Integrating Task Level Planning with Stochastic Control

UMass Technical Report UM-CS-2014-005

Shiraj Sen, Roderic Grupen
Department of Computer Science
University of Massachusetts Amherst
{shiraj, grupen}@cs.umass.edu

Abstract

Real-world tasks require planners to search for solutions in a partially observable state space. Planning in partially observable domains is computationally expensive—it is impossible to plan for all possible contingencies in advance. Furthermore, real world scenarios introduce a huge variety of context-dependent variation in the dynamics of belief in response to new information. Our approach investigates the use of models of belief dynamics designed to exploit latent structure in the environment in the form of "objects" and we evaluate the possibility of accumulating thousands of such models in a practical, high performance planning system. We evaluate the efficacy of our approach in various tasks using a real robot.

I. Introduction

An intelligent agent must reason about its own skills, and about the relationship between these skills and goals under run-time conditions. This requires the agent to represent knowledge about its interactions with the world in a manner that supports reasoning. Since the early 1970s, the AI and robotics community has been concerned with the design of efficient representations that support modeling and reasoning. However, most of these representations tend to tackle only one part of the problem—making either the modeling or reasoning problem easier.

Manipulation planning requires determining a goal configuration for possibly several objects, and generating a sequence of manipulation actions that result in the desired configuration [1, 24]. The planning community either uses geometric planners that plan in the configuration space of the robot and the world, or a propositional logic based planner that expresses the state of the world as logical assertions engineered by humans [14]. Lozano-Pérez, Mason, and Taylor presented the *preimage planning* framework [15] to address manipulation planning problems in configuration space with bounded uncertainty. The most popular method within the preimage planning framework involves performing a backward search from the goal until it reaches the starting state. Although this sounds simple enough, the set of possible motion commands is infinite. This has caused research in this area to shift from exact, complete algorithms to sampling-based algorithms, such as Rapidly-exploring Random Trees (RRT) [13], that can rapidly find a feasible solution at the expense of completeness. It is however a challenge to represent information about the world and task exclusively in configuration space.

Logic based representations have traditionally been used in AI to represent domain knowledge. One of the most well known applications of this representation was STRIPS [5], which represented the world using a set of well ordered formulas of the first-order predicate calculus. Fikes and Nilsson showed how such a representation can be used to generate plans—sequences of abstract high level actions—for the robot SHAKEY [18]. Planning in hybrid spaces, combining discrete mode switching with continuous geometry has also been used to sequence robot motions involving different contact states or dynamics [7]. Cambon *et al.* [3] showed how linking symbolic description to its geometric counterpart can lead to an

integrated planning process that is able to deal with intricate symbolic and geometric constraints. Plaku and Hager [20] extended this approach to handle robots with differential constraints and provide a utility-driven search strategy. A robot that uses logical reasoning can sound highly compelling, since all the agent will then require is a representation of the knowledge expressed in logic and a theorem prover as part of the problem-solver. However as Wooldridge [26] pointed out, to build an autonomous robot with such capabilities, robots need to possess the capability to translate the real world into an accurate and adequate symbolic representation, and then reason based on partial information.

This paper addresses these dual problems of modeling and reasoning by employing a state representation grounded in a robot's own actions. Our description of state is domain general, as it is directly computed from the status of the actions that the robot has access to, as opposed to being hand built for a specific task. The relationship between state and action is captured using probabilistic data structures that model objects in the environment. We present a planner that exploits the uniform description of state and the probabilistic models to plan efficiently in partially observable environments.

Hierarchical approaches to planning have been proposed to speed up the search for plans. Since the work of Sacerdoti [23] on the ABSTRIPS method that generated a plan in a hierarchy of abstraction spaces, many researchers have suggested a hierarchical approach to interleaving planning and execution [19]. Wolfe *et al.* [25] provided a task and motion planner based on hierarchical transition networks (HTNs) [17]. Platt [21] showed that one can compute reasonable policies in the belief space based on a local linearization of the belief space dynamics. The controller then selects actions based not only on the current most-likely state of the robot, but on the information available to the robot. This approach is promising, however in general belief space dynamics need not be linear, and hence resulting policies are applicable only in the vicinity of a locally stabilized region. Kaelbling and Lozano-Pérez [9, 10] proposed a hierarchical planner that sacrifices optimality quite aggressively, for efficiency, by having a planner that makes choices and commits to them in a top-down fashion in an attempt to limit the length of plans that need to be constructed, and thereby exponentially decreasing the amount of search required. Our approach is similar to the above, in which the robot selects the best possible action based on the current belief. However, the number of models over which the planner searches decreases with every new observation. This is achieved by pruning the model space of hypotheses that don't match the observation. This leads to faster generation of plans with increased observations.

II. State Representation

Our computational representation of knowledge makes use of closed-loop controllers and their interaction dynamics to represent robot specific knowledge structures [8]. Primitive controllers are constructed from combinations of navigation functions ($\phi \in \Phi$), sensory ($\sigma \subseteq \Sigma$), and motor resources ($\tau \subseteq \mathcal{T}$) defined by three finite sets:

- Φ is a set of navigation functions whose gradients lead asymptotically to fixed points, corresponding to the goal location [12]. A navigation function $\phi : \sigma \rightarrow [0, \infty]$ maps a feedback signal to a real number such that
  1) $\phi(\sigma) = 0$ for all $\sigma \in \Sigma_G$, the goal states.
  2) $\phi(\sigma) = \infty$ if and only if no point in Σ_G is reachable from σ.
  3) For every reachable state, $\sigma \in \Sigma \backslash \Sigma_G$, there exists a gradient towards the goal.

  A navigation function models the effector of a robot as a particle moving under the influence of a potential field that is proportional to the distance of the effector from its goal.
- Σ is a set of feedback entities that are computed by applying operators to sensory signals.
- $\mathcal{T}$ is a set of motor units that actuate independent degrees of freedom in the robot. Higher-level controllers submit patterns of real-valued references $\mathbf{u}_\tau$ to a set of motor units, $\tau \subseteq \mathcal{T}$.

Definition 1 (Controller): Let $c(\phi, \sigma, \tau)$, where $\phi \in \Phi$, $\sigma \subseteq \Sigma$, and $\tau \subseteq \mathcal{T}$ define a closed-loop controller.

Controllers achieve their objective by following gradients in the scalar navigation function $\phi(\sigma)$ with respect to changes in the value of the motor variables $\mathbf{u}_\tau$, as captured in the error Jacobian

$$J = \frac{\delta \phi(\sigma)}{\delta \mathbf{u}_\tau}. \tag{1}$$

Reference inputs to lower-level motor units are computed by

$$\Delta \mathbf{u}_\tau = \kappa J^{\#} \Delta \phi(\sigma), \tag{2}$$

where $J^{\#}$ is the pseudoinverse of J [16], $\Delta\phi(\sigma) = \phi(\sigma_{ref}) - \phi(\sigma_{act})$, the difference between reference and actual potential, and κ is a (small) positive gain. In our case, the reference potential σ_{ref} is the potential at the goal state.

The interaction between the robot and its environment is modeled as a dynamical system. Modeling the interactions as a dynamical system allows a robot to evaluate the status of its action as the state of a time varying control system. Baum-Welch showed that the time history (trajectory) of a dynamical system provides a highly informative basis for uncovering the parameters of the underlying stochastic process (Baum-Welch algorithm for HMMs [2]). Coelho [4] showed that the dynamics $(\phi, \dot{\phi})$ created when a controller interacts with the environment can be used to predict the state of the underlying environment. Our representation of state is derived directly from the dynamics of the controllers.

Definition 2 (Controller state): Let γ^t define the state of a controller $c(\phi, \sigma, \tau)$ at time t, where $\gamma \in \{-, 0, 1\}$ such that:

$$\gamma^t(c) = \begin{cases} - & : \quad \phi \text{ has undefined reference} \\ 0 & : \quad |\dot{\phi}| > \epsilon \\ 1 & : \quad |\dot{\phi}| \leq \epsilon \end{cases} \tag{3}$$

where ϵ is a small positive constant.

In this state representation, '$-$' indicates that the controller is currently not executable since the reference input signal σ required to compute the gradient, is not present in the feedback, '0' indicates that the controller is making progress but hasn't yet reached its target fixed point, and '1' denotes convergence/quiescence evaluated relative to a small positive threshold, ϵ. The undefined reference state '$-$' is an absorbing state since the navigation function has no gradient in this state and hence can't make progress towards the goal.

A collection of n distinct controllers forms a discrete state space $\mathbf{s}^t = [\gamma_1^t \cdots \gamma_n^t] \in \mathcal{S}$ at time t. Defining the state directly in terms of the dynamics of the actions that a robot can execute provides a representation that is obtained directly from a robot's action repertoire, as opposed to being defined by a human. Furthermore, the complexity of the representation is related to the cardinality of the action set.

A. Control Programs

Sets of control actions are sequenced by a finite state automaton into temporally extended control programs. These control programs are either hand built or learned using reinforcement learning [6]. In this paper, we make use of a special class of control programs called SEARCHTRACK programs. All programs belonging to this class have the property that they sequence actions that *search* for reference feedback in the environment, and then *track* it. A SEARCHTRACK program p proceeds by executing a track action

that preserves a reference value in the feedback signal σ e.g., tracking the position of a color feature in the "fovea" by moving a robot. However, in the absence of a feedback signal, the program searches by reorienting the robot's sensor geometry to increase the probability of success of the tracking action. This is achieved by executing an action that samples its goal configuration from a distribution over effector variables, $\Pr(\mathbf{u}_\tau|\gamma(p) = 1)$, where the program had previously converged. Initially the distribution $\Pr(\mathbf{u}_\tau|\gamma(p) = 1)$ is uniform; however, as it is updated over the course of many episodes, this distribution reflects the long term statistics of the run-time environment.

The state of a SEARCHTRACK program is evaluated in a similar fashion as that of controllers. However, unlike primitive controllers, the state of a control program is related to the dynamics of the TRACK action.

B. Object Models

SEARCHTRACK programs model action specific information. However, in order to perform reasoning, a model needs to predict action dynamics *in situ*, and utilize state feedback to update its posterior distributions over the model space. In this work, we model the most basic environmental structure—rigid body objects—to capture this relationship between state and action.

An object *affords* a set of control programs. Objects are modeled as (temporal) distributions over the status of multiple control programs. Figure 1 shows a graphical model that encodes the probabilistic dependencies between variables comprising the object model. An object $o \in O$ induces a distribution over a set of M mutually exclusive aspects. An aspect $x \in X$ is a latent variable that models patterns over the status of several control programs. It implicitly captures the kinematic constraints and the sensor geometry for a constellation of control programs—sets of programs that can or cannot track stimuli simultaneously in the environment. An aspect can include feedback from a set of visually-referenced actions, together with feedback from a set of force-referenced actions. The features comprising a visual aspect are mutually consistent with line of sight constraints (e.g., set of visual features on an object that can be tracked simultaneously). Elements of the haptic aspect are mutually consistent with kinematic reachability constraints.

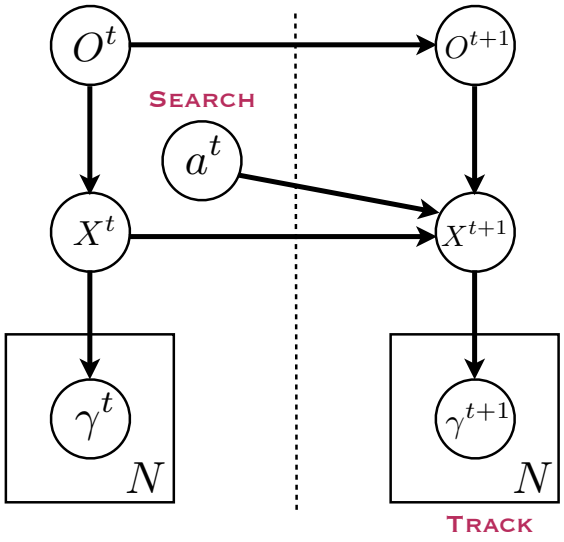


Fig. 1. A Bayesian network model representing objects O as a temporal distribution over aspects X. An aspect induces expectations over the state of N tracking programs (γ_j) as shown by the plate model. The two time slices in the model show the logical dependencies between aspects and an orienting search action a. O, X and a are modeled as multinomial random variables. γ_j is modeled as a Bernoulli random variable.

Each aspect $x \in X$ induces a distribution over the state of N-SEARCHTRACK programs. There can be multiple instances of each aspect within an object. Each control program is represented by a Bernoulli random variable γ_j describing the state of each associated action. ($\gamma_j = 1$, if the action converges, $\gamma =$'-', if the gradient is undefined).

The dependencies between the aspects (x^t and x^{t+1}) over two time steps ($t, t+1$) and the SEARCH action being executed (a^t) is encoded by the two time slices of the Dynamic Bayesian Network (DBN) in Figure 1. This part of the model describes how executing an action on an object, that reorients the robot's sensor geometry with respect to the object, influences the future state of control programs. For example, a hammer's handle affords the action of grasping, however if the handle is out of reach, the robot might have to use a non-prehensile tactile controller that pulls the hammer closer before it can succeed in grasping it. In this case, "pulling" changes the aspect of the object in a manner that supports the goal of grasping. Modeling objects in the world in terms of the properties derived from controllable actions and the relationships between them allows an agent to use the same model for all tasks.

III. TASK-LEVEL PLANNING

The challenge of integrating task-level planners with stochastic control requires dealing with the partial observability of the state while building plans. Since the true state[1] s^{t+1} of the system cannot be observed, it must be inferred from observations z^{t+1} made after taking action a^t. The observation is related to the state by the likelihood function $\Pr(z^{t+1}|s^{t+1}, a^t)$, which is proportional to the probability that observation z^{t+1} is the result of taking an action a^t to reach a particular state s^{t+1}. The probability density function $\Pr(z^{t+1}|a^t)$ of the observation is obtained by marginalizing over all states

$$\Pr(z^{t+1}|a^t) = \sum_{s^{t+1}} \Pr(z^{t+1}|s^{t+1}, a^t) \Pr(s^{t+1}) \tag{4}$$

The important quantity in this formalism is the action a^t. Since the likelihood function $\Pr(z^{t+1}|s^{t+1}, a^t)$ is conditioned on the action, it is clear that actions influence observations. The goal is to update belief in the true state s^{t+1}, given the observation z^{t+1}.

Our planner greedily selects actions that reduce the uncertainty of the state estimate optimally with respect to the task. In information theory, mutual information (MI) defines how much uncertainty is reduced in a random variable (s^t) provided an observation (z^{t+1}) is made. Since the information flow depends on the action a^t, we need to define conditional MI:

$$I(s^t; z^{t+1}|a^t) = H(s^t) - H(s^t|z^{t+1}, a^t) \tag{5}$$

where H is the entropy associated with the random variable. The entropy is zero if the state is uniquely determined; it reaches its maximum if all states are equally likely.

The optimal next action $\hat{a}^t$, given a belief over states $\Pr(s^t)$ and observation model $\Pr(z^{t+1}|s^t, a^t)$ is

$$\hat{a}^t = \arg\max_{a^t} I(s^t; z^{t+1}|a^t) \tag{6}$$

Initially with no experience to draw on and before any observations are made, $\Pr(s^0)$ is uniform. However, as actions are executed by the planner to optimize mutual information, new observations are used to update the *a posteriori* probability of each state s^t,

$$\Pr(s^t|z, a) = \frac{\Pr(z|s^t, a) \Pr(s^t)}{\Pr(z|a)} \tag{7}$$

[1]The state, s^{t+1} described in this section should not be confused with state s described in Section II.

In the next time step, the planner uses the set of *a posteriori* probabilities as *a priori* probability for s^t. This allows the planner to utilize its current belief to act optimally.

All tasks in our framework can be posed as the manipulation of belief—condensing belief over objects (recognition), aspects (pose recovery), or distribution of trackable features (external stimuli). In the next few subsections, we describe in detail how to use the same planner for building plans for some of the common tasks.

A. Object Recognition

The task of object recognition is described as the process of selecting actions that maximally reduce the uncertainty over objects—condensing belief over objects given observations of the status of multiple control programs. The overall algorithm for recognizing objects is shown in Alg. 1. Lines 4 to 10 are repeated until the uncertainty over objects is completely removed. The algorithm proceeds by making observations and using them to compute a belief over aspects (Line 5). Since the aspect of an object is a latent variable that needs to be inferred from the observations, we make use of the maximum likelihood estimate over aspects along with the prior distribution over objects to compute the *a posteriori* distribution as given by Equation 7. Using the definitions of entropies and our object model, the conditional MI is given by

$$I(o^t; x^{t+1}|a^t) = \sum_{o^t} \sum_{x^{t+1}} \Pr(o^t) \Pr(x^{t+1}|o^t, a^t) \times \log \left(\frac{\Pr(x^{t+1}|o^t, a^t)}{\Pr(x^{t+1}|a^t)} \right) \tag{8}$$

where, the maximum likelihood estimate over aspects, x^t_{ML} is utilized to compute the observation likelihood, $\Pr(x^{t+1}|o^t, a^t) = \Pr(x^{t+1}|o^t, a^t, x^t_{ML})$. The optimal action to execute (Line 8) is then given by

$$\hat{a}^t = \arg\max_{a^t} I(o^t; x^{t+1}|a^t) \tag{9}$$

Algorithm 1 Object Recognition Plan

```
1:  procedure RECOGNIZE(o_prior)
2:      action ← NULL
3:      while entropy(o_prior)! = 0.0 do
4:          Make Observations
5:          S ← UpdateAspectBelief()
6:          x_ML ← MostLikelyAspect(S)
7:          o_post ← ObjectPosterior(o_prior, x_ML, S, action)
8:          action ← NextAction(o_post, x_ML)
9:          Execute(action, x_ML)
10:         o_prior ← o_post
11:     end while
12:     o_recog ← MostLikelyObject(o_prior)
13:     return o_recog
14: end procedure
```

B. Action Execution

Once a closed-loop action is configured, executing it involves sampling search goals that maximize the possibility of success. The algorithm is shown in Alg 2. The algorithm takes as input the action to be executed, an object model, and a set of optional goals (dependent on the action). The plan proceeds by estimating the object pose and using the computed pose along with the model to sample search goals for the robot (Line 4). If the goal of the tracking action is not achieved, the posterior over search goals is updated, and the process repeated. This is just another instance of planning in belief space, where the robot acts greedily with respect to its present belief over the search goals.

Algorithm 2 Action Execution Plan

1: **procedure** EXECUTE($action, model_{prior}, goals = []$)
2: **while** Goal is not achieved **do**
3: $pose \leftarrow ComputePose()$
4: $action_{goals} \leftarrow SampleGoals(pose, action, model, goals)$
5: $state_{action} \leftarrow ExecuteAction(action_{goals})$
6: $model_{post} \leftarrow ModelPosterior(model_{prior}, state_{action}, action)$
7: $model_{prior} \leftarrow model_{post}$
8: **end while**
9: **end procedure**

IV. EXPERIMENTS

To evaluate the performance of the approach, we explore in detail the capabilities of our model and planner for the task of sorting objects, where the sub-tasks associated with the plan will be executed by a belief-space planner. The experiment was conducted using a two-wheeled dynamically balancing mobile manipulator with two 4 degrees-of-freedom (DOF) arms and a trunk rotation. The robot's sensor package includes encoder feedback on all 11 DOF and an ASUS RGB-D camera on a 1 DOF head. Control is implemented in Robot Open Source (ROS) publish-subscribe operating system [22].

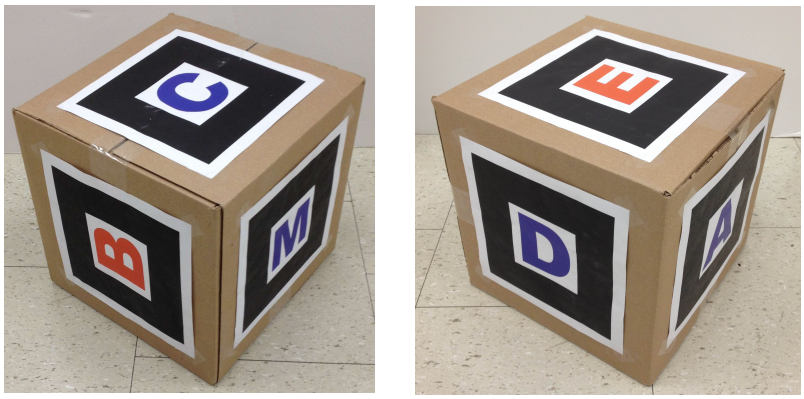


Fig. 2. Two of the boxes used in the experiments showing the various ARtag features on each of its faces. It should be evident from the images that certain features are repeated over multiple objects.

To illustrate the approach to belief-space planning in its simplest form, an extremely simple experimental object that we call an ARcube is employed. It consists of a roughly 28 cm cube made out of cardboard box, weighing about 0.4 kg with distinctive ARtag surface markings chosen from a set of 13 alternatives (A, B, C, D, E, M, T, 1, 2, 3, 4, 5, 6). These ARtags were trained using the ARToolkit [11]. Each of the

ARcubes used in these demonstrations incorporate an unique combination of six of these ARtags. Though the use of ARtag features sound simplistic, the above 13 alternatives provide a rich set of possible objects (in the order of 10^6) that can be generated by choosing 6 of the ARtags. In our experiments, the number of generated ARcube models are varied from 100 to 10000. Figure 2 shows the surface geometry of two of the ARcubes to illustrate the faces of each box with the associated ARtag. Some of the tags are repeated in multiple objects. These ambiguous features lead to partially observable state, therefore, detecting single tags is not enough for the robot to completely differentiate these objects.

The model of the objects contain a set of visual tracking actions (TRACK$_A$, ... TRACK$_T$, TRACK$_1$, ... TRACK$_6$) that track each of the ARtag features. The models also contain a set of actions that manually interact with the objects :

1) GRASP : Moves the robot base and the arms to obtain and track grasp forces in the end effector.
2) ROTATE+Y : Rotate the box counterclockwise around the Y-axis.
3) ROTATE-Y : Rotate the box clockwise around the Y-axis.
4) ORBIT+Z : Orbit the box by moving counterclockwise around the Z-axis of the box.
5) ORBIT-Z : Orbit the box by moving clockwise around the Z-axis of the box.
6) PLACE : Moves the box to a goal location by maintaining grasp on the object while moving, and then releasing it.

GRASP, PLACE, ROTATE+Y, and ROTATE-Y are TRACK-ing actions that depend on probabilistic search distributions describing the placement of contacts that cause force closure or a net moment on the object. These distributions are defined spatially in terms of sufficient combinations of visual trackers. ORBIT+Z and ORBIT-Z are SEARCH actions that reliably change the visual aspects by driving the mobile manipulator to a new position and heading relative to the estimate of object pose.

The task requires the robot to sort objects into two different locations based on the object being presented. Object are presented one at a time at random locations inside a $5m \times 5m$ area. The pseudocode for the task is shown in Alg 3. The pseudocode describes a high level task that requires the robot to recognize the object and then execute a sequence of actions to pick-and-place the object in its target location. Though the plan at this level can be expressed deterministically as a sequence of sub tasks—RECOGNIZE, GRASP, and PLACE—each sub-task in itself requires the robot to reason in a partially observable world. The task proceeds by calling the RECOGNIZE algorithm (Algorithm 1) to reduce uncertainty over objects (Line 2). Initially, when the robot has made no observations, it has uniform belief over objects. However, as the planner executes actions and makes new observations, it fuses these observations with its belief to compute a new posterior over objects. This process gets repeated until the object has been recognized—which is given by the entropy falling below a particular threshold.

Algorithm 3 Task: Sorting

```
1: procedure SORTING(o_prior, object_goal, goal_1, goal_2)
2:     object_recog ←RECOGNIZE(o_prior)
3:     EXECUTE(grasp, object_recog)
4:     if object_recog == object_goal then
5:         EXECUTE(place, object_recog, goal_1)
6:     else
7:         EXECUTE(place, object_recog, goal_2)
8:     end if
9: end procedure
```

This procedure leads to different plans based on the initial presentation of the object and the history of the agent's observation. Figure 3 shows one run of the planner when the robot had access to 100 models.

TABLE I
VARIATION IN PERFORMANCE OF THE PLANNER WITH THE NUMBER OF OBJECTS.

No. of Models	100	1000	10000
Avg. Planning time (in secs)	0.43	2.29	7.02
Avg. number of actions	3.3	3.8	5.2

Once the object has been recognized, the robot calls Algorithm 2 multiple times with different actions and goals to reduce its uncertainty over achieving the task of GRASPING and PLACING the object in its goal location.

Figure 4 shows how the planner's uncertainty over the objects change with the number of executed actions. For the sake of clarity, only 5 trials are shown on the plots. As is evident from the plots, the entropy decreases monotonically with increasing actions, even as the number of possible objects is increased from 100 to 10000. This indicates that the information theoretic planner is making greedy optimal decisions to reduce uncertainty at every iteration. For Trial 3 with 10000 object models, we notice that the entropy stays constant for a couple of actions. This happened in the case when the action failed in its execution because of noisy sensor measurements. However, after making observations following the failed action execution, the robot was able to correctly reason and select the next best action.

Furthermore, since the planner only computes the next best action to execute, the computation time required is minimal. Table I shows how the average planning time varies with the number of available object models. All experiments were conducted on a single quad-core machine with 16 GB of RAM. The machine performed both the vision processing as well as the action selection. The planning code was written in Python. As can be seen from the table, the planner can build plans in real-time even for 10000 objects. This is because after every observation, the planner searches for plans in a reduced set of possible objects. Figure 5 shows the size of the hypothesis space as the planner executes actions and makes new observations. It is evident that nearly 90% of the hypothesis space gets pruned after every action. This leads to a speed up in the planning process with increased number of observations. Table I also tabulates the average number of actions required to recognize the objects. The actions are averaged over 20 trials. The table shows that as the number of models increase, the performance of the planner degrades. This can be attributed to the increased pairwise similarity between objects as the number of models increases.

V. CONCLUSIONS

This paper describes a domain-general state representation that is derived directly from the actions that the robot can execute. We presented a Bayesian framework that utilizes this representation for building models of objects in its environment. We then showed the strengths of combining this representation with a Belief-space planner that allows various common tasks to be expressed as uncertainty reduction. For future work, we are interested in studying how a belief space planner can be used to solve tasks involving multiple objects in the scene. In such cases, the planner needs to reason about the effect of its action on multiple objects in the environment. We are working on extending our techniques to such scenarios involving movable obstacles. We are also interested in extending our plans as well as models to multi-object assemblies.

ACKNOWLEDGEMENTS

This material is based upon work supported under Grants NASA-GCT-NNH11ZUA001K and ONR-MURI-N000140710749.

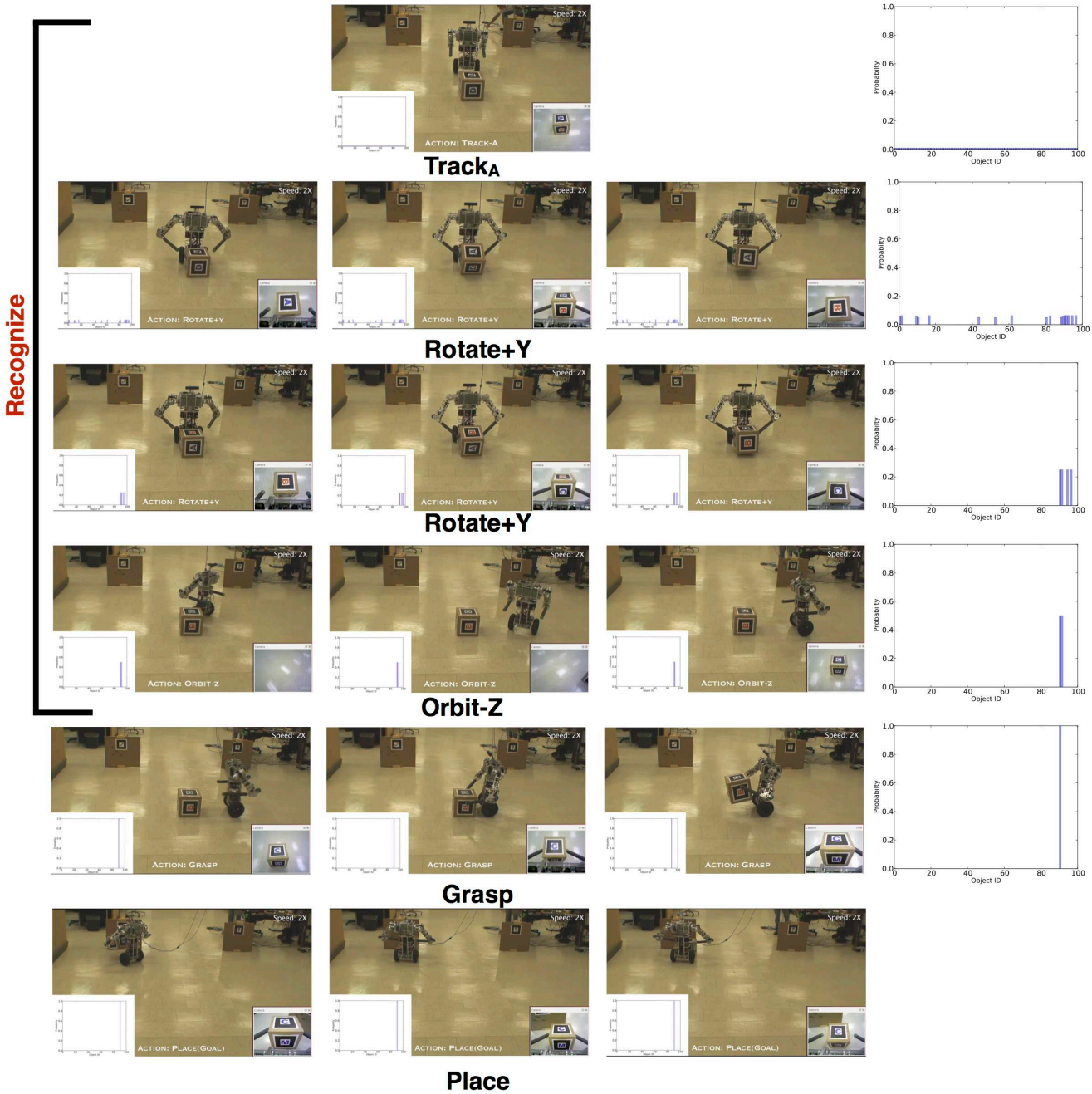
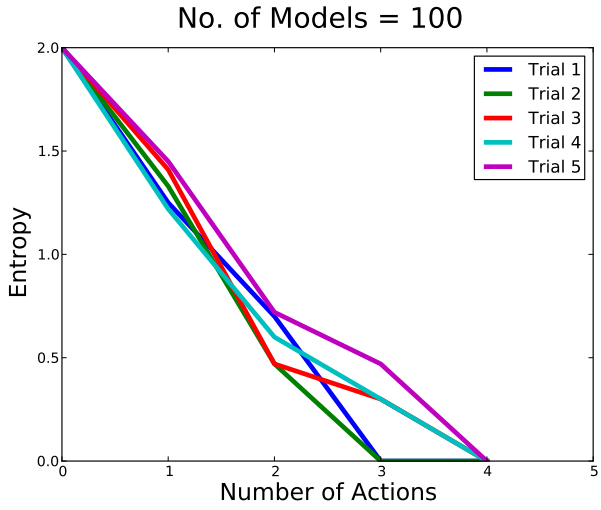
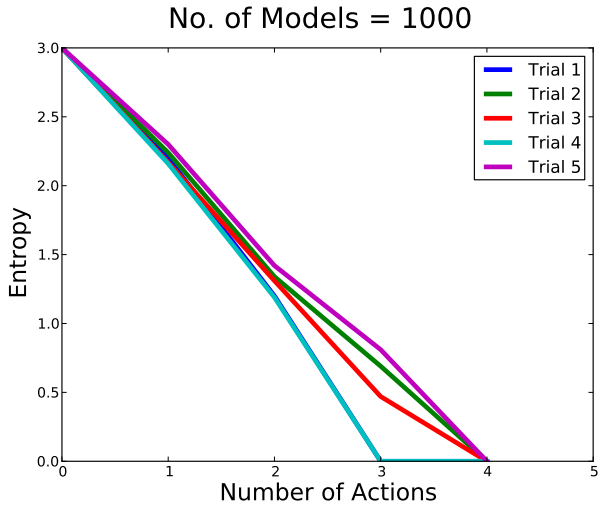


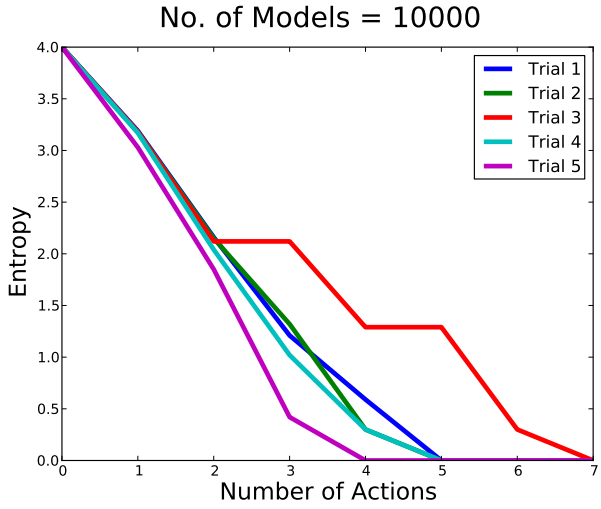
Fig. 3. Figure shows the execution of a plan. The high level task requires the robot to recognize the object and place the object in its target location. The plots in the rightmost column show the posterior probability over objects.



(a)

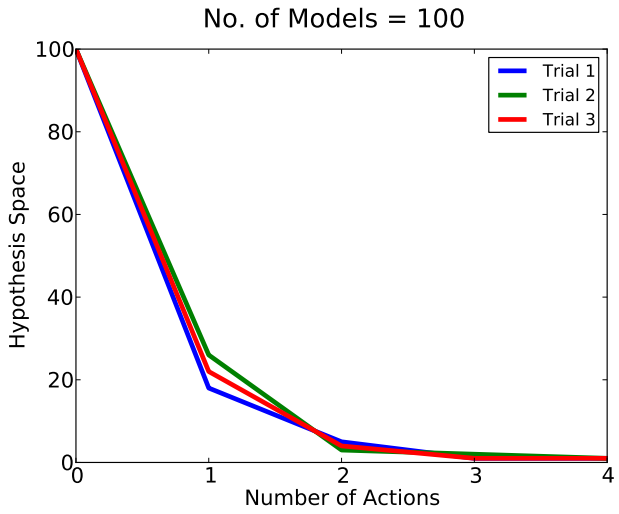


(b)

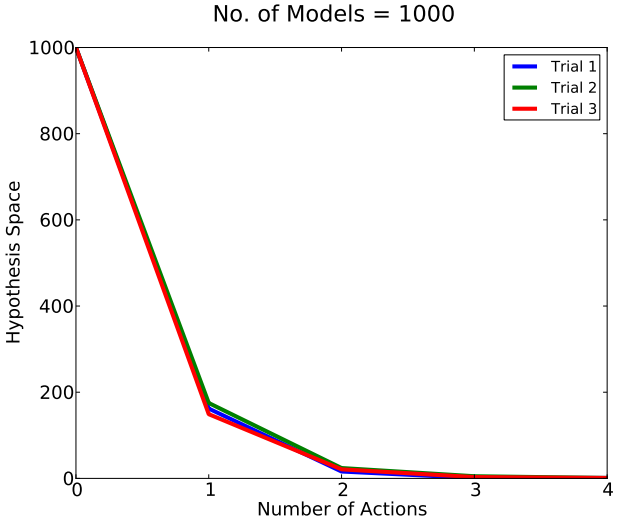


(c)

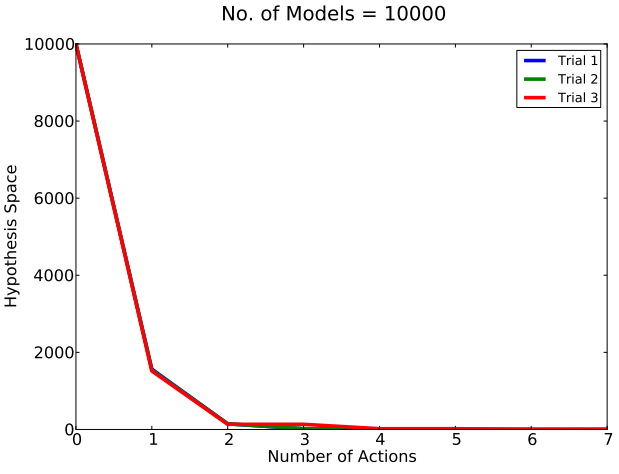
Fig. 4. The change in object entropy as a function of the number of actions executed by the robot. The entropy is computed over (a) 100 objects, (b) 1000 objects, and (c) 10000 objects.



(a)



(b)



(c)

Fig. 5. The change in space of number of object hypotheses remaining as a function of the number of actions executed by the robot.

REFERENCES

[1] R. Alami, J. P. Laumond, and T. Siméon. Two manipulation planning algorithms. In *Algorithms for Robotic Motion and Manipulation*, 1997.

[2] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[3] S. Cambon, R. Alami, and F. Gravot. A hybrid approach to intricate motion, manipulation and task planning. *International Journal of Robotics Research*, 28:104–126, January 2009.

[4] J. Coelho and R. Grupen. A control basis for learning multifingered grasps. *Journal of Robotic Systems*, 14(7):545–557, 1997.

[5] R. E. Fikes and N. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 5(2):189–208, 1971.

[6] S. Hart and R. Grupen. Learning generalizable control programs. In *Transactions on Autonomous Mental Development*, pages 1–16, Zaragoza, Spain, 2010.

[7] K. Hauser and J. Latombe. Integrating task and prm motion planning: Dealing with many infeasible motion planning queries, 2009.

[8] M. Huber. *A Hybrid Architecture for Adaptive Robot Control*. PhD thesis, Department of Computer Science, University of Massachusetts Amherst, 2000.

[9] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *IEEE Conference on Robotics and Automation Workshop on Mobile Manipulation*, 2010.

[10] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Pre-image backchaining in belief space for mobile manipulation. In *International Symposium on Robotics Research (ISRR)*, 2011.

[11] Hirokazu Kato and Mark Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*, IWAR '99, pages 85–94, Washington, DC, USA, 1999. IEEE Computer Society.

[12] D. E. Koditschek and E. Rimon. Robot navigation functions on manifolds with boundary. *Advances in Applied Mathematics*, 11(4):412–442, 1990.

[13] S. M. LaValle. Rapidly-exploting random trees: A new tool for path planning. Technical Report TR 98-11, Computer Science Dept., Iowa State University, October 1998.

[14] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, New York, NY, USA, 2006. ISBN 0521862051.

[15] T. Lozano-Pérez, M. T. Mason, and R. H. Taylor. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1):3–24, 1984.

[16] Y. Nakamura. *Advanced Robotics: Redundancy and Optimization*. Addison-Wesley, 1991.

[17] D. Nau, O. Ilghami, U. Kuter, J. W.Murdock, D. Wu, and F. Yaman. Shop2: An htn planning system. *Journal of Artificial Intelligence Research*, 20:379–404, 2003.

[18] N. Nilsson. Shakey the robot. *SRI International, Technical Report*, (323), 1984.

[19] I. Nourbakhsh. Using abstraction to interleave planning and execution. In *Proceedings of the Third Biannual World Automation Congress*, 1998.

[20] E. Plaku and G. D. Hager. Sampling-based motion and symbolic action planning with geometric and differential constraints. In *ICRA*, pages 5002–5008, 2010.

[21] R. Platt, R. Tedrake, L. Kaelbling, and T. Lozano-Perez. Belief space planning assuming maximum likelihood observations. In *Proceedings of Robotics: Science and Systems*, Zaragoza, Spain, June 2010.

[22] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[23] E. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[24] Mike Stilman, Jan-Ullrich Schamburek, James Kuffner, and Tamim Asfour. Manipulation planning

among movable obstacles. In *IEEE International Conference on Robotics and Automation*, pages 3327–3332, April 2007.

[25] J. Wolfe, B. Marthi, and S. Russell. Combined task and motion planning for mobile manipulation. In *ICAPS*, 2010.

[26] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.