Research Article

Bootstrap Learning and Visual Processing Management on Mobile Robots

Mohan Sridharan

Department of Computer Science, Texas Tech University, Lubbock, TX 79409, USA

Correspondence should be addressed to Mohan Sridharan, mohan.sridharan@ttu.edu

Received 1 October 2009; Accepted 10 November 2009

Abstract

A central goal of robotics and AI is to enable a team of robots to operate autonomously in the real world and collaborate with humans over an extended period of time. Though developments in sensor technology have resulted in the deployment of robots in specific applications the ability to accurately sense and interact with the environment is still missing. Key challenges to the widespread deployment of robots include the ability to learn models of environmental features based on sensory inputs, bootstrap off of the learned models to detect and adapt to environmental changes, and autonomously tailor the sensory processing to the task at hand. This paper summarizes a comprehensive effort towards such bootstrap learning, adaptation, and processing management using visual input. We describe probabilistic algorithms that enable a mobile robot to autonomously plan its actions to learn models of color distributions and illuminations. The learned models are used to detect and adapt to illumination changes. Furthermore, we describe a probabilistic sequential decision-making approach that autonomously tailors the visual processing to the task at hand. All algorithms are fully implemented and tested on robot platforms in dynamic environments.

1. Introduction

An open grand challenge in the field of robotics is to enable widespread deployment of robots in the real world, where they can operate autonomously and collaborate with humans. Addressing this grand challenge would in turn require answers to the following major questions.

(i) **Autonomous Learning and Adaptation.** How to enable a robot to autonomously learn models of environmental features based on sensory input, detect environmental changes, and adapt the learned models in response to such changes?

(ii) **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?

(iii) **Multiagent Coordination.** How to enable a team of robots, each with possibly different capabilities and constraints, to collaborate robustly towards a shared objective despite noisy sensing and communication?

In this paper, the focus is primarily on developing probabilistic methods for Autonomous Learning and Adaptation, and for Processing Management. We propose probabilistic methods that enable a robot to use sensory inputs to learn environmental models and respond to environmental changes. Furthermore, given multiple sources of information, the robot autonomously tailors the sensory processing to the task at hand.

Mobile robots that sense and interact with the environment through a set of sensors and actuators are characterized by the following features and requirements.

(a) **Partial Observability.** The true state of the world is not directly observable. The robot can only update its belief, that is, an estimate of the world state by executing actions and observing the noisy outcomes.

(b) **Nondeterministic Actions and Observations.** The outcome of executing actions or making observations based on sensory input is nondeterministic, that is, actions and observations are unreliable.
(c) **Computational Complexity.** Many state-of-the-art sensory processing algorithms (e.g., vision algorithms) have high computational complexity.

(ii) Requirements.

(a) **Dynamic Performance.** Robots operating in the real world need to respond to the changes in their environment despite computational constraints; that is, there is a strong real-time requirement.

(b) **Reliability.** Though outcomes of actions and observations are nondeterministic, the robot needs to operate with a high degree of reliability, especially in critical applications such as disaster rescue or surveillance.

Developments in sensor technology [1, 2] have resulted in the deployment of mobile robots in specific applications such as disaster rescue, navigation, and medicine [3–6]. The ability to accurately sense and interact with the environment is however still lacking. The state of the art in mobile robotics is hence far from achieving autonomous operation over a range of domains. Real world environments change in ways that cannot be specified in advance, while most sensors mounted on mobile robots require a time-consuming manual calibration phase before deployment. In addition, this calibration is sensitive to environmental changes. Furthermore, a robot can process the inputs from its multiple sensors using a set of algorithms, each of which may have a different reliability and computational complexity. Processing all the information would be infeasible in dynamic domains where real-time operation is essential.

The above-mentioned challenges are all the more pronounced in the case of visual input from color cameras. A color camera provides higher bandwidth information than range sensors (laser, sonar, etc.) at a much lower cost. Visual input is however more noisy and sensitive to environmental factors such as illumination, and visual information processing algorithms are typically computationally expensive. Until recently, many mobile robot applications have therefore relied on range sensors [7, 8]. Even the approaches that consider visual input make most high-level decisions based on other sources of input [6, 9], or only use the limited information obtained from intensity images [10]. A rich source of information is hence not fully exploited.

One factor that can be utilized to offset the challenges listed above is the presence of a moderate amount of structure in many mobile robot environments. Examples of such structure include known positions and properties (e.g., size, shape) of unique objects in the environment, information which can be manually provided or automatically inferred by the robot. This structure can be exploited to enable autonomous operation on mobile robots. Our work represents a comprehensive effort towards such learning, adaptation, and processing management using the input from color cameras as the primary source of information. The work on autonomous learning and adaptation focuses on color as the feature of interest and illumination as the environmental factor that changes over time. The work on processing management considers several sources of information that are based on visual input. Specifically, this paper summarizes the following contributions.

(i) A probabilistic **bootstrap** learning framework that enables a robot to plan its actions in order to learn models of color distributions and illumination conditions in its environment [11]. The robot uses these learned models to detect and adapt to illumination changes [12].

(ii) A **probabilistic sequential decision-making framework** which enables a robot to autonomously tailor its visual information processing to the task at hand [13].

These algorithms are tested on specific robot platforms and have the potential of generalizing to other applications.

The remainder of this paper is organized as follows. Section 2.1 summarizes a typical robot vision system, while Section 2.2 describes the test platforms used. Section 3.1 describes the related work in the areas of color segmentation, color learning, and illumination invariance, followed by an overview of AI planning methods as applied to robot vision tasks (Section 3.2). Next, Section 4.1 describes our proposed approach for autonomous illumination-invariant color learning, while Section 4.2 presents the approach for visual processing management based on probabilistic sequential decision processes. Finally, Section 5 summarizes the conclusions and directions for further research.

2. Baseline Vision System and Test Platforms

In this section, we present an overview of a typical robot vision system, followed by a description of the test platforms used to evaluate the proposed algorithms.

2.1. **Baseline Robot Vision System.** Figure 1 shows a flowchart of the typical robot vision system that uses color information. Color segmentation is typically the first step, where the goal is to cluster image regions into *similar* groups and/or to create a mapping from pixel values to discrete color labels, that is, to create a **color map**

\[
\Pi_E : \{m_1, m_2, m_3\} \rightarrow I_e [0, N - 1],
\]  

(1)

where \(m_1, m_2, m_3\) are the values along the color channels (e.g., R, G, B) that can take values in \([\text{min-max}]\) \((0-255\) for RGB), the subscript \(E\) represents the dependence on illumination, and \(I\) refers to the numerical indices of the color labels (e.g., blue = 1, orange = 2). This mapping is typically generated by extensive manual labeling of image regions.

The (color) segmented image regions are used to find “objects” and other desired structures using heuristics and constraints based on the known properties (size, shape, color, etc.) of the target objects. The detected objects and their locations in the image can be used along with other inputs (e.g., depth map from a stereo camera) for creating
a 3D model of the scene. On mobile robots, the relative distances and bearings of the detected objects can be used in a localization module that computes the position and orientation (i.e., pose) of the robot in the global frame of reference. The 3D model of the scene and/or the pose information is used by the robot to determine the high-level behavior suitable to achieve the desired goal (e.g., navigate to a location to retrieve an object). See [14] for an instance of such a robot vision system.

In order to operate robustly in dynamic environments, a robot has to deal with unexpected changes autonomously and efficiently. For instance, Figure 2 shows that the color map trained under one illumination results in poor segmentation under a different illumination. Robots however frequently have to operate in domains with changing illumination. Section 4.1 summarizes our approach for autonomous learning of color models and adaptation to illumination changes.

A mobile robot equipped with multiple sensors can process its sensory inputs using many algorithms that may have varying levels of uncertainty and computational complexity. Hence, a key requirement for autonomous operation is the ability to tailor the sensory processing to the task at hand. Section 4.2 describes an instance of such processing management of visual input using probabilistic sequential decision processes. The overall goal of our research is to enable autonomous mobile robot operation in a wide range of applications.

2.2. Test Platforms. In this work, we use two different test platforms to evaluate the algorithms: a four-legged robot in the robot soccer scenario, and a mobile robot playmate in a human-robot interaction scenario.

2.2.1. Robot Soccer. The color learning and illumination invariance methods were evaluated on the SONY ERS-7 Aibo, a four-legged robot whose primary sensor is a CMOS camera at the tip of its nose, with a limited field of view (56.9° horz., 45.2° vert.). The images are captured at 30 Hz with a resolution of 208 × 160 pixels. The robot has 20 degrees of freedom, three in each leg, three in its head, and the rest in its tail, mouth, and ears. It has wireless LAN for communication with an off-board PC or other robots. However, all processing for vision, localization, motion and strategy is performed on-board using a 576 MHz processor.

One major application domain for the Aibos is the RoboCup Legged League [15], an international research initiative where teams of four robots play a competitive game of soccer on a (4 m × 6 m) indoor field. As with other robots equipped with cameras, the vision system on the Aibo has an initial color calibration phase. The calibration includes extensive manual labeling of appropriate regions in the images captured by the robot’s camera, in order to obtain the color map (as described in Section 2.1). This manual labeling is a major challenge to autonomous operation, and the color map is sensitive to illumination changes—see Figure 2. Figures 3(a) and 3(b) show images of the Aibo and the soccer environment.

2.2.2. Robot Playmate. The visual processing management experiments were conducted on a mobile robot playmate that collaborates with a human to jointly manipulate and converse about objects on a tabletop [16] as shown in Figure 4(b). The robot is equipped with a stereo camera (640 × 480 images at 30 Hz), manipulator arm, on-board processors, and other sensors. The domain, though seemingly simple, represents the state of the art in cognitive robotics [17]. The processing cycle in this domain is different from the flowchart described in Section 2.1—modules operating in parallel process the vision and speech inputs, and create goals that are achieved by other modules such as manipulation. Visual processing in this domain, however, is characterized by the same features and requirements as the robot soccer scenario.

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 understand spatial relations between objects; to recognize actions performed by humans. Each of these vision tasks is a hard problem in itself, but we are faced with the formidable challenge of building a vision system capable of performing all of them. Consider the scene in Figure 4(a)
with rectangular regions of interest (ROI) extracted from the background. The robot has to use a subset of the available visual routines to execute commands or answer queries: “is there a blue triangle in the scene?”, “move the mug to the right of the circle”. However, it is neither feasible nor desirable to run all the routines on each image, since the robot has to respond to dynamic changes.

3. Related Work

This paper focuses on learning, adaptation, and processing management using visual input. The topics of interest such as color learning, illumination invariance, and planning of visual processing continue to be extensively researched in the fields of computer vision, robot vision, and planning. This section therefore reviews a set of representative methods for these topics and analyzes the approaches in terms of their applicability to robots operating in dynamic environments.


Color segmentation is a well-researched field in computer vision with several good algorithms [18–20]. The mean-shift algorithm is a nonparametric technique for the analysis of complex multimodal feature spaces and the detection of arbitrarily shaped clusters [18]. The feature space is modeled as an empirical probability density function (pdf). Dense regions in the feature space correspond to the modes of the unknown pdf. Once the modes are found, the clusters can be separated based on the local structure of the feature space. Mean-shift provides good performance on tasks such as segmentation and tracking, but its quadratic complexity makes it expensive to perform on robots with computational constraints.

Active contours are another set of popular methods for image segmentation [20–22]. The method defines initial contours and then deforms them towards object boundaries.

Manjunath et al. describe a region-based method [20] that segments images into multiple regions and integrates an edge-flow vector field-based edge function for segmenting precise boundaries. The method allows the user to specify the similarity measure based on features such as color or texture. The algorithm is not sensitive to the initial estimates and provides good segmentation results on a variety of images, but the iterative optimization is expensive to perform on robots.

Image segmentation can also be posed as a graph-partitioning problem, where each node represents a pixel in the image, and the edges connect certain pairs of neighboring pixels [19, 23]. Typically, graph-based segmentation methods find minimum cuts in the graph, where a cut measures the degree of dissimilarity between point sets by computing the weights of the graph edges that have to be removed to separate the two sets. Shi and Malik proposed the popular normalized cut algorithm, a robust global criterion that simultaneously maximizes the similarity within a cluster and the dissimilarity between clusters [19]. Normalized cuts have been used for computer vision tasks such as motion tracking [24] and 3D view reconstruction [25], but the approach is computationally expensive for robot platforms.

In the RoboCup domain, the typical approach is to create mappings (1) from the YCbCr values to the color labels [26]. Other methods include the use of decision trees [27] and axis-parallel rectangles in the color space [28]. These approaches involve the hand-labeling of images over a period of an hour or more before the color map can be generated (Section 2.1). Attempts to learn colors or make them independent to illumination changes involve the knowledge of the spectral reflectances of the objects under consideration and/or require additional transformations that are computationally expensive to perform on robots [29, 30].

An important consideration in color learning and segmentation is the choice of color space. However, there is
a lot of controversy on the “best” color space for different applications. In order to address this challenge, Gevers and Smeulders evaluated several color spaces to determine their suitability for recognizing multicolored objects invariant to significant changes in viewpoint, object geometry, and illumination [30]. They presented a detailed theoretical and experimental analysis of the several models. This research hence provides a good reference on the choice of color spaces.

Attempts to automatically learn the color map in the legged league have rarely been successful. Cameron and Barnes [31] present an approach that detects edges in the image and constructs closed figures to find image regions corresponding to known objects. The color information from these regions was used to build the color classifiers. Illumination changes are tracked by associating the current classifiers with the previous ones. This approach is time consuming even with the use of off-board processing. Jungel presents another approach where the color map is learned using three layers of color maps with increasing precision levels [32], with colors in each level being represented as cuboids. The colors are defined relative to a reference color (field green in the soccer domain) that is tracked with minor illumination changes, and all other color distributions are displaced in the color space by the same amount. However, different colors do not actually shift by the same amount with illumination changes, and hence the color map is reported to be not as accurate as the hand-labeled one. Unlike these prior approaches, our algorithm exploits domain knowledge to model color distributions and learn a color map in ≈6 minutes of robot time, resulting in performance comparable to the color map obtained after hours of human effort.

While learning can automate the color map generation, the learned map is still sensitive to illumination changes. The response obtained at a sensor can be defined as [33]

\[ m_j^p = \int (E(\lambda)S^p(\lambda)R_j(\lambda))d\lambda, \]

where \( E(\lambda) \) is the spectral power distribution of the illuminant, \( S^p(\lambda) \) is the surface reflectance at a scene point \( p \), while \( R_j(\lambda) \) is the spectral response (relative) of the imaging device’s \( j \)th sensor. The response of the \( j \)th sensor of the camera at pixel \( p \), \( m_j^p \), is the integral of the product of these three terms over the range of wavelengths. Changing the surface reflectance or the spectral power distribution of the illuminant can change the sensor response. Color constancy or illumination invariance is the ability to assign the same symbolic labels to color distributions despite illumination changes. Decades in computer vision have resulted in several methods for color constancy, most of which focus on static images and have high computational complexity.

The Retinex theory [34] is based on the assumption that white reflection induces maximal rgb camera responses, and uses the maximum r, g, and b responses as an estimate of
the illuminant. It was later modified to be based on global or local image color averages—the “Gray World” algorithm [35] is based on the same principle. However, the local or global image averages correlate poorly with the actual illuminant [36].

The gamut mapping algorithm [37] proposed by Forsyth is based on the fact that surfaces can reflect no more light than what is cast on them. The illuminant color is hence constrained by the colors observed in the image, and can be estimated using image measurements alone. The algorithm selected the most likely mapping from a set of mappings that transformed the sensor values under an unknown illuminant to the gamut of colors observed under a canonical illuminant. Finlayson proposed the median selection method that included a constraint on the possible color of the illuminant into the gamut mapping algorithm [38]. The more recent correlation framework [33] measures the likelihood that each of a possible set of illuminants is the scene illuminant. However, these approaches require prior knowledge of the illuminations which is not feasible in robot scene illuminant. However, these approaches require prior knowledge of the illuminations which is not feasible in robot domains.

Brainard and Freeman use a Bayesian decision framework, which combines statistics such as gray world, subspace, and physical realizability constraints [39]. They generate a priori distributions to describe the probability of existence of certain illuminants and surfaces. A maximum local mass (MLM) estimator integrates local probabilities and uses Bayes’ rule to compute the posterior distributions for surfaces and illuminants, for a given set of photosensor responses. However, significant prior knowledge of illuminants and other statistics is required, and the approach is computationally expensive. Tsin et al. present a Bayesian maximum a posteriori (MAP) approach for outdoor object recognition with a static surveillance camera [40]. Static overhead high-definition images collected over several days are used to learn models of reflectance and the light spectrum. A linear iterative scheme converges to the classification result on the test images. A mobile robot system, however, has to be robust to camera motions and dynamic changes.

On robots, the color constancy problem has often been avoided by using nonvisual sensors such as range finders [8]. Even when visual input is considered, the focus is on recognizing well-separated colors [3]. There has been little work on color constancy in the presence of shadows and artifacts due to rapid camera motion. Further, with few exceptions (e.g., [41, 42]), most methods do not function in real time.

Schulz and Fox estimate colors using a hierarchical Bayesian model with Gaussian priors and a joint posterior on position and environmental illumination [43]. Even when tested under two distinct illuminations and a small set of colors, the approach requires prior knowledge of color distributions and illuminations, in addition to being computationally expensive. Lenser and Veloso present a tree-based state description technique [41] for detecting changes in lighting on Aibo robots. A time-series of average screen illuminance is used to distinguish between illumination conditions. We however believe that the color space distributions could function as a better discriminating feature. Anzani et al. describe an attempt at illumination invariance in the RoboCup middle-size league [42], where a Mixture of Gaussians (MoG) is used to generate multimodal distributions for the various colors. The EM algorithm [44] is used with online adaptation of the number of mixture components, in order to adapt to minor illumination changes. However, the labeling of color classes and association with mixture components is done by human supervision, and the algorithm has been tested only over a few illuminations in the lab. In the recent DARPA grand challenges, Thrun [6] modeled colors as MoG and attempt to add additional Gaussians and modify the parameters of the existing Gaussians in response to the changes in illuminations. However, they were interested only is distinguishing safe regions on the ground from the unsafe regions and did not have to model overlapping color classes separately.

Section 4.1 summarizes our approach for modeling illuminations and overlapping colors without prior knowledge of color distributions. The learned models are used to detect and adapt to a range of illuminations. See [45] for a recent survey on color learning and illumination invariance.

3.2. Visual Processing Management. AI planning and cognitive planning architectures are well-researched fields [46–49]. The focus here is on a specific subcategory of the planning problem: the joint planning of the sensing (where to look) and information processing (what to look for) actions to achieve a desired goal.

Classical planning methods use deterministic models and require prior knowledge of state, action outcomes, and all contingencies. Many modern planning methods extend the machinery of classical planning in order to model the nondeterminism inherent in perception. Draper et al. [50] proposed C-BURIDAN, a planning scheme that incorporates a probabilistic model of the noisy sensors and effectors, while still retaining a symbolic STRIPS-like representation of action effects [51]. The plan-assessment phase treats actions as probabilistic state transitions, while the plan-refinement phase links the symbolic action effects to the symbolic subgoals of the desired goal state. Their formulation is similar to our POMDP approach as they reason about the best action to perform based on prior belief about the world and the observations obtained from action execution. However, their approach requires the action preconditions and effects to be manually specified, does not incorporate a notion of action costs, and requires a manual ordering of actions to accumulate belief from repeated execution of the same action.

In contrast to the C-BURIDAN system, Petrick and Bacchus’s PKS planner [52] describes actions in a first-order language, in terms of their effect on the agent’s knowledge rather than their effect on the world. The model is hence nondeterministic in the sense that the true state of the world may be determined uniquely by the actions performed, but the agent’s knowledge of that state is not. For example, dropping a fragile item will break it, but if the agent does not know that the item is fragile, it must use an observational action to determine its status. PKS captures
the initial state uncertainty and constructs conditional plans based on the agent’s knowledge. More recently, Brenner and Nebel [53] proposed the Continual Planning (CP) approach, based on the FF planner [54], which interleaves planning, plan execution, and monitoring. Unlike classical planning, an agent in CP postpones reasoning about unknowable or uncertain states until more information is available. Actions are allowed to assert that the preconditions for the action will be met when the agent reaches that point in the execution of the plan. If these preconditions are not met during execution, or are met earlier, replanning is triggered. CP is therefore similar to PKS in representation but works by replanning rather than constructing conditional plans. There is however no representation of the uncertainty in the observations and actions. In applications where observations are noisy, the optimal behavior may be to increase the confidence in the image interpretation by running the operators more than once on several images of a scene, and accumulating the evidence. This cannot be readily represented in the approaches described above.

There is a significant body of work in the image processing community on planning of visual operations [55–57]. Such approaches typically use a classical planner that takes a user-specified high-level goal and constructs a pipeline of image processing operations. The planners use deterministic models, actions represented as STRIPS-like operators with prespecified preconditions and effects, and domainspecific rules for evaluating the output of each operator. Unsatisfactory results are handled by re-planning the operator sequence or modifying the operators’ parameters [57, 58].

In the field of computer vision, probabilistic sequential decision processes, that is, Markov Decision Processes (MDPs) and Partially Observable MDPs (POMDPs), have been used for image interpretation. Darrell [59] used memory-based reinforcement learning and POMDPs to learn to foveate salient body parts in an active gesture recognition system. The action set consists of foveation actions and a special recognition action. During the learning phase, execution of the recognition action is followed by manual feedback on the target object’s presence in the scene, so that each action sequence can be assigned a reward. Reinforcement learning is used to learn what foveation actions to execute, and when to execute the terminal recognition action. More recently, Li et al. [60] posed image interpretation as an MDP, using human-annotated images in an offline process to determine the reward structure by applying all possible sequences of image operators to the labeled images. Dynamic programming methods are used to determine the value function for the explored parts of the state space, which is then extrapolated to the entire state space using an ensemble learning technique. During online execution, each step consists of feature extraction and the choice of an action that maximizes the learned value functions. Approaches that require manual feedback in an initial training phase are too time-consuming to use on a robot interacting with a human. It would instead be desirable to autonomously generate the models and policies.

Sequential decision processes have also been used for planning a sequence of gaze locations (image ROIs) that are analyzed to identify the desired target. In the recent work, Vogel and de Freitas [61] have posed gaze sequence selection as a finite-horizon sequential decision process that elegantly combines bottom-up saliency, top-down target knowledge, and spatial target context. The gaze planning strategy was tested on image databases to determine the location of computer monitors. Though this approach requires a prior distribution over object locations that is difficult to compute for multiple objects in practical scenes, it clearly demonstrates the benefits of visual processing management.

There has also been considerable related work in the field of active sensing, where the goal is to decide on sensor placement and sensor information processing based on its relevance to the task at hand [62, 63]. Kreucher et al. [62] presented an active sensing approach for scheduling sensors in order to learn the number and states of a group of moving targets in a surveillance region. The joint multitarget probability density is estimated using a particle filter. A sensing action is chosen (at each time step) based on the Renyi-divergence measure, and the probability density of the number and states of the targets is updated. However, estimating the joint probability density requires considerable prior information that is difficult to obtain in robot environments. Our visual processing management approach (Section 4.2) is also focused on computing a sequence of operations for a specific task. However, our approach uses automatic belief propagation to enable the robot to respond to dynamic changes.

Recently, there has been considerable interest in the use of submodular functions for sensor placements in spatial phenomena modeled as Gaussian processes [64, 65]. For objective functions that can be represented as submodular functions, the greedy policy provides performance that is at least 63% of optimal performance [65]. However, our approach is significantly similar to methods that aim to maximize the information gain, and such approaches cannot be represented using submodular functions [64].

Since POMDP solutions of practical-sized problems are typically intractable, several researchers have focused on imposing structure in POMDP formulations in order to make it more tractable. Pineau et al. [4, 66] propose a hierarchical POMDP approach for high-level behavior control on a nursing assistant robot, similar to the MAXQ decomposition for MDPs [67]. They impose an action hierarchy, with the top level action being a collection of simpler actions that are represented by smaller POMDPs. A hierarchical planning algorithm operates in a bottom-up manner, finding complete solutions, that is, policies for the smaller POMDPs. The execution proceeds in a top-down manner: invoking the policy at the top-level recursively traverses the hierarchy invoking a sequence of local policies until a primitive action is reached. Model parameters at all levels are defined over the same space of states, actions, and observations, but the relevant space is abstracted for each POMDP using a dynamic belief network. Hansen and Zhou [68] propose a similar Task Hierarchy (TH) for planning with POMDPs, where the policies are defined as finite-state
controllers (FSCs) and the dynamic programming policy of a subproblem is treated as an abstract action in the next higher level POMDP. The difference is that each POMDP in the hierarchy is an indefinite-horizon POMDP in order to allow FSC termination without recognition of the underlying terminal state. Similar systems have also been proposed for autonomous robot navigation [69–71]. In the actual application, however, a significant amount of data for the hierarchy and model creation has to be hand-coded.

There has been considerable work on exploring representations for hierarchical POMDPs that allow for tractable performance in practical applications. Theodoridis et al. [72] represented hierarchical POMDPs as dynamic Bayesian networks (DBNs), for the specific task of using multiresolution spatial maps for indoor robot navigation. They have shown that the DBN representation can train faster (and with fewer samples) than the hierarchical POMDP or the flat POMDP. More recent work by Toussaint et al. [73] aims to learn the hierarchical representation of a POMDP based on maximum likelihood estimation, using dynamic Bayesian networks and parameter estimation based on Expectation-Maximization. However, these approaches require considerable manual supervision, or are computationally expensive to use on robots.

Similar to existing approaches, the summarized hierarchical POMDP approach defines the higher level model parameters as functions of the lower level policies. However, automatic belief propagation is achieved through a functional decomposition that considers images of regions in space at the higher level, and the corresponding image ROIs at the lower level. Incorporating additional operators or ROIs is hence easier.

4. Proposed Approaches

This section summarizes our work on autonomous learning, adaptation, and visual processing management. Section 4.1 describes the algorithm that enables a robot to use autonomously learned color and illumination models to detect and adapt to illumination changes. Next, Section 4.2 presents the algorithm that enables a robot to autonomously tailor its visual processing to the task at hand.

4.1. Planned Illumination-Invariant Color Learning. As described in Section 2.1, the manual calibration of the color map is time-consuming and sensitive to illumination changes. However, as with many other application domains, the robot on the soccer field knows (or can infer) a significant amount of the structure in its environment—it knows the positions and color labels of the objects of interest (e.g., goals, markers, etc.). Here, we describe an approach that enables the robot to exploit this known structure to do the following.

(i) Learn models of color distributions and illuminations, which can be refined incrementally.

(ii) Use the learned models to detect and adapt to a range of illumination changes.

The overall algorithm is summarized in Algorithm 1—specific line numbers are referenced in the text below. The robot initially has no prior information of color distributions or illumination. It has an algorithm that exploits the known structure of the environment to plan a motion sequence for learning the color map—see Algorithm 2.

The first question to address is what to learn? That is, we need to decide on the appropriate models for color distributions and illuminations. We use a disjunctive representation that models the a priori probability density function (pdf) for each color (l) either as a 3D Gaussian or as a 3D Histogram.

\[
p(m \mid l) \sim N(\mu_l, \Sigma_l), \quad \text{or} \quad \frac{\text{hist}(b_1, b_2, b_3)}{\sum \text{hist}}, \quad (3)
\]

where \((b_1, b_2, b_3)\) are the histogram bin indices corresponding to the color channel values \(m = (m_1, m_2, m_3)\). The histogram is normalized to obtain a pdf. Assuming all colors are equally likely, that is, \(P(l) = 1/N, \forall l \in [0, N-1]\), each color’s a posteriori pdf is proportional to the a priori pdf. The color space is discretized and each color map cell is assigned the label of the most likely pdf. In a given situation, one of the two models may be more suitable for a color’s pdf, and the choice is made autonomously using the bootstrap test [74]. Though other color models are feasible, the disjunctive model provides a balance between accuracy and computation.

Each illumination is represented by a color map and autonomously-collected image statistics. Based on the hypothesis that images from the same illumination have measurably similar distributions of pixels in color space, images captured by the robot are transformed into the normalized RGB space \((r, g, b)\). Histograms in \((r, g, w)\), with 64 bins in each dimension, are normalized to obtain pdfs \((rgHist_{\text{illum}})\) that form the first statistic. The distance between every pair of pdfs is computed and the distribution of distances \((D_{\text{illum}})\), modeled as a Gaussian, constitutes the second statistic. The Jensen-Shannon (JS) measure is used for computing the distance between the distributions.

\[
\text{JS}(a, b) = \frac{\text{KL}(a, m) + \text{KL}(b, m)}{2},
\]

\[
\text{KL}(a, b) = \sum_i \sum_j \left( a_{i,j} \cdot \ln \left( \frac{a_{i,j}}{b_{i,j}} \right) \right), \quad m = \frac{a + b}{2}. \quad (4)
\]

The JS distance is a function of the log of the pdfs \((a, b)\) and is hence robust to peaks in the distributions, that is, large image regions of a single color.

A mobile robot typically has to operate in environments where the illumination changes unpredictably. Given the models for color distributions and illumination, the next question to address is: how to detect and adapt? That is, how to detect illumination changes and adapt to them. Major illumination changes, for instance, when the lamps are suddenly switched on (or off), cause large shifts in color distributions. The current color map is no longer valid and the robot is soon lost. Minor (or slow) illumination changes, for instance, the variation in natural light during the day,
require: for each known illumination \( E_i, i \in [0, M - 1] \), color map \( \Pi_{E_i} \), \((r, g)\) distributions \( r g Hist_{E_i} \), and distribution of \( JS-\)distances \( D_{E_i} \).

require: algorithm to plan motion and learn colors autonomously (algorithm 2).

require: positions, shapes and color labels of the objects of interest in the robot’s environment. Initial robot pose.

1. initialize: \( M = 0 \), \( illum = 0 \), \( testTime = 0 \) (no prior illumination knowledge).

2. plan motion and learn \( \Pi_{E_{illum}} \).

3. generate \( r g Hist_{E_{illum}} \), \( N(r, g) \) space distributions, and distribution of \( JS-\)distances \( D_{E_{illum}} \), using images captured at random during color learning.

4. save image statistics, \( M = M + 1 \).

5. while true do

6. get new image. segment image and detect objects.

7. if minorChange \( (\text{Color}) \) then

8. minorUpdate \( (\text{Color}) \). get \( \hat{\Pi}_E \) from current color distributions.

9. revise current illumination representation to get \( r g Hist_{\hat{E}} \) and \( D_{\hat{E}} \), to be used for subsequent operations.

10. end if

11. if \( currentTime - testTime \geq time_{th} \) then

12. \( rgDist = (r, g) \) distribution of current image.

13. for \( i = 0 \) to \( M - 1 \) do

14. \( dAvg[i] = (1/N) \sum_j JSDist(rgDist, r g Hist_{E_i}[j]) \)

15. end for

16. if exists \( \hat{E} \) then

17. \( dAvg_{\hat{E}} = (1/N) \sum_j JSDist(rgDist, r g Hist_{\hat{E}}[j]) \)

18. end if

19. if exists \( \hat{E} \) and withinRange \( (dAvg_{\hat{E}}, D_{\hat{E}}) \) then

20. continue with \( \Pi_{\hat{E}} \).

21. else if withinRange \( (dAvg[illum], D_{illum}) \) then

22. continue with \( \Pi_{illum} \).

23. else if withinRange \( (dAvg[i], D_{E_i}), i \neq illum \) then

24. use \( \Pi_{E_i}, illum = i \).

25. else

26. new illumination, \( illum = M, M = M + 1 \).

27. learn \( \Pi_{illum} \) autonomously.

28. learn \( r g Hist_{illum} \) for new illumination.

29. use \( \Pi_{illum} \) for subsequent operations.

30. end if

31. \( testTime = \text{currentTime} \).

32. end if

33. end while

algorithm 1: illumination adaptation algorithm.

cause the robot’s segmentation to slowly deteriorate as the color distributions shift.

for each object detected from the color segmented image regions, the robot computes

\[
\frac{\text{numPixels}_l}{\text{totalPixels}} \leq \text{changeThreshold},
\]

where \( \text{numPixels}_l \) represents the pixels of the color label \( (l) \) of the detected object, and \( \text{totalPixels} \) is the total number of pixels within the object’s bounding rectangle. if the value of this ratio falls below a threshold consistently (for \( \geq 60\% \) of \( N \) consecutive frames) it indicates a minor illumination change, denoted by \( \text{DetectMinor} \) \( (\text{minorChange}()) \)—line 7). the new illumination is denoted by \( \hat{E} \). the pixels within the corresponding image region are used to build a new model (i.e., a histogram or a gaussian) for the color distribution, which is merged with the current model for that color \( (\text{minorUpdate}()) \)—line 8). for gaussians, we use
the measurement update of a Kalman Filter [75]

\[
Gain \ K_l = \Sigma_l (\Sigma_l + \Sigma_{l_{\text{new}}})^{-1},
\]

\[
\mu_{l_{\text{up}}} = \mu_{l_{\text{old}}} + K_l (\mu_{l_{\text{new}}} - \mu_{l_{\text{old}}}),
\]

\[
\Sigma_{l_{\text{up}}} = (I - K_l) \Sigma_{l_{\text{old}}},
\]

where the subscripts \text{old}, \text{new}, and \text{up} represent the current, new, and updated model, respectively, for color \text{l}. For histograms, a weighted average is computed

\[
P_{l_{\text{old}}} = \frac{\text{hist}_{l_{\text{old}}}}{\sum \text{hist}_{l_{\text{old}}}}, \quad P_{l_{\text{new}}} = \frac{\text{hist}_{l_{\text{new}}}}{\sum \text{hist}_{l_{\text{new}}}},
\]

\[
P_{l_{\text{up}}} = \frac{P_{l_{\text{old}}} w_{\text{old}} + P_{l_{\text{new}}} w_{\text{new}}}{w_{\text{old}} + w_{\text{new}}}, \quad w_{\text{old}} + w_{\text{new}} = 1,
\]

for merging the normalized histograms with the existing normalized histograms to obtain the updated histograms. The weights are based on the number of samples in the corresponding histograms. The color map and the current illumination model are modified and used in subsequent operations (line 9). This adaptation scheme is called \text{Adapt}_{\text{minor}}.

In order to detect sudden illumination changes, the robot periodically (time_{ch} = 0.5 \text{seconds}) generates a test image histogram in the \((r,g)\) space (line 12). The average distance \(d_{\text{Avg}}\) is computed between this test histogram and the set of histograms corresponding to each illumination for which a representation has been learned (lines 13–15). Illumination representations created while tracking minor illumination changes are included in this computation (lines 16–18 in Algorithm 1).

If \(d_{\text{Avg}}\) lies within the threshold range (95\%) of the distance distribution corresponding to the current illumination (withinRange)—lines 19, 21), the robot continues to use the current color map. If \(d_{\text{Avg}}\) lies outside the range of the distance distribution of the current illumination, but within the range of the distance distribution corresponding to an illumination for which the robot has learned a model, the robot transitions to using the corresponding color and illumination models. However, if \(d_{\text{Avg}}\) lies outside the range of all known illuminations, the robot models a new illumination (Detect_{\text{major}}) and learns models for color distributions (lines 25–30). This adaptation scheme (Adapt_{\text{major}}) cannot be used with a reduced threshold to handle minor illumination changes, because it could result in a large number of color maps for changes in a few distributions. Both Adapt_{\text{minor}} and Adapt_{\text{major}} are hence necessary.

Given the algorithm to adapt to illumination changes, the final question to address is: \textit{how to learn}? That is, how to learn the color map and illumination model. As summarized in Algorithm 2 we answer this question by finding a sequence of poses \((x, y, \theta)\) the robot can move through, learning one color at each pose. The goal is to simultaneously maximize color learning opportunities while minimizing localization errors—the robot may obtain more training samples by moving a larger distance, but this motion may cause larger localization errors. This goal is achieved by discretizing the robot poses into cells, and using three components: a motion error model (MEM), a statistical feasibility model (SFM), and a search routine.

The MEM predicts the error in the robot pose in response to a motion command. The inputs are the difference between the starting pose \((x_i, y_i, \theta_i)\) and target pose \((x_f, y_f, \theta_f)\), and the list of colors the robot has learned. The output is the pose error that would be incurred during this motion. The MEM is represented as a back-propagation neural network [77] with \(N + 3\) inputs, three outputs and one hidden layer of 15 nodes

\[
\{\Delta_{x}, \Delta_{y}, \Delta_{\theta}, c_1, c_2, \ldots, c_N\} \rightarrow \{\text{err}_{x}, \text{err}_{y}, \text{err}_{\theta}\},
\]

where \(\{\Delta_{x}, \Delta_{y}, \Delta_{\theta}\}\) represent the difference in pose, and \(\{c_1, c_2, \ldots, c_{N-1}\}\) are binary variables representing the target colors. If all the colors are known, all the markers can be recognized—with only some colors known, some markers are not recognizable, and the robot’s localization suffers.

For each robot pose, the SFM provides the probability of learning each of the desired colors given that a subset of
the colors have been learned. A feasibility check based on the robot's joint angles and camera field of view eliminates a lot of cells—the robot can learn colors only when its camera is pointing towards a known object. The feasibility check is performed once for each object configuration. Each cell of the SFM stores a probability measure

\[ \text{SFM}(d, e, f, v_i) = p, \quad \forall [d, e, f] \in [0, K - 1], \]

where \( d, e, f \) are cell indices of the \( K \) discrete poses and \( v_i, i \in [0, M - 1] \) represents all possible combinations of colors.

In the training phase, the robot moves between randomly generated target poses and executes two localization routines, one with all colors known (provides ground truth) and another with only a subset of colors known. The difference of the two pose estimates provides the training samples to build the MEM (CollectMEMData(), NNetTrain() in Algorithm 2). In parallel, the robot attempts to learn colors based on the knowledge of a subset of the colors. The Gaussian-smoothed and normalized cell counts of the successful learning attempts are used to compute the SFM (CollectColLearnStats()—line 3, UpdateFM()—line 5). The SFM has to be relearned when the object configurations change, but even with just the geometric constraints the robot is able to provide motion sequences leading to successful color learning.

Given the learned models (MEM, SFM), for any given starting pose during testing, the robot generates all candidate motion sequences (GenCandidateSeq()—line 6), that is, all possible paths along the discretized pose cells. The depth of the search is equal to the number of colors to be learned—we assume that the robot learns one color at each pose. If the robot is to learn \( N \) colors, the motion sequence is

\[ \text{path: \{x, y, \theta, \text{color}_i\} \quad \forall i \in [0, N - 1].} \]

This formulation results in a large number of paths (\( \approx 10^8 \)). However, only a small subset of paths (\( \approx 10^4 \)) are evaluated completely. The MEM predicts the pose error if the robot travels from the starting pose to the first pose. The vector sum of the error and the target pose predicts the actual pose. If the desired color can be learned at this pose (high probability in SFM), the move to the next pose is evaluated. If the whole path is evaluated, the net pose error and probability of success are computed (EvalCandidateSeq()—line 7). The path that provides a high probability of success and a low pose error is executed (SelectMotionSeq()—line 8) by the robot. At each pose, the expected object location is projected on the image to extract pixels that are used to model the color distributions and learn the color map. The data for the illumination model is collected during the learning process.

4.1.1. Experimental Results. We need to evaluate the robot’s ability to (a) plan a motion sequence and learn models for color distributions and illumination for different object configurations and (b) use the learned models to detect and adapt to illumination changes.

Table 1: Planning and localization accuracies in challenging configurations. Planned motion sequence always succeeds in learning colors. Localization comparable to hand-labeled color map.

<table>
<thead>
<tr>
<th>Config</th>
<th>Plan (%)</th>
<th>Localized X (cm)</th>
<th>Localized Y (cm)</th>
<th>Localized ( \theta ) (deg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learned</td>
<td>100</td>
<td>9.6 ± 3.7</td>
<td>11.1 ± 4.8</td>
<td>9 ± 7.7</td>
</tr>
<tr>
<td>Hand-labeled</td>
<td>—</td>
<td>6.9 ± 4.1</td>
<td>9.2 ± 5.3</td>
<td>7.1 ± 5.9</td>
</tr>
</tbody>
</table>

The localization accuracy is used as the performance measure. Given the colors needed for localization (pink, yellow, blue, white, green), the depth of the search is limited to three for ease of analysis (and without loss of generality)—the ground colors (green, white) are learned by scanning in place. The field is discretized into \( 6 \times 9 \times 12 \) cells, that is, divisions of \( 600 \text{ mm, 600 mm, and 30°} \) along \( x, y, \) and \( \theta \). The back-propagation network is learned using the MATLAB Neural Network toolbox—the initial training of MEM and SFM takes \( \approx 1 \) hour of autonomous robot effort.

Different object configurations were created by placing six target objects at different positions along the boundary of the field that are known to the robot. The planning capability was evaluated for 7 challenging object configurations, each with 15 different robot starting poses. In addition, the localization errors were measured as the robot moved through a sequence of poses (15 trials of 10 poses)—ground truth was obtained with a tape measure and a protractor. The results are summarized in Table 1. The robot is able to generate a valid plan over all the trials, and the localization accuracy is comparable to that obtained from a hand-labeled color map. Figure 5 shows some planning results—the starting position is denoted by number “0” while the direction of the arrows show the orientation. The robot smoothly trades off the ability to learn better models for color distributions based on a larger object, against the associated motion-based localization errors.

In addition to the “best” motion-plan, several of the top sequences lead to successful color learning. If the robot is unable to learn all colors during plan execution, it creates a new plan based on current knowledge. Over a set of 20 images, the average segmentation accuracy of the learned and hand-labeled color map is 94.9 ± 3.9 and 96.7 ± 4.3 respectively (no difference at 95% significance). Ground truth is provided by a human observer. The motion planning is particularly useful where object configurations change less frequently than illumination. The entire color learning process takes \( \approx 6 \) minutes of robot effort instead of hours of human effort.

Next, the ability to detect and adapt to illumination changes was evaluated—here, using \( \text{Adapt}_X \) implies the use of \( \text{Detect}_X \) as well. First, the robot used \( \text{Adapt}_{\text{major}} \) as the illumination was slowly changed (over 20 seconds) between two conditions that would not be detected as being different by \( \text{Detect}_{\text{major}} \). The robot stood in place and panned its head, measuring the distance and angle to an object over the 20 seconds period. Table 2 summarizes the measurement errors averaged over four different objects and three different illuminations with \( \approx 15 \) trials under each
situation. Ground truth values were obtained with a tape measure and protractor. The results show that Adaptminor leads to segmentation accuracy (95 ± 4.3) and hence localization errors (≈10 cm, 12 cm, 10°) similar to those under constant illumination. In the absence of Adaptminor the measurement errors are significant.

Table 3: Time taken to find-and-walk-to-object.

<table>
<thead>
<tr>
<th>Illum + Alg</th>
<th>Time (sec)</th>
<th>Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant + NoAdapt</td>
<td>6.18 ± 0.24</td>
<td>0</td>
</tr>
<tr>
<td>Slow + Adaptmaj</td>
<td>31.73 ± 13.88</td>
<td>9</td>
</tr>
<tr>
<td>Slow + Adaptmaj,minor</td>
<td>6.24 ± 0.31</td>
<td>0</td>
</tr>
<tr>
<td>Sudden + Adaptmin</td>
<td>45.11 ± 11.13</td>
<td>13</td>
</tr>
<tr>
<td>Sudden + Adaptmaj,minor</td>
<td>9.72 ± 0.51</td>
<td>0</td>
</tr>
<tr>
<td>Sudden + Slow + Adaptmaj,minor</td>
<td>10.32 ± 0.83</td>
<td>0</td>
</tr>
</tbody>
</table>

Next, in order to show that both Adaptminor and Adaptmajor are essential, the time taken by the robot to find-and-walk-to-object is measured. The robot starts out near the center of the field with the object placed near the penalty box of the opponent’s goal. Table 3 summarizes the results averaged over different illuminations, with 15 trials under each illumination. With no change in illumination, the robot can find-and-walk-to-object in 6.18 ± 0.24 seconds. When the illumination changes slowly, using just Adaptmajor does not help—large variance in second row. Including Adaptminor provides good performance (6.24 ± 0.31 seconds). Similarly, when the illumination is changed suddenly, using just Adaptminor does not help—the robot totally fails to perform the task most of the time, resulting in a large number of failures (fourth row, third column). With both Adaptmajor and Adaptminor the robot can perform the task, the additional time being used to confirm that a change in illumination did occur (9.72 ± 0.51 seconds). In these experiments, major illumination changes result in illuminations for which the robot has already learned models—Adaptmajor implies a transition to the suitable model. Finally, the illumination is changed significantly, held constant for 3 seconds, and then changed slowly over the next 5 seconds. The robot is able to find-and-walk-to-object in 10.32 ± 0.83 seconds if and only if Adaptmajor and Adaptminor are used. The results show that the proposed algorithm enables the operation over a range of illuminations—different intensities (≈400Lux to ≈1600Lux) and color temperatures (2300 K–4000 K) were evaluated. Additional results (videos and images) are available online: http://www.cs.utexas.edu/AustinVilla/?p=research/autoplan_illum.

In the recent research, we have used the algorithms described above to learn models of other sensory features, and to robustly fuse information obtained from different sensory inputs on multiple robot platforms [78].

4.2. Visual Processing Management. The human–robot interaction domain described in Section 2.2.2 involves a robot equipped with multiple sensors whose inputs are processed by several algorithms with varying levels of uncertainty. Since the focus on this paper is on the visual input, we present an algorithm that autonomously tailors its visual processing to the task at hand.

Consider the example of an input image from the tabletop scenario (Section 2.2.2) that is preprocessed to yield regions of interest (ROI), that is, rectangular image regions that are different from a previously trained model of the
background—Figure 4(a) shows examples of ROIs. Consider
the query: “which objects in the scene are blue?” Without
loss of generality and for ease of analysis, assume that the
robot has the following set of visual operators at its disposal:
a color operator that classifies the dominant color of the ROI
it is applied on, a shape operator that classifies the dominant
shape within the ROI, a sift operator that uses the SIFT
features [79] to detect the presence of one of the previously
trained object models. The color operator characterizes
ROIs based on color-space histograms, while the shape
operator characterizes the dominant contour within the ROI
using invariant moments. The sift operator characterizes
target objects with local image gradients that are robust
to scale, orientation, and viewpoint changes. We use the
following terms interchangeably: visual processing actions,
visual actions, and visual operators. Given these operators,
the task is to plan a sequence of operators that can answer
user queries with high confidence.

We pose the visual processing management task as an
instance of probabilistic sequential decision making,
and specifically as a Partially Observable Markov Deci-
sion Process (POMDP) [80]. The POMDP formulation
captures the partial observability and non-determinism
that characterize visual processing on robots (Section 1).
The robot maintains a probability distribution over the
true underlying state, called the belief state. Each action
considers the true underlying state to be composed of
the class labels (e.g., red(R), green(G), blue(B) for color;
circle(C), triangle(T), square(S) for shape; picture, mug, box
for sift), a label to denote the absence of any valid object—
empty (φ), and a label to denote the presence of multiple
classes (M). The belief state maintenance also requires an
observation function that provides a probability distribution
over the set of possible outcomes of each action. The
set of action outcomes consists of the class labels, the
label empty (φ) which implies that the match probability
corresponding to the class labels is very low, and unknown
(U) which implies that multiple classes are equally likely
and the ROI may therefore contain multiple objects. U
is an observation, whereas M is part of the underlying
state: they are not the same since they are not perfectly
correlated.

Since operators only update belief states, we include
“special actions” that cause a transition to a terminal state
where no further actions are applied, that is, those query-
specific actions terminate processing to answer the query.
The answer could report or “say” (not to be confused with
language-based communication) which underlying state is
most likely to be the true state, or it could simply state the
presence or absence of the target object. In the description
below, without loss of generality and for ease of explanation,
we only consider two operators: color and shape, each of
which provides three class labels. The operators are denoted
with the subscripts c and s, respectively. The approach
generalizes to sift, other vision algorithms, and more out-
comes. True states and observations are distinguished by
the superscripts a and o, respectively. The POMDP for
a single ROI in the image can then be defined as the tuple
(Δ, A, T, Z, Θ, R).

(i) Δ : Δc × Δs ∪ term, the set of states, is a Carte-
sian product of the variables describing different
aspects of the underlying state. It also includes a
terminal state (term). Δc : {φc, Rc, Gc, Bc, Mc}, Δs : 
{φs, Cs, Ts, Ms}.

(ii) A : {color, shape, Asp} is the set of actions. The
first two entries are the visual operators. The rest are
special actions that represent responses to the queries,
describing the presence/absence of the target: Asp = 
{sFound, sNotFound}, or specific query responses:
Asp = {sRed, sGreen, sBlue}, that is, actions such as
“say blue”. All the special actions lead to term.

(iii) T : Δ × A × Δ → [0, 1] represents the state transition
function. For operators such as color and shape that
do not change the underlying state, it is the identity
matrix. For special actions it represents a transition to
term. For actions that change the state, the transition
function can be defined suitably [81].

(iv) Z : {φo, Ro, Go, Bo, Mo, φo, Co, To, Mo, Uo} is the set of
observations, a union of the observations for each visual
action under consideration, that is, Z = Zc ∪ Zs.

(v) Θ : Δ × A × Δ → [0, 1] is the observation function,
a matrix of size |Δ| × |Z| for each action. For each
visual action, it is learned offline by the robot, and it
is a uniform distribution for the special actions.

(vi) R : Δ × A → R specifies the reward, that is, the value
of taking a particular action in a particular state. It is
hence a mapping from the state-action space to real
numbers—a negative reward represents a cost. In our
case
∀s ∈ Δ, R(s, shape) = 1.25 · fc(ROI − size),
R(s, color) = −2.5 · fc(ROI − size),
R(s, special actions) = ±100 · α.

(11)
The cost for visual actions depends on the relative
computational complexity of the operator and the
size of the ROI. For instance, the color operator is
twice as costly as shape, and this is used to assign
a cost factor such that the least expensive operator
has a relative cost value close to 1. The dependence
on ROI size is captured using a polynomial function
(14)—the degree and coefficients of the function are
computed experimentally as described later in this
section. For special actions, a large positive (negative)
reward is assigned for making a right (wrong) decision for a given query. For instance, for “what
is the color of the ROI?”: R(RcTc, sRed) = 100 · α
and R(BoTo, sGreen) = −100 · α, while for “is there a
red object in the scene?”: R(RcTc, sNotFound) = 100 · α,
and R(BoTo, sNotFound) = −100 · α. The variable α
trades-off computational costs against reliability. For
instance, when α is large the special action is taken
after executing a larger number of actions, resulting
in higher reliability.
Given the belief state, that is, the probability distribution over the underlying state at time $t$: $b_t$, the belief update proceeds as

$$b_{t+1}(s') = \frac{\Theta(s', a_{t+1}) \sum_{a_i} \mathcal{T}(s, a_t, s') \cdot b_t(s)}{\mathcal{P}(a_{t+1} \mid a_t, b_t)}, \quad (12)$$

where $\Theta(s', a_{t+1}) = \mathcal{P}(a_{t+1} \mid s_{t+1} = s', a_t)$, $b_t(s) = \mathcal{P}(s_t = s)$, $\mathcal{P}(a_{t+1} \mid a_t, b_t) = \sum_{a'} \mathcal{P}(a_{t+1} \mid s', a_t) \cdot \sum_{s'} \mathcal{P}(s' \mid a_t, s) b_t(s)$ is the normalizer and $\mathcal{T}(s, a_t, s') = \mathcal{P}(s_{t+1} = s' \mid a_t, s_t = s)$. The planning task for a single ROI involves solving this POMDP to find a policy of the form

$$\pi^* : (b) \rightarrow a, \quad (13)$$

that is, a mapping from belief states to actions 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 receiving an observation. In order to ensure that the observations are conditionally independent of each other given different images of the same scene, we take a new image of the scene if an action is to be repeated on the same ROI. This independence assumption is essential for the belief update described in (12), and though the images are not strictly independent the assumption works well in practice.

For a single ROI with $m$ features (e.g., color, shape) each with $n$ values (e.g., $a^c_i$, $a^s_i$, $a^g_i$, $a^b_i$, and $a^m_i$), the POMDP has a state space of size $n^m + 1$. Actual scenes will contain several objects and hence several ROIs—for $k$ ROIs we have $n^mk + 1$ states, that is, the state space grows exponentially. In addition, the (worst case) time complexity of POMDPs is exponential in the state space dimensions. Therefore, POMDP formulations of all but the very simple problems soon become intractable, even with state-of-the-art approximate solvers [82].

We ameliorate part of the inherent intractability by introducing 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 is to be executed. This decomposes the overall problem into one POMDP with state space $2^k + 1$, and $k$ POMDPs with state space $n^m + 1$. Essentially, we have achieved a functional decomposition by separating the problem of what information to process (i.e., which ROI to focus on) from how to process it (i.e., which operators to use). Without loss of generality, we assume that there are two ROIs in the image and define the HL-POMDP as $(\mathcal{S}^H, \mathcal{A}^H, \mathcal{T}^H, \mathcal{Z}^H, \Theta^H, \mathcal{R}^H)$.

(i) $\mathcal{S}^H = \{R_1 \land \neg R_1, \neg R_1 \land R_2, \neg R_1 \land \neg R_2, R_1 \land R_2\} \cup \text{term}^H$ is the set of states. It represents the presence or absence of an object satisfying the query in one or more of the ROIs. It also includes a terminal state ($\text{term}^H$).

(ii) $\mathcal{A}^H = \{u_1, u_2, \mathcal{A}^H_{ot}\}$ are the actions. The sensing actions ($u_i$) denote the choice of executing one of the LL ROIs’ policy trees. The special actions ($\mathcal{A}^H_{ot}$) are query-specific, and defined in a manner similar to that for the LL-POMDP. All the special actions lead to $\text{term}^H$.

(iii) $\mathcal{T}^H$ is the state transition function, which leads to $\text{term}^H$ for special actions and is an identity matrix for other actions.

(iv) $\mathcal{Z}^H = \{FR_1, \neg FR_1, FR_2, \neg FR_2\}$ is the set of observations. It represents the observation of finding or not-finding the target object when each ROI’s policy is executed.

(v) $\Theta^H : \mathcal{S}^H \times \mathcal{A}^H \times \mathcal{Z}^H \rightarrow [0, 1]$, the observation function of size $|\mathcal{S}^H| \times |\mathcal{Z}^H|$, is a uniform matrix for special actions. For sensing actions, it is obtained from the policy trees for the LL-POMDPs.

(vi) $\mathcal{R}^H$ is the reward specification. For each sensing action, it is the “cost” of running the visual policy solution of the corresponding LL-POMDP. For a special action, it is a large positive (negative) value if it predicts the true underlying state correctly (incorrectly): $\mathcal{R}(R_1 \land R_2, sR_1 \land R_2) = 100$, while $\mathcal{R}(R_1 \land \neg R_2, sR_1 \land R_2) = -100$.

A key challenge in such hierarchical formulations is the belief propagation between the levels in the hierarchy. In order to automate this belief propagation, the HL reward and observation functions in our hierarchy are based on the policy trees of the corresponding LL-POMDPs. More specifically, the HL observation function and reward specification are computed by traversing the corresponding LL policy trees, while propagating an initial belief and using LL observation functions that are modified based on the target query. However, these changes to the LL belief states and observation functions are used only for building the HL-POMDP model. Normal belief updates in the LL-POMDPs use an unmodified observation function and an appropriate initial belief, that is, for instance, uniform if nothing is known about the contents of the corresponding ROI. Complete details on the belief propagation between the LL and the HL can be found in [13].

The overall planning and execution cycle is as follows. Based on the target query, the available visual actions and the number of ROIs, the LL-POMDPs are created and solved. The policy trees of the LL-POMDPs are parsed to automatically generate the required components (e.g., observation functions, rewards) of the HL-POMDP. The HL-POMDP is then solved to obtain the HL policy. During execution, invoking the HL-Policy results in the selection and analysis of a specific ROI. The ROI is analyzed until a terminal action is reached in the HL. The control then returns to the HL, where the beliefs are updated based on the LL response and a new action is chosen, that is, a ROI is selected for further analysis. The process continues until a terminal action is executed in the HL and the input query is answered.
In practical scenes, objects may overlap due to occlusions or a change in viewpoint, resulting in multiple objects being enveloped in a single ROI. Processing such scenes would require visual operators that split ROIs into subregions based on one or more of its properties (e.g., color, shape, local gradients). Planning with such actions that change the perceived state of the system is a significant challenge in POMDP formulations. However, such actions can be included in our hierarchy by defining suitable transition and observation functions. In the case of the region-splitting actions, the only difference during execution would be that a state change in the LL could create new ROIs. In addition to creating and solving POMDP models for the new ROIs in the LL, a new POMDP will have to be created and solved in the HL. The execution cycle would otherwise remain unchanged. Though we do not provide more information here on the incorporation of actions that analyze images with overlapping objects, complete details can be found in [81, 83]. In addition, the quantitative results described below were obtained by including such actions in the experimental analysis.

In summary, we propose a two-level hierarchy in the (image) state and action space. In the LL, each ROI is assigned a POMDP and analyzed using the visual operators, while the HL-POMDP maintains the belief over the entire image and chooses (at each step) the ROI best-suited for further processing, thereby answering the input query. The process of creating and solving the POMDPs proceeds automatically because of the elegant belief propagation between the LL and HL. As a result, the proposed approach can be used to address a range of queries in the test domain.

4.2.1. Experimental Results. The experimental setup is as follows. The camera mounted on a robot captures images of a tabletop scene. Any change from the learned model of the background is identified as a salient region, and all such regions of interest (ROI) are extracted. The system has a sophisticated saliency operator for complex scenes [84], but the background subtraction suffices for the tabletop scenario—it is also computationally efficient. In an initial training phase, objects of known properties are put on the table, and the robot repeatedly applies the available operators on these objects. Statistics are collected regarding the operator outcomes and the run-times of the individual operators on specific ROIs. These statistics are used to estimate the observation functions and the rewards/costs of the visual operators. For instance, we had defined the dependence of the visual operator costs on ROI size as a polynomial

\[ f(r) = a_0 + \sum_{k=1}^{N} a_k \cdot r^k, \]  

where \( r \) is the ROI-size (in pixels). The degree and coefficients of the polynomial are estimated from the collected statistics. In the experiments below, all POMDPs are created in the format of the ZMDP package [85]. The POMDPs are solved using a state of the art point-based solver [82] in the package.

We first describe the execution for the query: “where are the blue circles?” on the image shown in Figure 6(a). Since no prior information is available about either of the two ROIs the HL-POMDP first chooses to analyze the ROI \( R_1 \) because its smaller size results in lower action costs: action \( u_1 \) in Figure 6(b). The corresponding LL-POMDP runs the color operator on the ROI. Even though it is more costly, the color operator’s observation function indicates a higher likelihood of success in comparison to shape, and it is hence applied first. The outcome of applying an operator is one of the possible observations (e.g., \( R_1^c \), \( G_2^c \), \( B_3^c \), \( U_c^c \), \( \phi_c^e \) for color)—in this case the answer is \( R_1^c \), that is, \( \text{red} \). The observation is used to update the belief state. In this case, the outcome decreases the likelihood of finding a blue circle in \( R_1 \). The reward specification (\( \alpha = 0.2 \) in (11)) ensures a trade-off between computation and reliability, and there is no further investigation of this ROI (e.g., with a shape operator). The best action chosen in the next step of the LL policy for \( R_1 \) is hence a terminal action: \( \text{sNotFound} \). The HL-POMDP receives the observation that the target is not found in \( R_1 \), leading to a belief update and a subsequent action selection: action \( u_1 \) in Figure 6(c). Then \( R_1 \)'s LL-POMDP policy tree is invoked, causing the color and shape operators to be applied in turn on the ROI. The higher noise in the shape operator causes it to be applied twice (on two independent images) before the uncertainty is reduced sufficiently. Then a terminal action (\( \text{sFound} \)) is chosen—the increased reliability therefore comes at the cost of execution overhead. The response from the LL-POMDP of \( R_2 \) updates the HL belief, resulting in the selection of a terminal action in the HL-POMDP: \( (\neg R_1 \land R_2) \), that is, a blue circle exists in \( R_2 \) and not \( R_1 \)—Figure 6(d).

One could argue that it would be better to choose a new action in the HL at each time-step, instead of waiting for the LL-POMDP to terminate. However, the proposed approach provides the key benefit of automatic belief translation from the LL to the HL. In addition, it stops early if negative evidence is found for the target object. Finding positive evidence only increases the posterior probability of the ROI being explored—even if the HL-POMDP were to choose the next action, it would choose to process the same ROI again.

One advantage of the POMDP-based approach is that it is easy to incorporate prior knowledge in the decision-making. Consider the same scene in Figure 6(a) and the query: “where is the blue circle?”; that is, the location of the single blue circle in the image is to be determined. If it is known that the blue circle is more likely to exist in \( R_1 \), the initial beliefs of the ROI could be modified. As a result, the cost of execution of \( R_2 \)'s policy would be lower (in the HL-POMDP), and \( R_2 \) would be chosen to be analyzed first leading to a faster response.

In order to evaluate the proposed approach quantitatively, the hierarchical POMDP planner (HiPPo) is compared with a modern planner that handles the non-determinism qualitatively: Continual Planning (CP) [53]. As discussed in Section 3.2, CP is a fast planner that has been applied to human-robot interaction scenarios. The planning methods were also compared against the naive approach of applying all the available operators on the ROIs. The results obtained over a set of \( \approx 15 \) different queries, with \( \approx 10 \) trials for each
such query, are summarized in Figures 7(a) and 7(b). The naive approach is denoted by “no planning” in Figure 7(b).

First, Figure 7(a) compares HiPPo against the standard POMDP approach that plans in the joint space of all the ROIs. The nonhierarchical approach soon becomes intractable, even when as few as three operators are used to analyze three or more ROIs. HiPPo, on the other hand, is reasonably efficient even as the number of ROIs in the scene increases.

The LL-POMDPs for two ROIs typically differ only in terms of their action costs that are a function of ROI sizes. Hence, the policies computed over discretized ROI sizes were cached and reused for ROIs of similar size. This approximation makes HiPPo’s planning time comparable to that of CP, and the value estimation error introduced by this approximation can be measured and used to trade-off accuracy against efficiency [81]. As observed in Figure 7(b), the total (planning + execution) time for HiPPo is only slightly larger than that of CP—HiPPo has a larger execution time because some operators are executed more than once in order to reduce the uncertainty. In addition, both planners (HiPPo and CP) are significantly faster than the naive approach.

Finally, we compared the three approaches in terms of reliability, that is, their ability to provide correct answers to queries. HiPPo provides a reliability of 90.75% that is significantly better than the reliability of CP (76.67%) or the naive approach (76.67%). CP cannot perform any better than the naive approach because it does not account for the uncertainty in operator outcomes. HiPPo, on the other hand, inherently exploits the learned models of operator uncertainties and accumulates belief to provide reliable performance.

The key contribution is the hierarchical decomposition that can be modeled automatically to address a range of queries. It is easy to incorporate other operators, even those that change the state of the system. HiPPo is therefore an efficient and reliable approach towards visual processing management. Furthermore, the lessons learned in the tabletop scenario can be used in other applications.

5. Conclusions and Future Work

A central goal of robotics and AI is to enable a team of robots to operate autonomously in the real world and collaborate with humans over an extended period of time. In this paper...
we have described algorithms that address two key challenges to widespread deployment of mobile robots: autonomous learning and adaptation, and processing management. We have focused primarily on visual input from color cameras and summarized two key contributions: (a) a probabilistic framework where the robot autonomously learns models for color distributions and illumination, and detects and adapts to illumination changes; (b) a probabilistic sequential decision-making framework that enables the robot to autonomously tailor the visual information processing to the task under consideration.

Our bootstrap learning approach enables a robot to plan its actions in order to learn models for color distributions and illuminations. The lessons learned with this low-dimensional feature (i.e., color) can be extended to other visual features (e.g., texture, gradients) and nonvisual input (e.g., range information). Currently, the approach for planned learning requires information about the structure of the environment, that is, a map of the world. However, existing approaches in robotics and computer vision can be incorporated in this system to learn most of this structure autonomously [86, 87].

The approach described in this paper enables a mobile robot to use the learned models to detect and adapt to illumination changes. One future direction of research is to incorporate a joint model of color and illuminations. Existing research in the field of computer (and human) vision can be used to identify the parameters of this model, and the robot can estimate the values of the parameters based on data collected in its operating environment. The approach can then be extended to other visual features as well. The long-term goal would be to fully automate the learning of environmental models, and the adaptation to environmental changes.

A robot equipped with multiple sensors and multiple algorithms to process the sensory input needs a scheme to tailor the processing to the task under consideration. In this paper, we have focused on such processing management of visual input. One future direction of research is to include other operators and process more complex scenes. This increase in complexity may require a range of hierarchies in state and action spaces [66]. We are also interested in high-level scene processing, which could be defined as an additional level in the hierarchy above the existing levels. A particular region in space could be chosen for analysis with the objective of maximizing the information gain, and the existing hierarchy could then be used to process images of the chosen region in space. The key challenge would once again be the automatic belief propagation between the levels in the hierarchy.

As seen in Section 4.2, one key challenge with POMDP formulations of practical problems is the efficiency. Our hierarchical decomposition helps address part of the observed intractability. However, as the focus shifts to more complex scenarios, it may be essential to decouple the parts of the scenario that can be analyzed using nonprobabilistic methods. The POMDP-based analysis of the more uncertain components of the system would then be tractable.

Overall, we have summarized algorithms that address key challenges to the widespread deployment of mobile robots in the real world. We have shown that the robots can autonomously learn, adapt, and plan their sensory information processing. The long-term goal is to enable
robots to use a combination of learning and planning to respond autonomously and efficiently to a range of tasks, thereby collaborating with humans in a wide range of critical applications.

Acknowledgments

The author thanks collaborators from the University of Texas at Austin (Peter Stone) and University of Birmingham (UK) (Jeremy Wyatt, Richard Dearden, and Aaron Sloman). This work was supported in part by the ONR award N00014-09-1-0658.

References


