. We pose *visual processing management* as an instance of probabilistic sequential decision making, and specifically as a Partially Observable Markov Decision Process (POMDP). Our prior work introduced a hierarchical POMDP decomposition that enables a robot to plan a sequence of visual operators that reliably and efficiently analyze the state of the world represented by salient regions-of-interest (ROIs) in input images [20]. Here, we significantly enhance the capabilities of the existing system by: (a) extending our POMDP framework to autonomously adapt to a change in state space dimensions, thereby enabling the robot to effectively process partially overlapping objects in the image; and (b) enabling the robot to autonomously trade-off planning speed and plan quality, by theoretically and empirically evaluating the estimation errors involved in policy caching. All algorithms are implemented and tested on a physical robot platform. We show that the hierarchical planner performs significantly better than a modern planner that has been applied successfully to human-robot interaction domains [1].
Keywords: Cognitive Vision, Probabilistic Sequential Decision-making, Human Robot Interaction.

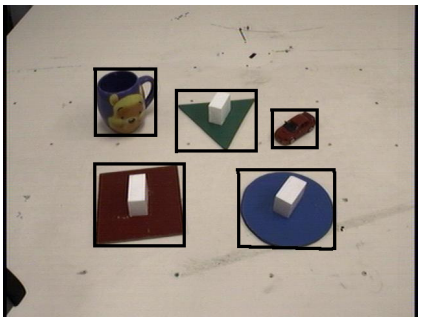


Fig. 1: An image of the table-top scenario: regions-of-interest (ROIs) bounded by rectangles.

1 Motivation

The ability to deploy robots in the real-world where they can learn from and collaborate with humans remains an open challenge. Major aspects of this challenge include:

- *Autonomous Adaptation*: How to enable a robot to autonomously model the environment based on sensory input, and revise the learned models in response to changes?
- *Processing Management*: Given multiple sources of information, which bits of information should be processed, and what processing should be performed in order to achieve a desired goal reliably and efficiently?
- *Learning from Human Feedback*: How to enable a robot to learn from limited human feedback, and collaborate with humans over a wide range of tasks?

Progress in sensor technology has made it possible to equip mobile robots with high-quality sensors at moderate costs [10, 24]. In addition, the availability of state of the art algorithms for segmentation, recognition and scene analysis, has enabled robots to exploit the rich information encoded in color images, in several applications [2, 17, 22]. However, most current robot systems equipped with visual sensors are designed for specific domains using a manually chosen set of visual operators or processing routines. In addition, visual processing on robots is characterized by non-deterministic actions (the operators are not completely reliable), high computational complexity and partial observability. The robot cannot observe the true state of the world—it can only update its *belief* by executing the processing routines and observing the outcomes. The robot can analyze the input images using a subset of a large set of processing routines, each with a different computational complexity and reliability. At the same time it has to respond to dynamic changes and operate with a high degree of reliability.

A robot equipped with visual sensors, and operating in a dynamic environment, therefore needs a strategy to autonomously tailor its processing to the task at hand. We pose *visual processing management* as a planning problem, where the robot plans a sequence of information processing (*what to look for?*) and sensing (*where to look?*) actions that would maximize the reliability while using the available resources optimally. In our experimental domain, the robot is equipped with cameras and a manipulator arm—Fig. 3(a). The robot and a human jointly converse about and manipulate objects on a tabletop. Such a scenario, though seemingly simple, represents the state of the art

in cognitive robotics—modules operating in parallel process visual and speech inputs, and bind the information to create goals that are achieved by other modules such as manipulation [6]. Consider the scene in Fig. 1, where the robot obtains images with the salient regions-of-interest (ROIs) extracted from the background. The robot has to answer queries such as: "is there a blue triangle in the scene?" or execute commands such as: "put the mug next to the circle?". In order to do so, the robot may need 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; and to plan and execute an obstacle-free path to move the object to a new location. However, it is infeasible to apply all the available operators on a given image, especially in domains that require a dynamic response. The robot needs to choose a subset that is relevant to the task.

Though planning of visual operators has been extensively researched, prior approaches typically used single images, required extensive domain knowledge to perform plan repair, have only been extended to robot systems in limited ways, or have posed the problem in deterministic or observable framework [3, 13, 15]. We formulate visual processing management as a Partially Observable Markov Decision Process (POMDP) [12], which elegantly captures the characteristic features of robot domains (e.g. non-deterministic actions, partially observable states). However, POMDP formulations of practical domains with reasonable-sized state spaces are intractable. Our prior work addressed this intractability by introducing a *hierarchical decomposition* that the robot can model automatically [20]—prior attempts at imposing structure on a POMDP have required significant manual supervision [8, 17]. Our hierarchical POMDP planner (*HiPPo*) provided higher reliability than a modern non-probabilistic planner that has been used successfully in human-robot collaboration scenarios [1]. There were however two limitations: (a) the system could not reliably process partially overlapping objects in the image, a common occurrence in practical domains, and (b) planning times comparable to that of the non-probabilistic planner could only be obtained by reusing the POMDP policy for similar ROIs, which introduced an approximation error in plan quality. Here, we address these limitations through the following contributions:

- We extend our POMDP approach to handle changes in the state space dimensions. As a result the robot is able to plan with an operator that splits the ROIs containing overlapping objects into sub-regions with distinct objects.
- We investigate the effects of policy-caching theoretically and empirically, and use it to autonomously trade-off plan quality against planning speed.

All algorithms are tested on a physical robot in the scenario described above.

2 Hierarchical POMDP Planner

The hierarchical POMDP framework that forms the basis of the work described in this paper, receives input images that are processed to yield salient regions-of-interest (ROIs) different from a previously trained background model (rectangular regions in Fig. 1). The robot is tasked with executing commands ("put the mug next to the red square") or answering queries ("which objects in the scene are blue?"). The goal is to plan a sequence of visual operators that would perform the task reliably and efficiently. Without loss of generality, assume that the robot has the following operators at its disposal: a $color$ operator (based on color histograms) that classifies the dominant color

of the ROI it is applied on, a $shape$ operator (based on shape moments) that classifies the dominant shape within the ROI, and a $sift$ operator (based on the SIFT features [14]) that detects 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.

Since the true state of the system cannot be observed, the robot maintains a probability distribution over the underlying state (*belief state*), and uses a probabilistic model of the action outcomes. In our work, each action considers the true underlying state to be composed of the class labels (e.g. *red(R), green(G), blue(B)* for color; *circle(C), triangle(T), square(S)* for shape; *picture, mug, box* for sift), a label to denote the absence of any valid object class—*empty* (ϕ), and a label to denote the presence of *multiple* classes (M). The model for each action's outcomes provides a probability distribution over the set composed of the corresponding class labels, the label *empty* (ϕ) which implies that the match probability corresponding to the class labels is very low, and *unknown* (U) which means that multiple class labels are equally likely. U is an observation whereas M is part of the state: there is a correlation but they are not the same.

Since visual operators only update belief states, we include query-specific "special actions" that indicate the presence or absence of the target object, or "say" (not to be confused with language 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. Below, for ease of explanation and without loss of generality, we only consider two operators: $color$ and $shape$ denoted by subscripts c, s respectively. States and observations are distinguished by 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 variables describing different aspects of the underlying state (e.g. color, shape). It also includes a *terminal* state ($term$). $\mathcal{S}_c : \{\phi_c^a, R_c^a, G_c^a, B_c^a, M_c\}$, $\mathcal{S}_s : \{\phi_s^a, C_s^a, T_s^a, S_s^a, M_s\}$.
- $\mathcal{A} : \{color, shape, A_{\mathcal{S}}\}$ is the set of actions. The first two entries are the visual operators. The rest are special actions representing query responses. For a query such as "is there a circle in the scene?", $A_{\mathcal{S}} = \{sFound, sNotFound\}$ describes the presence or absence of the target object, while a query such as "what is the color of the ROI?" has $A_{\mathcal{S}} = \{sRed, sGreen, sBlue\}$ i.e. actions such as "say blue". All special actions lead to $term$.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ represents the state transition function. For visual operators 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*.
- $\mathcal{Z} : \{\phi_c^o, R_c^o, G_c^o, B_c^o, U_c, \phi_s^o, C_s^o, T_s^o, S_s^o, U_s\}$ is the set of observations, a concatenation of the observations for each visual operator.
- $\mathcal{O} : \mathcal{S} \times \mathcal{A} \times \mathcal{Z} \rightarrow [0, 1]$ 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, a mapping from state-action space to real numbers.

$$\forall s \in \mathcal{S},\ \mathcal{R}(s, \text{operators}) = -\beta \cdot f(\text{ROI\_size}) \qquad (1)$$
$$\mathcal{R}(s, \text{special actions}) = \pm 100 \cdot \alpha$$

For visual operators, the cost depends on the size of the ROI (see Eqn 3) and the relative computational complexity (β for $color$ is twice that for $shape$). Special actions

are assigned a large positive (negative) reward for giving the right (wrong) answer for a query. Variable α is used to trade-off computational costs and reliability. Values for α, β are computed experimentally based on query complexity.

The visual planning for a single ROI now involves solving this POMDP to find a policy that maximizes reward over a range of belief states. Plan execution corresponds to 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. However, for a single ROI with m features (color, shape etc.) each with n values (e.g. R, G, B for color), the POMDP has an underlying space of $n^m + 1$; for k ROIs the overall space is: $n^{mk} + 1$. The problem soon becomes too large to solve even with state of the art (approximate) POMDP solvers.

We partially ameliorate the exponential state space explosion problem by imposing an intuitive *hierarchical decomposition* that encapsulates different cognitive levels of abstraction. Each ROI is modeled with a lower-level (LL) POMDP as described above, and a higher-level (HL) POMDP chooses, 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 $n^m + 1$. Without loss of generality, consider an input image with two ROIs, whose HL-POMDP is given by the tuple $\langle \mathcal{S}^H, \mathcal{A}^H, \mathcal{T}^H, \mathcal{Z}^H, \mathcal{O}^H, \mathcal{R}^H \rangle$:

- $\mathcal{S}^H = \{R_1 \wedge \neg R_2, \neg R_1 \wedge R_2, \neg R_1 \wedge \neg R_2, R_1 \wedge R_2\} \cup term^H$ is the set of states. It represents the presence or absence of the object in one or more of the ROIs, and includes a terminal state ($term^H$).
- $\mathcal{A}^H = \{u_1, u_2, A_S^H\}$ are the actions, where (u_i) denotes the choice of executing LL ROI R_i's policy tree. For queries such as "is there a circle in the scene?", the special action set $A_S^H = \{sFound^H, sNotFound^H\}$. However, for queries such as "where is the blue circle?", A_S^H represents the fact that one of the entries of $\mathcal{S}^H$ is the answer. All special actions 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 or not-finding the desired object when each ROI's policy is executed.
- $\mathcal{O}^H : \mathcal{S}^H \times \mathcal{A}^H \times Z^H \rightarrow [0, 1]$, the observation function, is a uniform matrix for special actions. For other actions, it is learned from the LL-POMDP policy trees.
- $\mathcal{R}^H$ is the reward specification. For a special action, it is a large positive (negative) value if it predicts the state correctly (wrongly). For other actions, it is a "cost" computed from the LL policy trees.

The LL observation functions and rewards are learned in a training phase where the robot automatically collects visual operator performance statistics with known objects in the scene. During execution, the robot creates a LL-POMDP for each ROI based on the available visual operators and the query being posed. The model file is in the format required by the ZMDP package [25] and it is solved using a point-based solver in the package [19]. The computational complexity is controlled by forcing each LL-POMDP's policy tree to terminate after N levels, at which point all branches have to take a terminal action—the value of N is determined experimentally based on query complexity. The LL policy trees and the query are used to compute the HL observation

functions and rewards at run-time, and the appropriate HL-POMDP model is created and solved autonomously to generate the overall policy that is executed by the robot. The proposed approach is therefore different from existing work on imposing structure on a POMDP, where the hierarchy and models are manually encoded [8, 17].

When compared against Continual Planning (CP) [1], a modern planner that has been used successfully in robot domains, our planner (HiPPo) is significantly more reliable (Table 2). CP does not model the action outcomes and hence cannot achieve higher reliability than "no-planning" i.e. the naive approach of applying all operators on the image. Both planners were significantly faster than no-planning, even for images with as few as two ROIs processed with two or more operators. However, HiPPo had two limitations: (a) overlapping objects in the image resulted in operator outcomes of *unknown* and hence low reliability for query responses; (b) planning times of HiPPo were comparable to those of CP *iff* the policies of LL ROIs were cached and reused for similar ROIs, but policy-caching leads to approximation errors that affect reliability. We address these limitations in this paper.

3 Handling Overlapping Objects

Scene objects frequently overlap in practical applications, and are likely to be detected as a single salient region (ROI). Visual operators applied on such ROIs are likely to return *unknown*, decreasing the robot's ability to respond to queries reliably. The occurrence of multiple objects in a single ROI can be tackled by using "region-splitting" actions that segment the ROI based on the underlying features such as $color$ or $shape$. For instance, $rSplit_{color}$ segments the input ROI into one or more ROIs based on color. Incorporating splitting in a POMDP planning framework is challenging because creating new ROIs changes the size of the state space. We cannot *plan* to split a ROI because we cannot reason in advance about the value of the resulting state.

Consider the action $rSplit_{color}$ that results in n sub-regions, and consider the goal of detecting the *blue* objects in the image. There are two possibilities: one of the new regions is blue (we assume that all blue areas are segmented together), or there are no blue regions. In the first case, for planning purposes, we can proceed 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 continue planning. In either case, the effect of splitting is to transform the ROI being operated on into a single "interesting" sub-region, so that the state space is unchanged (for planning purposes). However, the interesting sub-region still needs to be found. The execution of a split action based on a feature, is therefore followed by the execution of the corresponding feature detector on each resulting sub-region. For instance, $rSplit_{color}$ is followed by the application of $color$ on each resultant ROI. We can hence model $rSplit_{color}$ as:

- The number of ROIs resulting from a split action is assumed to follow a geometric distribution. The maximum number of possible ROIs is equal to the number of labels of the underlying operator (e.g. five for $color$).
- The cost of the operator is the sum of three values. The first is the cost of performing the split (i.e. segmenting the ROI based on color). The second is the cost of re-applying the color operator on the expected number (d) of ROIs created by the split. Since each of the d ROIs is only $1/d$ the size of the ROI being split, for linear

action costs, this is equivalent to applying the action once on the original ROI. The third is the cost of solving the POMDPs corresponding to the new ROIs.

- The observation function ($\mathcal{O}$) is the same as that of the underlying operator ($color$). It is used to perform the belief update on each ROI created by the split.
- For a split action based on a feature with n possible classes, each of the ROIs created by the split has probability $\frac{1}{n}$ of having the class label relevant to the query, and the geometric distribution assigns a probability of $1/2^{(i-1)}$ for producing i ROIs after the split. The special case of $i = n$ occurs with probability: $1/2^{(n-2)}$ so that the geometric distribution-based probabilities sum to one. The probability that one of the ROIs has the desired class label is therefore given by:

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

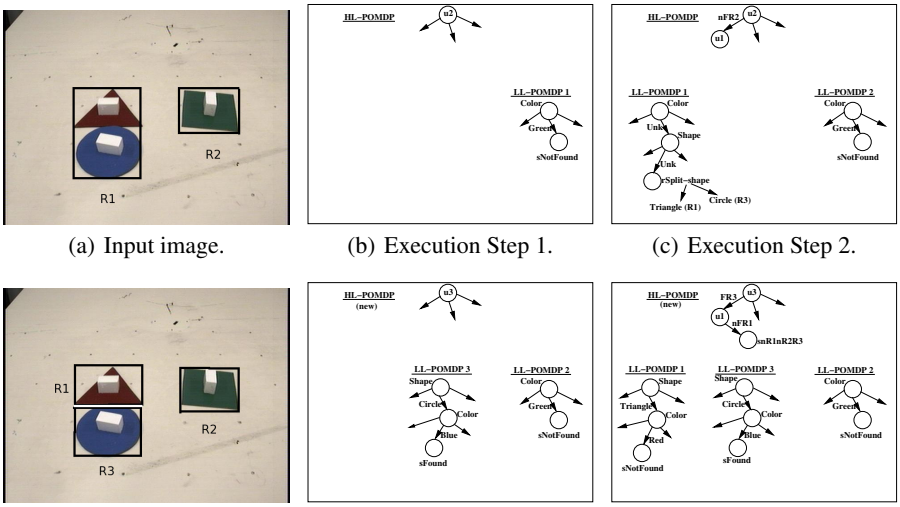
This formulation sets up a transition where, *iff* the underlying state is "multiple", with probability p we reach a state where the ROI has the relevant label, and with probability $1 - p$ we reach a random state where the ROI has some other label. The transition matrix for $rSplit_{color}$ while looking for blue objects, is given by 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.

This approach lets us plan with split operators in the LL-POMDP, but relies on the assumption that applying $rSplit_{color}$ followed by $color$ on each new ROI returns the true color of the ROI. Since this is not always true, it would be better to update the belief of the new ROIs and include them in subsequent analysis. However, including new ROIs would require the HL-POMDP to be modified and new policies to be computed for these ROIs and the HL-POMDP. Therefore, though our approach enables planning with split operators, executing a split operator would require replanning. Such cases are handled at run-time by solving a LL-POMDP for each new ROI (since the region-sizes and hence action costs are different). The observations from the split action are used to update the beliefs about each region and estimate the costs and observation probabilities for executing the policies. A new HL-POMDP is then created and solved, the whole process still being autonomous.

Figs. 2(a)-2(f) show the execution steps for the query: "where are the blue circles?". Fig. 2(a) shows three objects, two of which overlap to provide two ROIs. Since both ROIs are equally likely target locations (no prior information), the HL-POMDP chooses to execute the policy tree of the second ROI—action u_2 in Fig. 2(b)—because it is smaller and hence cheaper to process. The corresponding LL-POMDP applies the color operator on the ROI—$color$ is twice as costly as $shape$ but it is more reliable and hence gets chosen first. The outcome (*green*) causes the likelihood of finding a blue circle to



(a) Input image. (b) Execution Step 1. (c) Execution Step 2.

(d) Execution Step 3. (e) Execution Step 4. (f) Execution Step 5.

Fig. 2: (a)-(f) Execution steps for query: "Where are the Blue Circles?"—region-splitting operators lead to reliable query response.

be reduced significantly, and the *best* action chosen at the next step is a terminal action: *sNotFound*. The HL-POMDP receives this input, updates its belief and chooses the action u_1—Fig. 2(c). The policy tree of the LL-POMDP of R_1 causes $color$ and $shape$ to be applied in turn on the ROI. Both operators come up with outcomes of *unknown* because two different colors and shapes exist in the ROI. At this point, $rSplit_{shape}$ gets chosen as the best action (lower cost than $rSplit_{color}$), and R_1 is split into R_1 and R_3 based on the shape contours in the ROI—Fig. 2(d). Similar algorithms can be invoked, when necessary, to split a ROI based on color [7] or clustering of gradient features [5]. Here, $rSplit_{shape}$ is followed by the application of $shape$ on each sub-region, providing observations *triangle* and *circle* for R_1 and R_3 respectively—Fig. 2(c). The current LL and HL belief states are used to create and solve a new HL-POMDP for three ROIs. The subsequent HL action selection (u_3) executes the LL-policy of R_3. Since the shape operator is less reliable, it is sometimes applied more than once on independently captured images of the same scene in order to accumulate belief and negate the effects of illumination changes etc. However, the ROI's belief state reflects the prior application of $shape$ during splitting, and hence $color$ and $shape$ are applied once before the terminal action (*sFound*) is chosen—Fig. 2(e). The HL belief update then chooses R_1 (action u_1) leading to the terminal action: *sNotFound* in R_1 and $s(\neg R_1 \wedge \neg R_2 \wedge R_3)$ at the HL i.e. the desired object is in R_3 but not in R_1 or R_2—Fig. 2(f).

Though the overlap is limited in this example (the objects are touching each other), the approach also works when the level of overlap is greater, as long as the features (e.g. shape) of the overlapping objects are not completely indistinguishable. In the absence of the split operators, the reliability drops to $\approx 60\%$ on images with overlapping objects. However, our approach provides high reliability ($\geq 90\%$) even in these problem

cases. The system fails only when it consistently receives noisy images that the operators are unable to process reliably. The *key contribution* is the ability to incorporate actions that change the state space dimensions, which is a general challenge in POMDP formulations. This capability could, for instance, prove useful when analyzing actions that learn object properties by interacting with them.

4 Analyzing Policy Caching

In our prior work, we compared the planning and execution times of HiPPo with CP [20]. The average execution time of HiPPo was slightly larger due to instances where the same operator had to be applied multiple times to accumulate belief. The planning time of HiPPo was however substantially larger than that of CP, primarily because policies had to be generated for each LL-POMDP. Since robot applications require reliability and efficiency, we had claimed that the planning times would be comparable if the policy computed for one ROI is cached and reused for similar ROIs—see Fig. 3(b).

While the results in Fig. 3(b) are valid if all ROIs have the same size, the ROIs in a typical scene will have different sizes and hence different costs associated with each operator. The operator costs (Equation 1) are a function of the relative time complexity of the action, and the size of the ROI. The dependency on the ROI_size is modeled as:

$$f(r) = a_0 + \sum_{k=1}^{N} a_k \cdot r^k \tag{3}$$

where r is the ROI_size in pixels, and $N = 3$ i.e. we use a cubic polynomial to approximate the dependency on the ROI_size. The robot estimates the parameters during the training phase where it learns the LL observation functions. For policy-caching, the ROI sizes can be discretized and all ROIs within a particular size range can use the same action costs. However, any such approximation introduces an error in the estimation of the value of a policy. There is hence a trade-off between the computation involved in solving the LL-POMDPs, and the value estimation error incurred by caching.

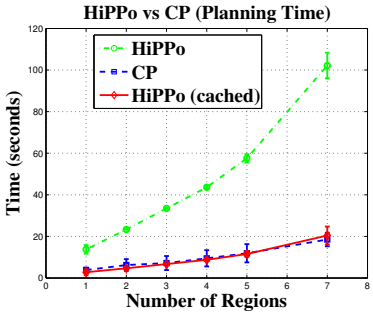
Consider the image shown in Fig. 3(c) with three ROIs extracted from the background. The individual ROI sizes for R_1, R_2, R_3 are 23400, 11050 and 20800 pixels respectively. There are three discretization options: (1) different action costs for each individual ROI, which would require three LL-POMDP solutions; (2) the same action costs for R_1 and R_3 and a different set of action costs for R_2, requiring two LL-POMDP solutions; (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 the computational costs for creating the LL policies:

$$\begin{aligned} \text{ModelCost}_{Option_2} &= 2/3 \times \text{ModelCost}_{Option_1} \\ \text{ModelCost}_{Option_3} &= 1/3 \times \text{ModelCost}_{Option_1} \end{aligned} \tag{4}$$

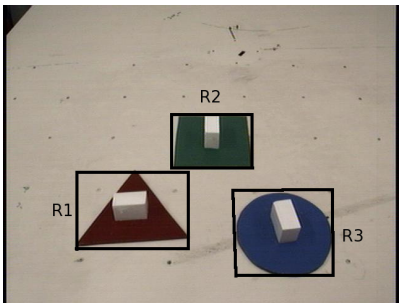
However, $Option_2$ and $Option_3$ incur approximation errors. We compute a theoretical upper bound on this error first. The maximum approximation error in the action costs, over all visual operators under consideration, and for ROIs whose sizes fall within the discretization range is:

$$max_{\substack{a \in \mathcal{A} \\ a \notin \mathcal{A}_S}} |f(r_i) - f(r_{avg})| = \delta \tag{5}$$

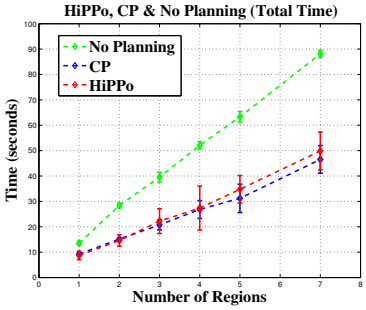
(a) Scenario Overview.



(b) Planning Time.



(c) Policy-caching Analysis.



(d) Planning + Execution Time.

Fig. 3: (a) Overview of the scenario; (b) Planning times of HiPPo vs. CP, policy-caching makes results comparable; (c) Sample image with three ROIs; (d) Planning and execution times of HiPPo vs. CP—HiPPo takes slightly more time but provides significantly higher reliability.

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 $Option_3$:

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

For a discount factor of γ in the POMDP models, the net maximum error due to the ROI_size approximation is:

$$error = \delta + \gamma \cdot \delta + \cdots + \gamma^{N-1} \cdot \delta \tag{6}$$
$$= \delta\{\frac{1-\gamma^N}{1-\gamma}\}$$

where N represents the number of levels in the LL policy tree. 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 Fig. 3(c), are 0.33 and 2.35 respectively. In order to empirically estimate the value estimation error in $Option_2$ we estimate two

policies for R_1 using models that include actions costs based on r_{avg} and ROI_size(R_1) respectively. The difference in the *values* of the two policies for the initial state of R_1 estimates the actual error. The error value is found to be 0.021 for $Option_2$, and a similar computation for $Option_3$ with R_2 (where the maximum error is expected to occur) provides an error of 0.24. The actual error is hence significantly smaller than the theoretical upper bound. Based on the acceptable approximation error, one of the discretization options may be chosen.

The *key consequence* is that the robot can autonomously evaluate different ROI_size discretizations and trade-off the expected approximation error against the reduction in the effort spent computing the policies. Fig. 3(d) compares the total (planning+execution) times of CP and HiPPo against no-planning, as a function of the number of ROIs in the images. The results were obtained by performing multiple trials of different queries (≈ 30), using three or more visual operators. The policy caching trade-off ensures that the total time of HiPPo is comparable to that of CP, even though HiPPo takes slightly more time than CP because some operators are executed more than once. However, HiPPo provides significantly higher reliability than CP, even over images with overlapping objects—Table 2. Though the individual operators are optimized, they are tested

Approach	**% Reliability**
Naive	76.67
CP	76.67
HiPPo	91.67

Table 2: Reliability of visual processing

on-board a cognitive robot that analyzes inputs from multiple modalities in parallel. There is hence a delay before the operators are triggered, leading to the planning and execution times reported in Figs. 3(b), 3(d).

5 Related Work

There is a significant body of work in computer vision on using a user-specified high-level goal to plan a pipeline of visual operators. However, many of these planning algorithms use deterministic models of the action outcomes: the pre-conditions and the effects of the operators are propositions that are required to be true a priori, or are made true by the application of the operator. Unsatisfactory results are detected using hand-crafted rules, and handles by re-planning the operator sequence or modifying the operator parameters [3, 15, 21]. There has also been some work on autonomous object avoidance in vehicles [18], and on interpretation of 3D objects' structure [11] but extensions to general vision tasks have proven difficult. In practical applications, the true state of the system is not directly observable and actions have unreliable outcomes.

Recent research in AI planning has focused on relaxing the limitations of classical planners to make them suitable to practical domains [1, 16]. The PKS planner [16] uses a first-order language to describe actions in terms of their effect on the agent's knowledge, rather than their effect on the world. Hence the model is non-deterministic in the sense that the true state of the world is determined uniquely by the actions performed,

but the agent's knowledge of that state is not. PKS captures the initial state uncertainty and constructs conditional plans based on the agent's knowledge. The Continual Planner (CP) [1], on the other hand, interleaves planning, plan execution and monitoring. An agent in CP postpones reasoning about uncertain states until more information is available. It allows actions to assert that the preconditions for the action will be met when that point is reached during plan execution. If those preconditions are not met during execution (or are met earlier), replanning is triggered. In applications with noisy observations, it may be necessary to take several images of a scene and run the operators more than once to reduce uncertainty. This cannot be represented in PKS or CP.

Unlike the approaches based on classical planners, Li et al. [13] modeled image interpretation as a Markov Decision Process (MDP). Human-annotated images are used to determine the reward structure, explore the state space and compute value functions that are extrapolated to the entire state space. Online operation involves action choices that maximize the value of the learned functions. Trevor Darrell [4] used manual feedback, memory-based reinforcement learning and POMDPs to learn what foveation actions to execute, and when to execute the terminal recognition action, in order to foveate salient body parts in an active gesture recognition system. Manual feedback and extensive training trials are however infeasible in many mobile robot domains.

The POMDP formulation [12] is appropriate for domains where the state is not directly observable, and the actions are unreliable. However, a POMDP formulation is intractable for most practical domains. Pineau and Thrun [17] proposed a hierarchical POMDP approach for behavior control of a robot assistant. In their action hierarchy, the top level action is a collection of simpler actions that are represented by smaller POMDPs and solved completely; planning happens in a bottom-up manner, combining individual policies to provide the total policy. All model parameters are defined over the same state space, but the relevant space is abstracted for each POMDP using a dynamic belief network. Similar approaches have been proposed for robot navigation [8] but a significant amount of data for the hierarchy and model creation is hand-coded. Theoretical analysis has resulted in techniques that impose and discover the hierarchy in POMDPs [9, 23], but the methods are computationally expensive. We have proposed an intuitive hierarchy, where the reward and observation models can be generated autonomously to answer a range of queries reliably and efficiently.

6 Conclusions

Enabling mobile robots to learn from and collaborate with humans remains an open challenge despite significant progress in some areas such as sensor technology, visual information processing and autonomous sensor adaptation. Robots still lack the ability to autonomously tailor their sensory processing to the task at hand. Our prior work posed visual processing management as a POMDP and introduced a hierarchical structure to make planning feasible in practical domains [20]. In this paper we have extended the existing framework to handle dynamic changes in state dimensions. As a result we are able to introduce appropriate visual operators and handle the problem of overlapping objects in the scene. In a domain where a robot and a human jointly converse about and manipulate objects on a tabletop, our approach enables the robot to exploit learned probabilistic models of action outcomes to answer queries efficiently and with a higher reliability than a representative modern planner.

The current experimental scenario, though seemingly simple, does represent the state of the art in cognitive robotics. The proposed approach is not an attempt to improve visual processing routines. Our aim is to introduce a novel probabilistic planning scheme based on cognitive levels of abstraction, which for a given task, reliably and efficiently sequences a subset of the existing visual processing routines. One future research direction is to include other operators that analyze a static scene (e.g. texture), and those that change the state dimensionality (e.g. a viewpoint change). A change in viewpoint can be modeled as a higher level of abstraction where the goal is to maximize information i.e. minimize entropy. The existing framework also provides the tools to address challenges such as learning 3D object affordances by interacting with the objects. However, an increase in the size of action space and state space may require a range of hierarchies [8, 17], though the hierarchies would be learned through approximations of theoretical advances [23].

Here, we have presented a theoretical and empirical analysis of the approximation error introduced by policy-caching. The robot is able to trade-off the effort involved in solving the LL-POMDPs, against the approximation error introduced by caching and re-using the computed policies of the ROIs—the appropriate ROI size discretization is computed autonomously. We aim to enable the autonomous analysis of other factors involved in the trade-off between reliability and efficiency.

Though we have analyzed the human-robot interactions in a probabilistic framework, some parts of the system may be fully observable, for instance inferring relationships (e.g. "left of") between objects that are distinctly separated from each other. Incorporating a hybrid combination of probabilistic and non-probabilistic planners would make the hierarchical solution more tractable for the uncertain aspects of the system.

Overall, our work has opened up a promising direction of research for efficient processing management. Ultimately, the aim is to enable robots to use a combination of learning and planning to respond autonomously and efficiently to a range of queries and commands in several human-robot interaction domains.

References

1. Michael Brenner and Bernhard Nebel. Continual Planning and Acting in Dynamic Multiagent Environments. *Journal of Autonomous Agents and Multiagent Systems*, 2008.
2. J. Casper and R. R. Murphy. Human-robot Interactions during Urban Search and Rescue at the WTC. *Transactions on SMC*, 2003.
3. R. Clouard, A. Elmoataz, C. Porquet, and M. Revenu. Borg: A knowledge-based system for automatic generation of image processing programs. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 21(2):128–144, 1999.
4. T. Darrell. Reinforcement Learning of Active Recognition Behaviors. Technical report, Interval Research Technical Report 1997-045, extended version of paper in NIPS-95 pp.858-864, 1997.
5. R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley Publishers, 2nd edition, 2000.
6. Nick Hawes et al. Towards an Integrated Robot with Multiple Cognitive Functions. In *AAAI*, 2007.
7. P. F. Felzenswalb and D. P. Huttenlocher. Efficient Graph-Based Image Segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

8. Amalia F. Foka and Panos E. Trahanias. Real-time Hierarchical POMDPs for Autonomous Robot Navigation. In *IJCAI Workshop on Reasoning with Uncertainty in Robotics*, 2005.
9. Eric A. Hansen and Rong Zhou. Synthesis of Hierarchical Finite-State Controllers for POMDPs. In *ICAPS*, pages 113–122, 2003.
10. Hokuyo. Hokuyo Laser, 2008. `http://www.hokuyo-aut.jp/products/urg/urg.htm`.
11. X. Jiang and H. Bunke. Vision planner for an intelligent multisensory vision system. Technical report, University of Bern, Switzerland, 1994.
12. Leslie Kaelbling, Michael Littman, and Anthony Cassandra. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, 101:99–134, 1998.
13. L. Li, V. Bulitko, R. Greiner, and I. Levner. Improving an Adaptive Image Interpretation System by Leveraging. In *Australian and New Zealand Conference on Intelligent Information Systems*, 2003.
14. D. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision (IJCV)*, 60(2):91–110, 2004.
15. S. Moisan. Program supervision: Yakl and pegase+ reference and user manual. Rapport de Recherche 5066, INRIA, Sophia Antipolis, France, December 2003.
16. R. Petrick and F. Bacchus. Extending the Knowledge-Based approach to Planning with Incomplete Information and Sensing. In *ICAPS*, pages 2–11, 2004.
17. Joelle Pineau and Sebastian Thrun. High-level Robot Behavior Control using POMDPs. In *AAAI*, 2002.
18. C. Shekhar, S. Moisan, and M. Thonnat. Use of a real-time perception program supervisor in a driving scenario. In *Intelligent Vehicle Symposium '94*, Paris, France, Oct 1994.
19. T. Smith and R. Simmons. Point-based POMDP Algorithms: Improved Analysis and Implementation. In *UAI*, 2005.
20. M. Sridharan, J. Wyatt, and R. Dearden. HiPPo: Hierarchical POMDPs for Planning Information Processing and Sensing Actions on a Robot. In *International Conference on Automated Planning and Scheduling (ICAPS)*, September 14-18 2008.
21. M. Thonnat and S.Moisan. What can program supervision do for program reuse? *IEE Proc. Software*, 147(5):179–185, 2000.
22. Sebastian Thrun. Stanley: The Robot that Won the DARPA Grand Challenge. *Journal of Field Robotics*, 23(9):661–692, 2006.
23. Marc Toussaint, Laurent Charlin, and Pascal Poupart. Hierarchical POMDP Controller Optimization by Likelihood Maximization. In *Uncertainty in AI (UAI)*, 2008.
24. Videre Design Camera. `http://www.videredesign.com/stereo_on_a_chip.htm`.
25. ZMDP Planning Code, 2008. `http://www.cs.cmu.edu/~trey/zmdp/`.