Modular Task and Motion Planning in Belief Space

Dylan Hadfield-Menell[1*], Edward Groshev[2*], Rohan Chitnis[2*], and Pieter Abbeel[1]

***Abstract*— The execution of long-horizon tasks under uncertainty is a fundamental challenge in robotics. Recent approaches have made headway on these tasks with an integration of task and motion planning. In this paper, we present Interfaced Belief Space Planning (IBSP), a modular approach to task and motion planning in belief space. We use a task-independent interface layer to combine an off-the-shelf classical planner and motion planner, enabling us to solve complex problems in belief space. We determinize the problem under the maximum likelihood observation assumption to obtain a deterministic representation that can generate reasonable information-seeking behavior. We leverage properties of maximum likelihood determinizations to utilize a simple representation of (optimistic) belief space dynamics that is well-suited to planning. Our interface can be implemented with standard belief state queries, requiring only the ability to sample, compute unnormalized likelihoods, and answer maximum likelihood queries. Our contribution is a novel algorithm for task and motion planning in belief space. The algorithm is a novel approach that substantially reduces dependence on details of the belief state dynamics. IBSP can work with a broad class of black box state estimators, with zero changes to the algorithm. We show that IBSP is complete for a (restricted) class of partially observable problems. We validate our approach in simulated tasks for the PR2 that account for continuous state, different belief distributions, and negative observations.**

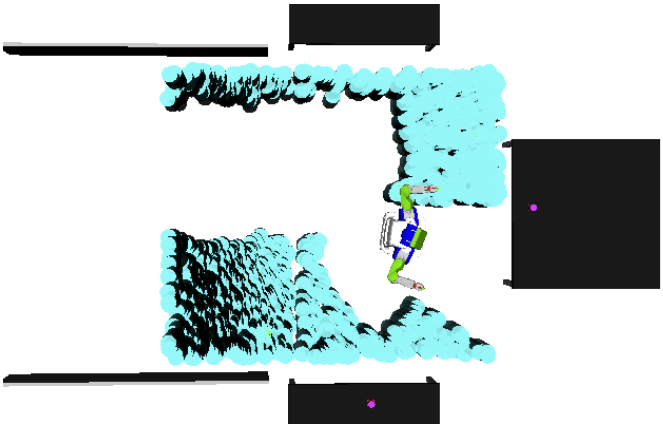

Fig. 1: A screenshot from one of our experiments. The robot is tasked with navigating to the other side of the corridor through a uniform distribution over obstacles. Our low level refinement algorithm detects the probability of obstruction and propagates this information to the high level. The objects shown are from the posterior distribution after several negative observations.

I. INTRODUCTION

A central goal in robotics is the execution of long-horizon tasks in the face of uncertainty. Any reader that has spent frantic mornings searching for lost keys understands the challenges such problems present. Completing such a task corresponds to reasoning about multimodal belief distributions where obtaining an observation can require long, temporally extended plans.

Discovering a solution to this task necessitates finding a policy that accounts for uncertainty in locations of objects, uncertainty in robot position, and non-determinism in the dynamics (among other challenges). Solving such problems exactly is far beyond the state of the art in partially observable Markov decision processes.

Given the intractability of this problem, how can we hope to make progress? Our starting point takes inspiration from recent methods for fully observed task and motion planning. This is a challenge in its own right, but careful applications of abstraction, lazy discretization, and motion planning have made inroads on this problem [1], [7]. These approaches plan in an *abstract* representation of a problem (without continuous variables) then use sampling and motion planning to *refine* abstract plans into fully grounded plans involving continuous operators.

A key insight is that many of the difficulties in planning with belief states, which are inherently continuous, are shared by the continuous state-action spaces for task and motion planning. This suggests a strategy that applies similar abstraction techniques to belief state representations of a planning problem.

Another important development is that of *maximum likelihood determinizations* (MLD) of POMDPs [4]. This approximation assumes that each belief state produces its maximum likelihood observation. The result is a deterministic problem that encourages reasonable information-gathering behavior. It is amenable to suitably modified standard methods; these methods typically prescribe solving a complex, continuous, under-actuated problem. Integration with task planners provides relief from this problem—high level tasks often decompose a problem so that observation planning and motion planning are separate.

We present an algorithm, Interfaced Belief Space Planning (IBSP), that leverages these insights. It extends the domain abstraction techniques from Srivastava et al. [1] to MLD formulations of large continuous POMDPs. We leverage properties of the MLD to construct a simple high-level representation of belief states in order to compactly represent (optimistic) belief state dynamics. The algorithm

[1]{dhm, pabbeel}@eecs.berkeley.edu
[2]{eddiegroshev, ronuchit}@berkeley.edu
* *denotes equal contribution among these authors*

alternates among extracting a plan skeleton, assigning values to continuous variables, and updating the high level with failure information when a refinement cannot be found. IBSP enforces a clear separation among extracting plan skeletons from a domain description, refining a plan skeleton, and determining success or failure of a plan; furthermore, each component relies on standard operations. Our implementation uses off-the-shelf classical planners to generate plan skeletons along with sampling and trajectory optimization to generate refinements.

Our refinement strategy only requires the ability to compute maximum likelihood observations for a belief state and the ability to filter beliefs. We detect plan failure and propose state updates with Monte Carlo sampling. Such structured access to the belief state allows IBSP to work with a broad class of beliefs with minimal algorithmic changes. While our high-level planner uses an abstract belief representation, the full algorithm plans with the true belief dynamics and is capable of generating and executing plans with complex state estimation methods. We validate our approach in partially observable mobile manipulation tasks with a variety of belief distributions that account for negative observations, such as object retrieval from one of many possible drawers and navigation through a corridor containing several objects whose locations are only initially known to be in a uniform distribution over the corridor. We present formal results showing completeness for a subclass of POMDPs.

II. Related Work

Our work builds on recent results in deterministic task and motion planning. Early work in task and motion planning was embodied in the aSyMov system [8], where a task plan was generated and used to guide motion planning. However, the focus was on determining a way to solve motion plans in parallel. Dornhege et al. use semantic attachments to do task and motion planning with classical planners [2]. This approach makes use of task planners, but requires representation of discretized locations within the high level. Havur et al. use local search to find an optimal geometric configuration before discretizing the continuous plane into non-uniform cells and using a hybrid planner to find a feasible motion plan [**?**]. Lagriffoul et al. parameterize symbolic actions with coarse geometric representations of objects and grasp types. Possible geometric instances are sampled from the symbolic plan and are pruned by constraints to reduce geometric backtracking [**?**].

Maximum likelihood determinization was introduced by Platt et al. [4]. They used this approach to frame problem solving in belief space as an underactuated control problem. They use LQR and transcription methods to find trajectories in belief space and characterize scenarios where a replanning strategy is guaranteed to succeed. Van Den Berg et al. use LQG to solve Gaussian belief space planning problems that incorporate collision avoidance constraints [9]. They are able to plan without the maximum likelihood assumption and explore the approximations introduced with maximum likelihood determinizations.

Our determinization-replan approach shares similarity with determinization-replan approaches for probabilistic planning, such as FF-replan [10]. Both leverage determinization in order to obtain major performance gains from classical planners. The maximum likelihood assumption is similar to the most likely transition determinization from that literature, while our optimistic belief updates are similar to an all outcomes determinization.

The idea of using knowledge space representations is an old idea in artificial intelligence that dates back to McCarthy and Hayes [11]. Bonet and Geffner provide in-depth experimentation and analysis of discrete deterministic partially observed planning [5]. They provide conditions under which discrete formulations of partially observable planning (with binary, factored beliefs) can be compiled to sound and complete representations. The conditions for this are similar to conditions for our completeness theorem; however, our algorithms are aimed at large continuous problems for robotics while their work is aimed at classical planning.

The BHPN planning and execution system is a task and motion planning approach to handle uncertainty [6]. Similar to our approach, they construct task plans in belief space under maximum likelihood determinization. Levihn et al. extend this system to construct shorter plans at execution time by smart replanning and reconsideration [12]. They explicitly formulate the regression of belief goals under Gaussian distributions and plan with an exact represention ofe belief state dynamics. In contrast, we use references to the belief so we can plan with arbitrary belief representations, not just Gaussians.

Srivastava et al. formulate open world POMDPs with a probabilistic program [13] – they develop a generalization of point-based value iteration to that setting. While both our method and theirs can been viewed as solving very large POMDPs, their method's size stems from the (unbounded) size of the world and complexity of corresponding beliefs, while ours stems from uncertainty about continuous quantities.

Gashler et al. use a contingent planner to generate plans that react to feedback from the world [14]. They generate contingent plans and use external function calls during their planning step to generate effects of actions. However, they explicitly represent poses and belief updates in symbolic models for task planning. We use pose and belief references to plan abstractly. We refine and verify plans with the *true* belief.

Nebel et al. use a three valued logic for the TidyUp project that allows fluents to take the value *uncertain* [16]. In their system, however, they assume that uncertain items become known once the robot is close enough and so do not explicitly plan sensing actions. In contrast, we represent uncertainty directly and combine reasoning about uncertainty with reasoning about maximum likelihood states.

III. Technical Background

A. Observable planning formulation

We use a STRIPS-style formalism to define our planning problems. We introduce a simple pick and place task that will serve as a running example for the paper.

Formally, we define a (fully observed) planning problem, Π as a tuple, $\Pi = \langle \mathcal{E}, \mathcal{F}, I, G, \mathcal{A} \rangle$, with the following definitions:

$\mathcal{E}$: a set of *entities* within our domain; e.g. individual objects or locations.

$\mathcal{F}$: a set of *fluents* that describe relations between entities.

$I \in 2^{\mathcal{F}}$: a conjunction of fluents that are initially true.

$G \in 2^{\mathcal{F}}$: a conjunction of fluents that characterizes the set of goal states.

$\mathcal{A}$: a set of parametrized *operators* that describes ways the agent can alter the world. Each *op* $\in \mathcal{A}$ is characterized by: *preconditions*, *pre(op)*, a conjunction of fluents that captures the set of states where an operator is applicable; and *effects*, *eff(op)*, which specifies the fluents that become true or false after executing *op*.

A solution to Π is a sequence of operators and states, $\{(op_1, s_1), \ldots, (op_N, s_N)\}$, such that preconditions are satisfied and the goal is achieved. That is, $s_i = apply(eff(op_i), s_{i-1})$; $pre(op_i) \in s_{i-1}$, $pre(op_0) \in I$; $s_N \in G$. We now give a specification for our pick and place task:

$\mathcal{E}$ objects (o_i, represented as strings), Null (a nonexistent object), object poses (continuous 6DOF), robot poses (continuous 20DOF), robot trajectories (continuous $20 \cdot T_{max}$ dimensional), and grasps (continuous 6DOF).

$\mathcal{F}$ *Loc*(obj, obj_pose), *RLoc*(robot_pose), *Obstructs*(obj, traj), *Holding*(obj, grasp), *Connects*(rob_pose1, rob_pose2, traj), *GraspPose*(obj, obj_pose, r_pose, grasp).

$\mathcal{A}$
- *MoveTo*(r_pose1, r_pose2, traj)
  pre: *RLoc*(r_pose1) $\wedge$ *Connects*(r_pose1, r_pose2, traj) $\wedge \forall i \neg Obstructs(o_i, traj)$
  eff: *RLoc*(r_pose2) $\wedge$ $\neg$*RLoc*(r_pose1)
- *Pick*(obj, obj_pose, r_pose, grasp)
  pre: *RLoc*(r_pose) $\wedge$ *Holding*(Null, Null) $\wedge$ *GraspPose*(obj, obj_pose, r_pose, grasp) $\wedge$ *Loc*(obj, obj_pose)
  eff: *Holding*(obj, grasp) $\wedge$ $\neg$*Loc*(obj, obj_pose) $\wedge$ $\neg$*Holding*(Null, Null) $\wedge \forall$t $\neg$*Obstructs*(obj, t)
- *Place*(obj, obj_pose, r_pose, grasp)
  pre: *RLoc*(r_pose) $\wedge$ *Holding*(obj, grasp) $\wedge$ *GraspPose*(obj, obj_pose, r_pose, grasp)
  eff: $\neg$*Holding*(obj, grasp) $\wedge$ *Loc*(obj, obj_pose) $\wedge$ *Holding*(Null, Null) $\wedge \forall$ t s.t. *overlaps*(obj, obj_pose, t): *Obstructs*(obj, t)

B. Abstraction of continuous variables

While this representation unambiguously specifies problems, the presence of continuous parameters and conditional effects that depend on this parameters (e.g., new obstructions from placing an object) prevents directly passing this representation to a classical planner. We will need to compile our specification into a discrete problem that classical planners can readily solve.

Algorithm 1 An interface for deterministic problems

```
1:  procedure InterfacePlan(S_0, γ)
2:      S ← S_0
3:      s ←ExtractSymbols(S_0)
4:      p_pre ← None
5:      R_pre ← None
6:      p_post ← Plan(s, γ)
7:      while not resource limit reached do
8:          R_post ← MotionPlan(s, p_post)
9:          is_fail, i_fail, failure
10:             ← CheckSuccess(S, (p_post, R_post))
11:         if ¬is_fail then
12:             return (p_pre + p_post, R_pre + R_post)
13:         end if
14:         p_pre ← p_pre + p_post[: i_fail]
15:         R_pre ← R_pre + R_post[: i_fail]
16:         S ← ApplyActions(S_0, p_pre, R_pre)
17:         s ← ExtractFailureSymbols(S, failure)
18:         p_post ← Plan(s, γ)
19:         if max attempts reached then
20:             reset pose generators
21:             reset S, s, p_post, p_pre, R_pre
22:         end if
23:     end while
24: end procedure
```

We achieve this through the abstraction of continuous aspects of our state in the style of Srivastava et al. [1]. The key step in this approach replaces continuous variables by a discrete set of references to useful potential values. These references are included as objects in the domain description with appropriate facts for the symbolic planner to interpret them.

For example, in order to derive an abstract representation of the pick operation for object o_i, we need a reference to the pose of the object and the grasp we will use to pick it. We represent these as functions: *Pose*(o_i), *Grasp*(o_i), and *GP*(o_i, *Pose*(o_i), *Grasp*(o_i)). The proper interpretation for the last of these is 'the pose from which we will choose to grasp o_i when it is at…'. These are represented to the classical planner as standard, discrete objects; we then declare facts about these variables that are guaranteed to be true. This process is referred to as *skolemization*.

To capture the effects of actions that use these parameters, we use *sound* representations: facts which are guaranteed to become true are included, but conditional effects are not. These are effects that depend on the particular binding of a skolem variable. For example, after executing a place action we know that *Loc*(o_i, Pose(o_i)) will be true, but the particular obstructions introduced will depend on the exact bindings of

Pose(o_i). We include the former in our description and rely on an interface layer to discover conditional facts, such as the latter, as they become relevant and update the high level description accordingly. The process of assigning values to the variables of a high level plan is called *refinement* of the high level plan.

Algorithm 1 shows pseudocode for such an interface layer. Lines 1-5 set up our planning problem. Line 6 constructs a high level plan. Lines 8-10 attempt to find a motion plan for the high level plan. If refinement fails, we identify a fact that was missing from the high level plan that may have caused failure, and the high level then replans from that step. This provides a random trajectory (as the refinement step is randomized) through the solution space. If we cannot find a solution, we reset random seeds and replan from the initial state. This integration is complete for a subclass of planning problems [1].

C. Assumed maximum likelihood determinization

In this paper, we consider problems from a more general class than is described in Section III-A – problems with partial observability. To solve partially observable problems with deterministic solvers, we use the *maximum likelihood determinization* (MLD) [4].

The introduction of partial observability into a planning problem associates with each state a distribution over observations. For example, a state where a robot's depth sensor is pointed at an object might give a noisy measurement of the object's pose. A solution for this problem is a policy that maps a *belief state*, a probability distribution over states, to actions.

The central idea behind the MLD is that, if we fix the observation for each state, the Bayesian updates to our belief are deterministic. For example, in the Gaussian case, belief dynamics are defined by the Kalman filter update equations, and fixing the observation for each state reduces this to a deterministic update. While there are many determinizations of belief space problems, the benefit of the MLD is that it is a deterministic problem that still generates information-seeking behavior.

IV. Interfaced belief space planning

In this section, we present our primary contribution: IBSP, a modular approach to task and motion planning in belief space. Section IV-A extends our symbolic representation to include belief states. Section IV-B details an approach based on optimistic observations and a simplified belief representation to compactly represent belief space dynamics. Section IV-C describes the IBSP algorithm in detail and provides conditions under which it is complete.

A. Belief space planning formulation

In our full description of the problem, we need to allow for probabilistic representations in belief space. We restrict ourselves to problems where the uncertainty is confined to the continuous state (e.g., we may not know o_1's location, but we know its name or whether we are holding it). The entities for our partially observed problems fall into one of the following three categories:

- named objects
- continuous variables (either object attributes or observations)
- distributions over continuous variables

Our fluents will describe relations between these entities. The initial state and goal are defined in the same manner. Our input additionally specifies a conditional distribution over observations given the state.

We assume that our operators are separated into *actions*, which alter the true state but do not get any information about it, and *observations*, which do not alter the true state but garner information about it. Actions are deterministic in belief space (even when the underlying dynamics are not) and follow a similar parametrization to the deterministic case.

The source of non-determinism in our specification is confined to observations. An observation operator is parametrized by the belief about state variables that determine its conditional distribution. To formalize the MLD, we include a fluent, *MLObs*(obs, p_1, p_2, ...), that is true iff obs is the maximum likelihood observation under the state distribution specified by the p_i. We include the observation as a parameter of the operator and include the *MLObs* fluent as a precondition. This provides us a deterministic planning problem of the kind described in Section III-A.

It is tempting to attempt to apply the skolemization approaches from [1] to this representation. Unfortunately, that is unlikely to be successful in practice. The issue is that the number of skolemized observation variables will typically be exponential in the number of skolemized beliefs. For example, consider the following observation operator for our pick-and-place domain:

Observe(obs, r_bel, o_1_bel, ..., r_bel', o_1_bel', ...)
pre: *Rloc*(r_bel) $\wedge$ *Loc*(o_1, o_1_bel) $\wedge$ ...
$\wedge$ *Preimage_RLoc*(obs, r_bel, r_bel')
$\wedge$ *Preimage_At*(obs, o_1, o_1_bel, o_1_bel') $\wedge$...
$\wedge$ *MLObs*(obs, r_bel, o_1_bel, ...)
eff: $\neg$*Rloc*(r_bel) $\wedge$ $\neg$*Loc*(o_1, o_1_bel) $\wedge$ ...
Rloc(r_bel') $\wedge$ *Loc*(o_1, o_1_bel') $\wedge$ ...

Creating a skolem function for the *MLObs* fluent will require considering all combinations of the other beliefs. For small problems this approach may be reasonable, but for larger problems this direct abstraction is intractable.

B. Optimism for observations

The insight we use to solve this issue is that the MLD has very restricted dynamics for our problem class. An observation operator can only concentrate belief at the maximum likelihood location for a distribution. An action operator, on the other hand, could change the maximum likelihood location of a distribution (and potentially introduce variance into our beliefs). Thus, we reformulate our problem in terms of two types of fluents: those which characterize the maximum likelihood state, and those which characterize the certainty in our belief. The latter of the two shares similarities with the BVLoc(ϵ) fluents from Kaelbling and Lozano-Pérez [6].

We will illustrate this simplification with the *Pick* operator. To simplify description, we represent objects in the frame of the robot and assume that the robot's pose in its own frame is known. It is possible to relax this assumption at the cost of introducing more complex descriptions.

Using a full belief representation, we have the following description:

BPick(o, o_pose_bel, r_pose, grasp)
pre: *BLoc*(o, o_pose) ∧ *RLoc*(r_pose)
∧ *BGraspPose*(o, o_pose_bel, r_pose, grasp)
∧ *Holding*(Null, Null)
eff: *Holding*(o, grasp) ∧ ¬*BLoc*(o, o_pose)
∧ ¬*Holding*(Null, Null)
∧ ∀t, h, g ¬*BObstructs*(o, t, h, g)
∧ ∀ o_p, r_p ¬*BOccludes*(o, o_p, r_p)

BGraspPose characterizes distributions over the location of o such that the probability with which a grasp succeeds is higher than some safety margin. The issue is that this is only a conditional effect of observation actions, so a sound representation can never plan to make this true.

To rectify this issue we use the following property about an MLD representation. If *BGraspPose* is not true, then one of two cases will hold: either 1) the maximum likelihood state is not a successful grasp pose or 2) the maximum likelihood state admits a successful grasp, but the distribution has too much variance to allow a successful grasp with high probability. In the first case, we must manipulate the state, while in the other we must observe it. We thus replace belief fluents with maximum likelihood fluents and *uncertainty* fluents:

MLPick(o, o_pose_bel, r_pose, grasp)
pre: *MLLoc*(o, o_pose_bel) ∧ *MLRLoc*(r_pose_bel)
∧ *MLGraspPose*(o, o_pose_bel, r_pose, grasp)
∧ ¬*UncertainGP*(o_pose_bel, r_pose, grasp)
∧ *Holding*(Null, Null)
eff: *Holding*(o, grasp) ∧ ¬*MLLoc*(o, obj_pose)
∧ ¬*Holding*(Null, Null)
∧ ∀t, h, g ¬*MLObstructs*(o, t, h, g)
∧ ∀ o_p, r_p ¬*MLOccludes*(o, o_p, r_p)

In order to achieve ¬*UncertainGP* facts, we include an observation operator:

ObserveGP(r_obs_pose, o, o_pose)
pre: *MLLoc*(o, o_pose) ∧ *MLRLoc*(r_obs_pose)
∧ ∀o_i¬*MLOccludes*(o_i, o_pose, r_obs_pose)
∧ *MLInView*(r_obs_pose, o_pose)
eff: ∀ r_p, g ¬*UncertainGP*(o_pose_bel, r_p, g)

This corresponds to a conditional effect of the original *Observe* operator if we try to include *UncertainGP* fluents in that representation: the true effect can only be guaranteed with complete knowledge of the true belief. However, in order to correct a high level plan, we must include these fluents as a possibility. We call this an *optimistic* observation because the actual effect is not guaranteed, and we thus rely on the interface layer to discover cases when it does not hold and correct our representation. Note that this assumption of optimism is only made in the description for the high level. The full algorithm plans with a complete belief representation; if multiple observations are needed to successfully localize an object, the algorithm will plan to do them.

C. The IBSP algorithm

In this section, we detail the necessary changes to the interface from Srivastava et al. to handle our belief space formulation. The primary change is that the refinement operation must find an assignment of continuous variables that maximizes the probability of 'success.' Algorithm 2 shows pseudocode with changes to Algorithm 1 highlighted in bold.

Algorithm 2 The IBSP planning algorithm

procedure IBSP(B_0, γ, ϵ)
  $B \leftarrow B_0$
  $b \leftarrow$ ExtractSymbols($B_0, None$)
  $p_{post} \leftarrow$ Plan(b, γ)
  $p_{pre} \leftarrow None$
  $R_{pre} \leftarrow None$
  while not resource limit reached **do**
    $R_{post} \leftarrow$ **MLRefine**(B, p_{post})
    $is\_fail, i_{fail}, failure$
     $\leftarrow$ **BCheckSuccess**($B, (p_{post}, R_{post}), \epsilon$)
    if $\neg is\_fail$ **then**
      return $(p_{pre} + p_{post}, R_{pre} + R_{post})$
    end if
    $p_{pre} \leftarrow p_{pre} + p_{post}[: i_{fail}]$
    $R_{pre} \leftarrow R_{pre} + R[: i_{fail}]$
    $B \leftarrow MLFilter(B_0, p_{pre}, R_{pre})$
    $b \leftarrow$ ExtractFailureSymbols($B, failure$)
    $p_{post} \leftarrow$ Plan(b, γ)
    if max attempts reached **then**
      reset pose generators
      reset $B, b, p_{post}, p_{pre}, R_{pre}$
    end if
  end while
end procedure
procedure IBSP-EXECUTE($B, \gamma, world, \epsilon$)
  $(p, R) \leftarrow IBSP(B, \gamma, \epsilon)$
  for $i \in len(p)$ **do**
    $B \leftarrow$ ExecuteAndFilter($p[i], R[i], B, world$)
    if ¬BCheckSuccess($B, (p[i+1:], R[i+1:])$) **then**
      return IBSP-execute($B, \gamma, world, \epsilon$)
    end if
  end for
end procedure

We take as input a safety parameter ϵ. This determines the acceptable probability of failure for each step in the plan. The first change is that we replace *MotionPlan* with *MLRefine*. The difference is that *MotionPlan* does collision-free motion planning to connect successive states, while *MLRefine* minimizes the probability of collision. *MLRefine* can be defined either by direct trajectory optimization that minimizes an unnormalized likelihood of collision, or by

sampling objects from the belief state and minimizing the number of samples that are in collision.

The second change is the introduction of *BCheckSuccess*, which examines a refinement and checks that the preconditions are satisfied. We assume that preconditions are defined by a function that will return true or false indicating whether the precondition holds or not, given a state of the world. We determine that each of these is satisfied with probability greater than $1 - \epsilon$ by Monte Carlo sampling from the belief state. $N_{samples}$ determines the fidelity of this sampling.

When we detect failure for a belief fluent, we must decide which type of failure information to propagate to the high level. We do this by checking if the predicate holds in the maximum likelihood state. If it does, then our error is caused by uncertainty in our belief state, and so we propagate an uncertainty failure. Otherwise, we return a maximum likelihood fluent as the violated precondition. The exceptions to this rule are predicates about observations (e.g. occlusions). Because these are deterministic in our representation, it does not make sense to propagate uncertainty failures in this case. Algorithm 3 illustrates these ideas.

Algorithm 3 Determining failure or success of a refinement

```
1:  procedure BCheckSuccess(B, p, R, ε)
2:      for p_i, R_i do
3:          preds ← ExtractPreconds(p_i, R_i)
4:          W ← Sample(B, N_samples)
5:          for pred ∈ preds do
6:              if PercentSucceeds(pred, W) < 1 − ε then
7:                  if ¬pred.eval(B.ML) or IsObs(pred) then
8:                      return (False, i, ¬ML(pred))
9:                  else
10:                     return (False, i, Uncertain(pred))
11:                 end if
12:             end if
13:         end for
14:     end for
15:     return (True, -1, None)
16: end procedure
```

An important observation is that these implementations only require us to be able to sample from our belief distribution and/or compute unnormalized likelihoods. In order to compute the MLD, we need to be able to compute a maximum likelihood state estimate and maintain a filtered distribution. These are generic operations for belief states and are typically implemented for most state estimation techniques. This lightweight dependence on the form of the belief distribution allows one to switch state estimators with no change to the algorithm. Thus, our result is a belief space planning algorithm that operates with only black box access to the belief distribution. As an example, our experiments run with no modification to the planning code for uniform distributions, unimodal Gaussian distributions, and multimodal Gaussian distributions.

We can extend the completeness proof from Srivastava et al. [1] to provide conditions that the MLD for a planning problem will be solved by IBSP.

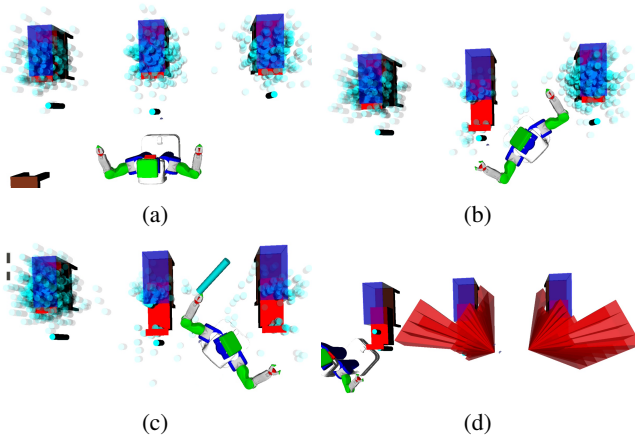

Fig. 2: Intermediate belief states from a sample execution in our drawer search domain. We search for an object with a multimodal Gaussian distribution over three drawers. We illustrate beliefs with samples from the posterior distribution and show likelihoods with the transparency of objects. In this case, the object was in the leftmost drawer, and was found after searching the other drawers because they had a higher likelihood of containing the object. (a) shows the initial state and (b) and (c) show intermediate states at which we had to replan. Notice that we only clear the obstruction in front of the drawer when it is close enough to hinder sufficient opening. (d) shows the view cones from negative observations that were generated during planning.

Theorem 1: Let P be a partially observable planning problem with deterministic transitions. If the fully observed version of P satisfies *InterfacePlan*'s conditions for completeness, then Algorithm 2 will find a solution to the MLD of P with error rate ϵ as $N_{samples} \to \infty$, provided that all the calls to motion planning return.

Proof: (sketch) Deterministic state transitions imply that the domain is uniform with respect to uncertainty predicates. The remaining predicates characterize the fully observed problem and satisfy the conditions for completeness by assumption. In the limit of infinite samples, we are able to exactly evaluate whether predicates are satisfied. These combine to satisfy the conditions to apply Theorem 1 from [1]. ■

V. Experiments

We evaluate IBSP in partially observable pick-and-place and navigation tasks. The robot in our simulation is a full model of the Willow Garage PR2. We provide results for several belief distributions: unimodal Gaussian, multimodal Gaussian, and uniform.

A. Observation Operator Specification and Generators

The maximum likelihood component of the determinized problem essentially follows the dynamics specified in Sec-

(a) Initial belief state (b) After 1 observations (c) After 2 observations (d) After 3 observations

Fig. 3: Execution trace of interfaced belief space planning traversing a corridor with uncertain obstacles. The blue cylinders are drawn from the uniform belief distribution during execution. After reaching the other side of the corridor, the robot will pick up two target tables from the three tables.

tion III-A. In order to extend this domain to handle partial observability, we add the following fluents and operators:

Fluents:

MLInView(r_pose, o_pose_bel)
MLOccludes(obj, o_pose_bel, r_obs_pose)
MLObstructs(obj, traj)
UncertainGP(o_pose_bel, r_pose, grasp)
UncertainObstructs(obj, traj)

Operators:

ObserveGP(r_obs_pose, o, o_pose_bel)
pre: *MLLoc*(o, o_pose_bel) $\wedge$ *MLRLoc*(r_obs_pose)
$\wedge$ $\forall o_i \neg$*MLOccludes*(o_i, o_pose_bel, r_obs_pose)
$\wedge$ *MLInView*(r_obs_pose, o_pose)
eff: $\forall$ r_p, g $\neg$*UncertainGP*(o_pose_bel, r_p, g)
ObserveTrajectory(r_obs_pose, traj)
pre: *MLRLoc*(r_obs_pose)
$\wedge$ *MLInView*(r_obs_pose, traj)
$\wedge$ $\forall o_i \neg$*MLOccludes*(o_i, traj, r_obs_pose)
$\wedge$ $\forall o_i \neg$*MLObstructs*(obj, traj)

eff: $\forall o_i \neg$*UncertainObstructs*(o_i, traj)

The additional skolem functions needed to support this functionality are *ViewPose*(Obj) and *ViewPose*(Traj) for each object and trajectory reference. The generator for object view poses draws samples from a 1-meter unit disc around the object, and the head pose is fixed to point at the object's maximum likelihood position. The generator for trajectory view poses returns the first step along the trajectory that has non-negligible probability of being in collision. To observe the uncertain step, a viable base pose is sampled from regions with negligible collision probability along the same trajectory.

B. Observation model and state estimation for the PR2

We model observations with ray casting from a simulated head-mounted Kinect on our robot. For objects that are in the view frustum, we get a false negative with probability proportional to the amount of the object that is occluded. If we do get a positive observation, we observe the object's true pose with Gaussian noise.

In order to successfully plan for this domain, it is important that we effectively incorporate negative observations into our updates. Without these negative observations, it is impossible to determine if a trajectory is clear without observing actual objects locations.

We account for negative observations with a factored representation for the belief state. In one component, we maintain a distribution that accounts for positive observations (e.g., a Kalman filter). The other component is an explicit representation of the view cones associated with negative observations. We refer to the first component as the positive observation distribution and the second as the negative regions. To sample from this distribution, we perform probabilistic rejection sampling: sample from the positive observation distribution and then discard the sample in inverse proportion to the probability that it lies in a negative region. The inverse probability is given by $\max(1, \frac{\text{penetration depth}}{\text{object radius}})$, where penetration depth is the minimum translation distance that takes two objects out of collision and object radius is found by approximating the object as a sphere.

C. Evaluations

We implement our experiments in Python and use OpenRave [18] to represent the environment. We use trajectory optimization for our motion planning [19] and implement custom collision checking in Flexible Collision Library [**?**] to handle collisions. We use the Fast-Forward and Fast-Downward task planners to solve our high-level planning problems. Because our system works independently of the chosen task planner, we use Fast-Forward in domains involving negative preconditions, which are not supported by Fast-Downward. Experiments were run in series on Intel Core i7-4770K machines with 16GB RAM. We validate our approach in three distinct scenarios.

To explore ensuring safety in robot navigation tasks under uncertainty, we evaluate performance in a corridor domain. We place the robot at one end of a corridor and present it with the goal of traversing the corridor, then grasping objects on tables on the other side. Uniform distributions model the locations of a varying number of obstructing columns throughout the corridor; thus, the solution typically ends up as an alternating sequence of observations and motions. Note that this task would be essentially insoluble without accounting for negative observations appropriately.

The second task demonstrates the ability to reason about occlusions. We set a goal of grasping a target object from a cluttered table with other objects on it: some have deterministic locations and others have a Gaussian distribution modeling their location. The target object's location is also

Problem Type	% Solved	Avg Time (s)	Avg # Planner Runs
Uni. corridor, 1 obj	100	232	1.88
Uni. corridor, 2 obj	100	445	2.45
Uni. corridor, 3 obj	88	926	2.78
Uni. corridor, 4 obj	58	1168	2
Table, 5 obj	95	89	2.2
Table, 10 obj	95	162	2.21
Table, 15 obj	83	135	2.26
Table, 20 obj	70	229	2.03
Table, 25 obj	68	166	1.84
Table, 30 obj	48	122	1.96
3-Drawer search	93	325	2.11

TABLE I: Solution percentages, running times, and number of high-level planner calls for IBSP in occluded search and navigation tasks. Unimodal cluttered table results are from 40 randomly generated problems per number of objects. Multimodal 3-drawer search results are from 30 iterations. Uniform corridor navigation results are from 40 random locations of corridor obstructions, per number of objects. Time limit: 1800s.

drawn from a Gaussian distribution. IBSP plans to remove any occlusions, to observe the target object, and then to pick it up. In our experiments, we maintain that 3 of the objects on the table – always including the target object – are part of the belief space.

In our final scenario, the robot has lost its keys and is trying to locate them among three drawers they could be in. This setup illustrates multimodal planning behavior with a mixture of Gaussians as the prior for the target's location. Furthermore, we introduce a deterministic object in front of each drawer, which possibly obstructs it from being opened. IBSP begins by assuming the target is placed at its maximum likelihood location; accordingly, it discovers that the target is obstructed by the drawer containing it, then plans to open this drawer. If the outer deterministic object impedes the drawer from opening, this obstruction fact is also discovered and accounted for. During execution, upon performing an observation inside the drawer, our system chooses to refine the plan if the object was not actually seen (indicating that its true location is in another drawer). The new refinement incorporates this negative observation, and the system may then try opening another drawer.

Table I shows timing results for the three described tasks. Figure 2 and Figure 3 show intermediate steps and belief states for the drawer and corridor tasks. The included video demonstrates plan execution for all three tasks.

VI. Conclusion

We presented a novel approach to combining task and motion planning in belief space through the *maximum likelihood determinization* principle. We evaluated our algorithm on several domains, where it completed challenging sequences of perception and geometric reasoning by task planning in an abstract representation of the problem. Potential future work includes experimenting on a real PR2, incorporating noisy transition models between environment states, and answering maximum likelihood queries using methods other than Monte Carlo sampling (e.g. SQP).

References

[1] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," *IEEE Conference on Robotics and Automation*, 2014.

[2] C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, and B. Nebel, "Semantic attachments for domain-independent planning systems," in *Towards Service Robots for Everyday Environments*. Springer, 2012, pp. 99–115.

[3] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1470–1477.

[4] R. Platt Jr, R. Tedrake, L. Kaelbling, and T. Lozano-Perez, "Belief space planning assuming maximum likelihood observations," in *Robotics: Science and Systems (RSS)*, 2010.

[5] B. Bonet and H. Geffner, "Planning under partial observability by classical replanning: Theory and experiments," in *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, vol. 22, no. 3, 2011, p. 1936.

[6] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *The International Journal of Robotics Research*, p. 0278364913484072, 2013.

[7] T. Lozano-Pérez and L. P. Kaelbling, "A constraint-based method for solving sequential manipulation planning problems," in *IEEE Conference on Intelligent Robots and Systems*, 2014.

[8] F. Gravot, S. Cambon, and R. Alami, "asymov: a planner that deals with intricate symbolic and geometric problems," in *Robotics Research*. Springer, 2005, pp. 100–110.

[9] J. Van Den Berg, S. Patil, and R. Alterovitz, "Motion planning under uncertainty using iterative local optimization in belief space," *The International Journal of Robotics Research*, vol. 31, no. 11, pp. 1263–1278, 2012.

[10] S. W. Yoon, A. Fern, and R. Givan, "Ff-replan: A baseline for probabilistic planning." in *ICAPS*, vol. 7, 2007, pp. 352–359.

[11] J. McCarthy and P. Hayes, *Some philosophical problems from the standpoint of artificial intelligence*. Stanford University USA, 1968.

[12] M. Levihn, L. P. Kaelbling, T. Lozano-Perez, and M. Stilman, "Foresight and reconsideration in hierarchical planning and execution," 2013.

[13] S. Srivastava, S. Russell, P. Ruan, and X. Cheng, "First-order open-universe pomdps," *Proc. UAI-14*, 2014.

[14] A. Gaschler, R. Petrick, M. Giuliani, M. Rickert, and A. Knoll, "Kvp: A knowledge of volumes approach to robot task planning," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 202–208.

[15] R. P. Petrick and F. Bacchus, "A knowledge-based approach to planning with incomplete information and sensing." in *AIPS*, 2002, pp. 212–222.

[16] B. Nebel, C. Dornhege, and A. Hertle, "How much does a household robot need to know in order to tidy up your home?" in *AAAI Workshop on Intelligent Robotic Systems*, 2013. [Online]. Available: http://www.informatik.uni-freiburg.de/~ki/papers/nebel-etal-aaai13irs.pdf

[17] S. Patil, G. Kahn, M. Laskey, J. Schulman, K. Goldberg, and P. Abbeel, "Scaling up gaussian belief space planning through covariance-free trajectory optimization and automatic differentiation," in *Workshop on the Algorithmic Foundations of Robotics (Wafr)*, 2014.

[18] R. Diankov and J. Kuffner, "Openrave: A planning architecture for autonomous robotics," Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34, July 2008.

[19] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization." in *Robotics: Science and Systems*, vol. 9, no. 1. Citeseer, 2013, pp. 1–10.