2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

Online Replanning in Belief Space for Partially Observable Task and Motion Problems

Caelan Reed Garrett[1,2], Chris Paxton[2], Tomás Lozano-Pérez[1], Leslie Pack Kaelbling[1], Dieter Fox[2]

***Abstract*— To solve multi-step manipulation tasks in the real world, an autonomous robot must take actions to observe its environment and react to unexpected observations. This may require opening a drawer to observe its contents or moving an object out of the way to examine the space behind it. Upon receiving a new observation, the robot must update its belief about the world and compute a new plan of action. In this work, we present an online planning and execution system for robots faced with these challenges. We perform deterministic cost-sensitive planning in the space of hybrid belief states to select likely-to-succeed observation actions and continuous control actions. After execution and observation, we replan using our new state estimate. We initially enforce that planner reuses the structure of the unexecuted tail of the last plan. This both improves planning efficiency and ensures that the overall policy does not undo its progress towards achieving the goal. Our approach is able to efficiently solve partially observable problems both in simulation and in a real-world kitchen.**

Fig. 1: The robot pulls open a drawer to detect whether the spam object is at a continuous pose particle within the drawer.

I. Introduction

Robots acting autonomously in human environments are faced with a variety of challenges. First, they must make both discrete decisions about what object to manipulate as well as continuous decisions about which motions to execute to achieve a desired interaction. Planning in these large *hybrid* spaces is the subject of integrated *Task and Motion Planning* (TAMP) [1], [2], [3], [4], [5], [6]. Second, real-world robot actions are often quite *stochastic*. Uncertainty in the effects of actions can manifest both locally due noisy continuous actuation or more broadly due to unexpected changes in contact. Third, the robot can only *partially observe* the world due to occlusions caused by doors, drawers, other objects, and even the robot itself. Thus, the robot must maintain a *belief* over the locations of entities and intentionally select actions that reduce its uncertainty about the world [7].

This class of problems can be formalized as a hybrid *partially observable Markov decision process* (POMDP) [8]. Solutions are *policies*, mappings from distributions over world states (*belief-states*) to actions. Because solving these problems exactly is intractable [8], we compute a policy *online* via repeatedly *replanning* [9], [10], each time solving an approximate, *determinized* [9], [10] version of the problem using an existing TAMP approach [11]. POMDP planning can be viewed as searching through *belief space*, the space of belief states, where perception and control actions operate on distributions over states instead of individual states.

Most prior work approximately models belief space using either discrete [12], [13], [14] or fluent-based [7], [15] *abstractions*. In contrast, we operate *directly* on hybrid belief distributions by specifying procedures that model observation sampling, visibility checking, and Bayesian belief filtering. This allows us to tackle problems where continuous and geometric components of the state govern the probability of an observation. For example, a movable object at a particular pose might occlude a goal object, substantially reducing the probability that it will be detected. By using a *particle-based* belief representation, we can model multi-modal beliefs that arise when several objects occlude regions of space. During planning, we conservatively approximate the probability of detection by *factoring* it into a conjunction over conditions on each individual object. This exposes *sparse* interactions between an observation and each object, enabling the planner to efficiently reason about and rectify occlusion.

Additionally, we introduce a replanning algorithm that uses past plans to *constrain* the structure of solutions for the current planning problem. This forces future plans to retain the structure of prior plans but allows some of the parameter values to change in response to stochastic execution or new observations. As a result, this ensures that the overall induced policy continues to make *progress* towards achieving the goal. Additionally, this reduces the search space of the planner and thus speeds up successive replanning invocations.

Finally, we introduce a mechanism that *defers* computing values for plan parameters that 1) are used temporally later in the plan, 2) almost always admit satisfying values, and 3)

[1]MIT CSAIL,{caelan,tlp,lpk}@csail.mit.edu
[2]NVIDIA,{cpaxton,dieterf}@nvidia.com
We gratefully acknowledge support from NSF grants 1523767 and 1723381; from AFOSR grant FA9550-17-1-0165; from ONR grant N00014-18-1-2847; from the Honda Research Institute; and from SUTD Temasek Laboratories. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

978-1-7281-7395-5/20/$31.00 ©2020 IEEE

are expensive to compute. Intuitively, this strategy performs the least amount of computation possible to obtain the next action and certify that it will make progress towards the goal. Finally, we evaluate our algorithms on several simulated tasks and demonstrate our system running on a real robot acting in a kitchen environment in the accompanying video.

II. Related Work

There is much work that addresses the problem of efficiently solving *deterministic*, *fully-observable* TAMP problems [1], [2], [3], [4], [5], [6]. However, only a few of these approaches have been extended to incorporate some level of stochasticity or partial observability [7], [15], [12].

Solving for an optimal, closed loop policy is undecidable in the infinite-horizon case [8], [16], even for discrete POMDPs. An alternative strategy is to dynamically compute a policy *online* in response to the current belief, rather than *offline* for all beliefs, by *replanning* [10]. One approach to online planning is to use Monte-Carlo sampling [17], [18] to explore likely outcomes of various actions. These methods have been successfully applied to robotic planning tasks such as grasping in clutter [19], non-prehensile rearrangement [20], and object search [21]. However, the hybrid action space in our application is too high-dimensional for uninformed action sampling to generate useful actions.

Another online planning strategy is to approximate the original stochastic problem as a deterministic problem through the process of *determinization* [9], [22], [10]. This enables deterministic planners, which are able to efficiently search large spaces, to be applied. *Most-likely outcome* determinization always assigns the action outcome that has the highest probability. When applied to observation actions, this approach is called *maximum likelihood observation* (MLO) determinization [23], [24], [15]. However, this approximation fails when the success of a policy depends on some outcome other than the mode of observation distribution.

There are many approaches for representing and updating a belief such as joint, unscented Kalman filtering [23], [7], *factoring* the belief into independent distributions per object [15], [25], and maintaining a particle filter, which represents the belief as a set of weighted samples [17], [18], [19], [21]. Many approaches use a different belief representation when planning versus when filtering. Several approaches plan on a purely discrete *abstraction* of the underlying hybrid problem [12], [13], [14]. Other approaches plan using a calculus defined on *belief fluents* [7], [15], logical tests on the underlying belief such as "the value of random variable X is within δ of value x with probability at least $1-\epsilon$". In contrast, our approach plans directly on probability distributions, where actions update beliefs via proper transition and observation updates.

III. Problem Definition

We address hybrid, belief-state *Stochastic Shortest Path Problems* (SSPP) [26], a subclass of hybrid POMDPs where the *cost* $c_a > 0$ of action a is strictly positive. The robot starts with a *prior* belief b_0. Its objective is to reach a goal set of beliefs B_* while minimizing the cost it incurs. The robot selects actions a according to a *policy* $a \sim \pi(b)$ defined on belief states b. We evaluate $\pi(b)$ *online* by *replanning* given the current belief state b. We approximate the original belief-space SSPP by *determinizing* its action outcomes (Section V). We formalize each determinized SSPP in the PDDLStream [11] language and solve them using a cost-minimizing PDDLStream planner.

Although our technique is general-purpose, our primary application is partially-observable TAMP in a kitchen environment that contains a single mobile manipulator, counters, cabinets, drawers, and a set of unique, known objects. The robot can observe the world using an RGB-D camera that is fixed to the world frame. The camera can *detect* the set of objects that are visible as well as noisily *estimate* their poses. The latent world state is given by the robot configuration, door and drawer joint angles, the discrete frame that each object is attached to, and the pose of each object relative to its attached frame. We maintain a *factored* belief as the product of independent posterior distributions over each variable. In our environment, the robot's configuration as well as the door and drawer joint angles can be accurately estimated using our perception system [27], so we only maintain a point estimate for these variables. However, there is substantial partial observability when estimating object poses due to occlusions from doors, drawers, other objects, and even the robot. We represent and update our belief over the pose state of each object using *particle filtering* (Fig. 3).

IV. PDDLStream Formulation

PDDLStream is an extension of Planning Domain Definition Language (PDDL) [28] that adds the ability to programmatically declare procedures for sampling values of continuous variables in the form of *streams*. Like PDDL, PDDLStream uses *predicate* logic to describe planning problems. An evaluation of a predicate for a given set of arguments is called a *literal*. A *fact* is a true literal. *Static* literals always remain constant, but *fluent* literals can change truth value as actions are applied. States are represented as a set of fluent literals. Our domain makes use of the following fluent predicates: `(AtConf ?r ?q)` states that robot part `?r` (the `base` or `arm`) is at configuration `?q`; `(AtAngle ?j ?a)` states that a door or drawer `?j` is at joint angle `?a`; `(HandEmpty)` indicates that the robot's end-effector is empty; and `(AtGrasp ?o ?g)` states that object `?o` is attached to the end-effector using grasp `?g`. Here, `?q`, `?a`, and `?g` are all real-valued high-dimensional parameters.

An *action schema* is specified by a set of free parameters (**:param**), a precondition formula (**:pre**) that must hold in a state in order to execute the action, and a conjunctive effect formula (**:eff**) that describes the changes to the state. Effect formulas may set a fluent fact to be true, set a fluent fact to be false (**not**), or increase the plan cost (**incr**) [29]. For example, consider the following action descriptions for `move` and `pick`. Other actions such as `place`, `push` door, `pull` door, `pour`, and `press` button can be defined similarly to `pick`. We used universally quantified *conditional effects* [30] (omitted here for clarity) to update the world poses of objects placed in drawers for `push` and `pull` actions.

```
(:action move
 :param (?r ?q1 ?t ?q2)
 :pre (and (Motion ?r ?q1 ?t ?q2) (AtConf ?r ?q1))
 :eff (and (AtConf ?r ?q2) (not (AtConf ?r ?q1))))
(:action pick
 :param (?o ?pb ?g ?bq ?aq)
 :pre (and (Kin ?o ?pb ?g ?bq ?aq) (AtPoseB ?o ?pb)
  (HandEmpty) (AtConf base ?bq) (AtConf arm ?aq))
 :eff (and (Holding ?o ?g)
  (not (AtPoseB ?o ?pb)) (not (HandEmpty))))
```

The novel representational aspect of PDDLStream is *streams*: functions from a set of *input* values (**:inp**) to a *generator* that enumerates a possibly infinitely-long sequence of *output* values (**:out**). Streams have a *declarative* component that specifies 1) arity of input and output values, 2) a *domain formula* (**:dom**) that governs legal inputs, and 3) a conjunctive *certified formula* (**:cert**) that expresses static facts that all input-output pairs are *guaranteed* to satisfy. Additionally, streams have a *programmatic* component that implements the function in a programming language such as Python. For example, the `inv-kin` stream takes in a tuple of values specifying an object `?o`, its pose `?pb`, a grasp `?g`, and a robot base configuration `?bq`. Using an inverse kinematics solver, it generates robot arm configurations `?aq` that satisfy the attachment kinematic relationship `Kin` involving the provided placement and grasp for the object. The `motion` stream uses a motion planner, such as RRT-Connect [31], to produce trajectories `?t` that certify the static `Motion` precondition of the `move` action.

```
(:stream inv-kin
 :inp (?o ?pb ?g ?bq)
 :dom (and (Conf base ?bq)
  (PoseB ?o ?pb) (Grasp ?o ?g)
 :out (?aq)
 :cert (and (Conf arm ?aq)
  (Kin ?o ?p ?g ?bq ?aq)))
```

```
(:stream motion
 :inp (?r ?q1 ?q2)
 :dom (and (Conf ?r ?q1)
       (Conf ?r ?q2))
 :out (?t)
 :cert (and (Traj ?r ?t)
  (Motion ?r ?q1 ?t ?q2)))
```

A. Modeling Observations

Our first contribution is applying PDDLStream [11] to model determinized, hybrid belief-state SSPPs. This allows us to plan using *domain-independent* PDDLStream algorithms, such as the *Focused* algorithm [11], without modification. We model the ability for the robot to perform a sensing action, receive an observation, and update its belief using the `detect` action. The `detect` action is parameterized by an object `?o`, a *prior pose belief* `?pb1`, an *observation* `?obs`, and a *posterior pose belief* `?pb2`. The fluent `(AtPoseB ?o ?pb)` states that object `?o` has current pose belief `?pb`. Critically, `?pb1` and `?pb2` represent *distributions* over real-valued poses. The `BeliefUpdate` precondition ensures these values represent a Bayesian update. If the observation `?obs` is not `BOccluded` by another object, `detect` updates the current pose belief for `?o`.

```
(:action detect
 :param (?o ?pb1 ?obs ?pb2)
 :pre (and (BeliefUpdate ?o ?pb1 ?obs ?pb2)
  (AtPoseB ?o ?pb1) (not (BOccluded ?o ?pb1 ?obs)))
 :eff (and (AtPoseB ?o ?pb2) (not (AtPoseB ?o ?pb1))
  (incr (total-cost) (ObsCost ?o ?pb1 ?obs))))
```

Our key representational insight is that we can encode the Bayesian filtering process by defining sampling and inference *streams* that operate on distributions. The `sample-obs` stream samples `?obs` from the distribution of observations using the *observation model* for `?o` and a pose belief `?pb`. The `test-vis` stream returns true if object `?o2` at belief `?pb2` prevents the robot from observing `?obs` with probability less than ϵ, a value described in Section V. The probability of occlusion is estimated by performing ray-casting along `?obs` using poses sampled from `?pb2`.

```
(:stream sample-obs
 :inp (?o ?pb)
 :dom (PoseB ?o ?pb)
 :out (?obs)
 :cert (Obs ?o ?obs))
```

```
(:stream test-vis
 :inp (?o ?obs ?o2 ?pb2)
 :dom (and (Obs ?o ?obs)
  (PoseB ?o2 ?pb2))
 :cert (BVis ?o ?obs ?o2 ?pb2))
```

The `update-belief` stream computes the posterior pose belief `?pb2` by performing a *Bayesian update* using the prior pose belief `?pb1` and hypothesized observation `?obs`. Note that although observations are stochastic, the belief update process given an observation is a deterministic function.

```
(:stream update-belief
 :inp (?o ?pb1 ?obs)
 :dom (and (PoseB ?o ?pb1) (Obs ?o ?obs))
 :out (?pb2)
 :cert (and (PoseB ?o ?pb2)
  (BeliefUpdate ?o ?pb1 ?obs ?pb2)))
```

Finally, we specify `BOccluded` as a derived predicate [32], [33], a fact that is logically inferred given the current state. `BOccluded` is true if there exists another object `?o2` currently at pose belief `?pb2` that prevents `?obs` from being received with probability exceeding ϵ.

```
(:derived (BOccluded ?o ?obs)
 (exists (?o2 ?pb2)
  (and (Obs ?o ?obs)  (AtPoseB ?o2 ?pb2)
   (not (BVis ?o ?obs ?o2 ?pb2)))))
```

V. Determinized Observation Costs

We are interested in enabling a deterministic planner to perform approximate probabilistic reasoning by minimizing plan costs. The maximum acceptable risk can always be specified using a user-provided maximum expected cost $c_* \in [0, \infty)$. We focus on computing `ObsCost`, the cost of `detect`, which is a function of the prior pose belief `?pb1` and the observation `?obs`. Similar analysis can be applied to other probabilistic conditions, such as collision checks.

```
(:function (ObsCost ?o ?pb ?obs)
 :dom (and (PoseB ?o ?pb) (Obs ?o ?obs)))
```

A. Self-Loop Determinization

The widely-used most-likely-outcome and all-outcome determinization schemes do not provide a natural way of integrating the cost c_a of action a and the probability of an intended outcome p_a [34], [7]. Thus, we instead use *self-loop* determinization [35], [10], which approximates the original SSPP as a simplified self-loop SSPP. In a self-loop SSPP, an action a executed from state s may result in only two possible states: a new state s' or the current state s. For this simple class of SSPPs, a planner can obtain an optimal policy by optimally solving a deterministic problem with *transformed* action costs. Let c'_a be the cost of a upon a failed (self-loop) transition. The determinized cost $\hat{c}_a$ of action a is then

$$\hat{c}_a \equiv c_a + \sum_{t=1}^{\infty} c'_a (1-p_a)^t = c_a + \Big(\frac{c'_a}{p_a} - c'_a\Big).$$

We directly model our domain as a self-loop SSPP by specifying an upper bound for expected cost of a successful outcome c_a, an upper bound for the expected *recovery* cost c'_a to return to s (*i.e.* the self-loop transition), and a lower bound for the probability of a successful outcome p_a.

B. Computing the Likelihood of an Observation.

Suppose there are n unique objects in the world, and we are interested in detecting object i. Let $\mathcal{X}_j$ be the latent continuous pose random variable for an object j, and let x_j be a value of $\mathcal{X}_j$. As shorthand, define $\bar{\mathcal{X}}_{-i}$ to be a tuple of latent poses for each of the n objects *except* for object i. Let $P(\mathcal{X}_i)$ be a probability density over $\mathcal{X}_i$, which in our application, is represented by a set of weighted particles. Let $\mathcal{Z}_i^v$ and $\mathcal{Z}_i^d$ be observed Bernoulli random variables for whether object i is *visible* and is *detected*. When $\mathcal{Z}_i^d$ is true, let $\mathcal{Z}_i^p$ be a continuous random variable for the observed pose of object i. Otherwise, $\mathcal{Z}_i^p$ is undefined. For detection, we will assume that $P\left(\mathcal{Z}_i^d{=}1 \mid \mathcal{Z}_i^v{=}1\right) = 1 - p_{\text{FN}}$ where p_{FN} is the probability of a false negative. We will conservatively use zero as a lower bound for the probability of a false positive, *i.e.* $P\left(\mathcal{Z}_i^d{=}1 \mid \mathcal{Z}_i^v{=}0\right) \geq 0$, which removes false detection terms. For pose observations, we will assume a multivariate Gaussian noise model $\mathcal{Z}_i^p \mid \left(\mathcal{Z}_i^d{=}1, \mathcal{X}_i{=}x_i\right) \sim \mathcal{N}(x_i, \Sigma_i)$. We are interested in $P(\mathcal{Z}_i^p)$, the probability of receiving a pose observation for object i.

$$P(\mathcal{Z}_i^p) = \int_{x_i} P\left(\mathcal{Z}_i^p \mid \mathcal{Z}_i^d, x_i\right) P\left(\mathcal{Z}_i^d \mid \mathcal{Z}_i^v\right) P(\mathcal{Z}_i^v \mid x_i)\, dP(x_i)$$

The key component of this expression is $P(\mathcal{Z}_i^v \mid x_i)$, the probability that x_i is currently visible, which is contingent on the poses of the other objects $\bar{\mathcal{X}}_{-i}$. Define $B_j^i(x_i, x_j)$ as a deterministic function that is 1 if object j at pose x_j *blocks* object i from being visible at pose x_i and otherwise is 0. Ultimately, $P(\mathcal{Z}_i^v \mid x_i)$ will be a component of the cost function `ObsCost` and thus must only depend on pose belief `?pbl` and observation `?obs`. However, it is currently still dependent on the current beliefs for each of the other $n{-}1$ objects all at once. While we could instead parameterize `ObsCost` using the pose belief of all objects, it would be combinatorially difficult to instantiate as n increases. And due to its *unfactored* form, we would not be able to benefit from efficient deterministic search strategies that leverage factoring. Thus, we marginalize out x_i, which ties the $n-1$ objects together, by taking the worst-case probability of visibility $L_j(X_i)$ due to object j over a subset of states X_i.

$$L_j(X_i) = \inf_{x_i \in X_i} \int_{x_j} \big(1 - B_j^i(x_i, x_j)\big) dP(x_j)$$

As a result, we can provide a non-trivial lower bound for $P(\mathcal{Z}_i^v \mid x_i)$ that no longer depends on x_i. Suppose there exists $\epsilon \in [0,1)$ that satisfies $\min_{j \neq i} L_j(X_i) \geq \epsilon$, then

$$\begin{aligned} P(\mathcal{Z}_i^v \mid x_i) &= \int_{\bar{x}_{-i}} (1 - B_{-i}^i(x_i, \bar{x}_{-i})) dP(\bar{x}_{-i}) \\ &\geq \prod_{j \neq i}^{n} \int_{x_j} \big(1 - B_j^i(x_i, x_j)\big) dP(x_j) \qquad (1) \\ &\geq \big(\min_{j \neq i} L_j(X_i)\big)^{n-1} \geq (1-\epsilon)^{n-1}. \end{aligned}$$

Inequality 1 follows from the fact that some combinations of $\bar{x}_{-i}$ would result in object collision and thus are not possible. Finally, this gives us the following lower bound for p_a:

$$\begin{aligned} P(\mathcal{Z}_i^p) &\geq P(\mathcal{Z}_i^p, \mathcal{X}_i \in X_i) \\ &\geq p_{\text{FN}} P\left(\mathcal{Z}_i^p, \mathcal{X}_i \in X_i \mid \mathcal{Z}_i^d\right)(1-\epsilon)^{n-1} \qquad (2) \end{aligned}$$

This probability depends on both X_i and ϵ. Ideally, we would select X_i and ϵ that maximize equation 2; however, this requires operating on all n objects at once. Instead, we let the planner select `?obs` $= (z_i^p, X_i, \epsilon)$ by sampling different values of `?obs`. However, `detect` can only be applied at this cost if $\forall j \neq i,\ L_j(X_i) \geq 1 - \epsilon$, which is enforced using the `BOccluded` derived predicate (section IV-A) by quantifying over the `BVis` condition for each object. The choice of X_i presents a trade off because the prior probability $P(X_i)$ increases as X_i grows but each $L_j(X_i)$ decreases. In practice, we sample $x_i \sim \mathcal{X}_i$ and take $X_i = \{x'_i \in B_\delta(x_i) \mid 0 < P(x'_i)\}$ to be a δ-neighborhood of x_i, capturing a local region where we anticipate observing object i.

C. Observation Example

Consider the scenario in Fig. 2 with objects `A`, `B`, `C`, and `D`. Suppose that the object poses for `A`, `B`, and `C` are perfectly known, but object `D` is equally believed to be either at pose $x_{\mathtt{D}}^1$ or $x_{\mathtt{D}}^3$ (but not $x_{\mathtt{D}}^2$). First, note that $L_j(X_i) = 0$ for all choices of X_i because object `A` occludes $x_{\mathtt{D}}^1$, object `B` occludes $x_{\mathtt{D}}^2$, and object `C` occludes $x_{\mathtt{D}}^3$, all with probability 1. If we take $X_{\mathtt{D}} = \{x_{\mathtt{D}}^1, x_{\mathtt{D}}^2, x_{\mathtt{D}}^3\}$, then $L_{\mathtt{A}}(X_{\mathtt{D}}) = L_{\mathtt{B}}(X_{\mathtt{D}}) = L_{\mathtt{C}}(X_{\mathtt{D}}) = 1$, meaning all three objects must be moved before applying `detect`, despite the fact that $P\left(x_{\mathtt{D}}^2\right) = 0$. If we take $X_{\mathtt{D}} = \{x_{\mathtt{D}}^1, x_{\mathtt{D}}^3\}$ then $L_{\mathtt{A}}(X_{\mathtt{A}}) = L_{\mathtt{C}}(X_{\mathtt{D}}) = 1$ but $L_{\mathtt{B}}(X_{\mathtt{A}}) = 0$, indicating that `B` does not need to be moved. Finally, if we take $X_{\mathtt{D}} = \{x'_{\mathtt{D}}\}$ then only $L_{\mathtt{A}}(X_{\mathtt{A}}) = 1$, and only `A` must be moved. Intuitively, this shows that selecting X_i to be a small, local region improves sparsity with respect to which objects affect a particular observation under our bound.

VI. ONLINE REPLANNING

Now that we have incorporated probabilistic reasoning into a deterministic planner, we induce a policy π by replanning after executing each action a. However, done naively, it is possible to result in a policy that never reaches the goal set of beliefs B_*. This is even true when acting in a *deterministic problem* using replanning. For example, consider a deterministic, observable planning problem where the goal is for the robot to hold object `D`. The robot's first plan might require moving its `base` (`b`), moving its `arm` (`a`), and finally picking object `D`: [$\mathtt{move}(\mathtt{base}, q_{\mathtt{b}}^0, t_{\mathtt{b}}^1, q_{\mathtt{b}}^1)$, $\mathtt{move}(\mathtt{arm}, q_{\mathtt{a}}^0, t_{\mathtt{a}}^1, q_{\mathtt{a}}^1)$,

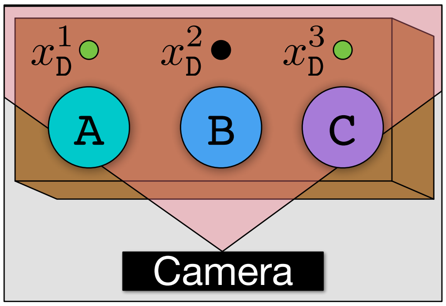

Fig. 2: An example detection scenario where object D is believed to be either behind object A or object C with equal probability.

pick$(\mathtt{D}, p^0_{\mathtt{D}}, g_{\mathtt{D}}, q^1_{\mathtt{b}}, q^1_{\mathtt{a}})$]. Suppose the robot executes the first move action, arrives as expected at base configuration $q^1_{\mathtt{b}}$, but replans to obtain a new plan: [move(base, $q^1_{\mathtt{b}}, t^2_{\mathtt{b}}, q^2_{\mathtt{b}}$), move(arm, $q^0_{\mathtt{a}}, t^2_{\mathtt{a}}, q^2_{\mathtt{a}}$), pick($\mathtt{D}, p^0_{\mathtt{D}}, g_{\mathtt{D}}, q^2_{\mathtt{b}}, q^2_{\mathtt{a}}$)].

While this is a satisfactory, despite suboptimal, solution when solving for a single plan in isolation, it is undesirable in the context of the previous plan because it requires another base movement action despite the robot having just executed one. This process could repeat indefinitely, causing the robot to *never* achieve its goal despite never failing to find a plan. For a deterministic problem, this is easily preventable by simply executing the first plan all at once. However, in a stochastic environment where, for example, base movements are imprecise, executing the full plan open loop will almost always fail. Thus, we must replan using the base pose $\hat{q}^1_{\mathtt{b}}$ that we actually reach instead of $q^1_{\mathtt{b}}$, the one we intended to reach. This at least requires planning new values for anything that was a function of $q^1_{\mathtt{b}}$, such as $q^1_{\mathtt{a}}$, which was sampled by the inv-kin($\mathtt{D}, p^0_{\mathtt{D}}, g_{\mathtt{D}}, q^1_{\mathtt{b}}$) stream.

Intuitively, we need to enforce that some amount of overall *progress* is retained when replanning. One way to do this is to impose a decreasing *constraint* on the length of future plans. This constraint could be that the next plan must have fewer actions than the previous plan. If actions always have positive probability of successful execution, for example if the domain is dead-end free, then this strategy will achieve the goal B_* with probability 1. While this strategy ensures that the robot almost certainly reaches the goal, it incurs a significant computational cost because the robot plans from scratch on each iteration, wasting previous search effort.

A. Reuse-Enforced Replanning

Although some of the values in the previous plan are no longer viable due to stochasticity, the plan's overall structure might still be correct. Thus, one way to speed up each search is to additionally constrain the next plan to adhere to the same *structure* as the previous plan. To do this, we first identify all action arguments that are *constants*, meaning that they are valid quantities in subsequent problems. These include the names of objects and grasps for objects but not poses, configurations, or trajectories, which are conditioned on the most recent observations of the world. We replace each use of a non-constant with a unique free variable symbol (denoted by the prefix @). For the previous example, this produces

Algorithm 1 Online Replanning Policy

procedure POLICY(b, B_*, c_*)
  $f_{prev}, \vec{a}_{prev} \leftarrow \emptyset,$ **None**
  while True do
    $A, s, S_* \leftarrow$ DETERMINIZE(b, B_*)
    $\vec{a} \leftarrow$ **None**
    if $\vec{a}_{prev} \neq$ **None then** ▷ Reuse plan constraints
      $A', S'_* \leftarrow$ CONSTRAINPLAN$(\vec{a}_{prev}, S_*)$
      $f, \vec{a} \leftarrow$ PLAN$(A', s \cup f_{prev}, S'_*, c_*)$
    if $\vec{a} =$ **None then** ▷ No plan constraints
      $f, \vec{a} \leftarrow$ PLAN(A, s, S_*, c_*)
    if $\vec{a} \neq$ **None then** ▷ No plan with cost below c_*
      return False
    if $\vec{a} = [\,]$ **then** ▷ Reached goal belief
      return True
    $o \leftarrow$ EXECUTEACTION(a_1) ▷ Receive observation o
    $b \leftarrow$ UPDATEBELIEF(a_1, o)
    $f_{prev}, \vec{a}_{prev} \leftarrow$ CONSTANTFACTS$(f), \vec{a}_{2:|\vec{a}|}$

the following plan structure, which is used to constrain the second replanning effort: [move(arm, @aq1, @at1, @aq2), pick(D, @p1, $g_{\mathtt{D}}$, @bq1, @aq2)]. Thus, the planner does not need to search over sequences of actions schemata, objects to manipulate, or grasps because these decisions are fixed.

Algorithm 1 gives the pseudocode for our online replanning policy. The inputs to POLICY are the prior belief b, goal set of beliefs B_*, and maximum cost c_*. POLICY maintains a set of previously proven static facts f_{prev} as well as the tail of the previous plan $\vec{a}_{prev}$. On each iteration, first, the procedure DETERMINIZE models the belief SSPP as a deterministic planning problem with actions A, initial state s, and goal set of states S_*. If the prior plan $\vec{a}_{prev}$ exists, POLICY applies the plan constraints using the CONSTRAINPLAN procedure described in algorithm 2. If the PDDLStream planner PLAN is unable to solve the constrained problem within a user-provided timeout, the constraints are removed, and planning is *reattempted*. If successful, PLAN returns not only a plan $\vec{a}$ but also the certified facts f within the *preimage* of $\vec{a}$ that prove that $\vec{a}$ is a solution. Then, POLICY executes a_1, the first action of $\vec{a}$, receives an observation o, and updates its current belief b. Finally, it extracts the subset of constant facts in f, static facts that only involve constants, and sets $\vec{a}_{prev}$ to be remainder of $\vec{a}$ that was not executed.

Algorithm 2 gives the pseudocode for the constraint transformation. It creates a new set of action schemata A', each of which have modified preconditions and effects, using the previous plan $\vec{a}$. The fact (Applied i) is a total-ordering constraint that enforces that action a_{i-1} be applied before action a_i. For each argument v of action a_i, if v is a constant, the new action is forced to use the same value. The fact (Bound v) is true if symbol v has already been assigned to some value in the action sequence. If (Bound v) is true, the fact (Assigned v ?p) is true if free variable v has been assigned to new value ?p. Each free variable v must either be unbound or assigned to action argument ?p.

VII. Deferred Stream Evaluation

We use the *Focused* algorithm [11] to solve each determinized PDDLStream problem. The *Focused* algorithm

Algorithm 2 Plan Constraint Compilation

```
procedure ConstrainPlan(a⃗, S*)
    A ← ∅
    for a_i ∈ a⃗ = [a_1, a_2, ...] do
        if 2 ≤ i then                               ▷ Total ordering constraint
            a_i.pre ← (and a_i.pre (Applied i − 1))
        for v, ?p ∈ zip(a_i.args, a_i.param) do
            if IsConstant(v) then                   ▷ Enforce fixed value ?p = v
                a_i.pre ← (and a_i.pre (= v ?p))
            else                                    ▷ Otherwise, v is a free variable
                f_p ← (imply (Bound v) (Assigned v ?p))
                a_i.pre ← (and a_i.pre f_p)
                f_e ← (and (Bound v) (Assigned v ?p))
                a_i.eff ← (and a_i.eff (Applied i) f_e)
        A ← A ∪ {a_i}
    S* ← (and S* (Applied |a⃗|))
    return A, S*
```

optimistically plans using hypothetical stream output values before actually calling any stream procedures. As a result, it not only generates candidate action plans but also *stream plans*, which consist of a sequence of scheduled stream queries that could bind the hypothetical values (**bolded**) in the action plan. For example, consider the following possible stream plan for the action plan in section VI.

$$[\texttt{grasps}(\texttt{D})\rightarrow \boldsymbol{g}_{\texttt{D}}, \texttt{inv-reach}(\texttt{D}, p_{\texttt{D}}^0, \boldsymbol{g}_{\texttt{D}})\rightarrow \boldsymbol{q}_{\texttt{b}}^{\mathbf{1}}, \texttt{inv-kin}(\texttt{D}, p_{\texttt{D}}^0, \boldsymbol{g}_{\texttt{D}}, \boldsymbol{q}_{\texttt{b}}^{\mathbf{1}})\rightarrow \boldsymbol{q}_{\texttt{a}}^{\mathbf{1}}, \texttt{motion}(\texttt{base}, q_{\texttt{b}}^0, \boldsymbol{q}_{\texttt{b}}^{\mathbf{1}})\rightarrow \boldsymbol{t}_{\texttt{b}}^{\mathbf{1}}, \texttt{motion}(\texttt{arm}, q_{\texttt{a}}^0, \boldsymbol{q}_{\texttt{a}}^{\mathbf{1}})\rightarrow \boldsymbol{t}_{\texttt{a}}^{\mathbf{1}}]. \quad (3)$$

The *Focused* algorithm will not terminate until it has successfully bound all hypothetical values. As a result, it will recompute the `motion` stream for every `move` action on its plan per replanning invocation, spending a significant amount of computation constructing trajectories that will never be executed. To avoid this, we *defer* the evaluation of expensive streams if they are not required before we next replan. For the example in equation 3, the `inv-kin` and `motion(arm,...)` streams could both be deferred because they are first used within the `move(arm, ...)` action, rather than the `move(base, ...)` action. Still, the stream plan in equation 3 prematurely evaluates `inv-kin`. Thus, we *reschedule* the stream plan by identifying streams that must be queried to perform the first action or that should never be deferred. We then *recursively* propagate this criterion for streams that are required by the aforementioned streams. Finally, we query all of the identified streams, which are `grasps`, `inv-reach`, and `motion(base, ...)` and defer the rest: `inv-kin`, `motion(arm, ...)`.

It is not always advantageous to defer streams. For instance, the initial pose $p_{\texttt{D}}^0$, sampled grasp $g_{\texttt{D}}$, and sampled base configuration $q_{\texttt{b}}^1$ might not admit a kinematic solution (`Kin`), which is required to perform `pick`. Rather than move to $q_{\texttt{b}}^1$ before discovering this, it is less costly to infer this at the start and sample new values for $g_{\texttt{D}}$ or $q_{\texttt{b}}^1$. Thus, we only defer the evaluation of streams that are both *likely* to succeed and computationally *expensive*. In our domain, we only defer the `motion` stream as it almost always succeeds as long as the the initial and final configurations are not in collision.

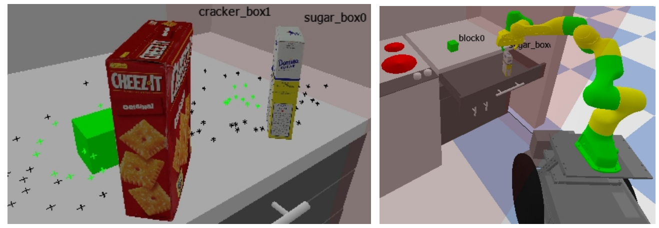

Fig. 3: **Left**: the particle-filter pose belief for the green block after one observation. Green particles have high weight and black particles have low weight. **Right**: in the *Stow* task, the robot must remove the sugar to place the block and close the drawer.

VIII. Experiments

We experimented on 25 randomly generated problems within 4 simulated domains. We used PyBullet [36] for ray-casting and collision checking and IKFast [37] for inverse kinematics. See `https://github.com/caelan/SS-Replan` for our open-source Python implementation. We experimented with 4 versions of our system: one per combination of using plan *constraints* and/or *deferred* streams. Each policy was limited to 10 minutes of total planning time.

For the *Swap* task, the block starts in one drawer, but the goal is to believe it is in the other drawer. The robot's pose prior is uniform over both drawers. Successful policies typically inspect the goal drawer, fail to observe it, and then look for it in the other drawer. This requires placing the block in an intermediate location to close one drawer and open the other. See the extended version of this paper `https://arxiv.org/abs/1911.04577` for descriptions of the other tasks. Table I shows the results of the experiments. Applying *both* plan constraints and deferring streams results in a large improvement in the success rate and generally a reduction in total planning time while executing the policy.

Alg:	Neither		Constraints		Deferred		Both	
Task:	%	t	%	t	%	t	%	t
Inspect	96	41	**100**	45	**100**	32	**100**	**26**
Stow	80	124	**88**	226	**88**	**92**	**88**	108
Swap	52	190	40	484	60	**105**	**80**	232
Cook	20	375	40	406	56	249	**100**	**203**

TABLE I: The success rate (%) and mean total planning time for *successful* trials in seconds (t) over 25 generated problems per task.

We applied our system to real-world kitchen manipulation tasks performed by a Franka Emika Panda robot arm. The full perception system is described in prior work [38]. See `https://youtu.be/IOtrO29DFUg` for a video of the robot solving the the *Inspect Drawer*, *Swap Drawers*, and *Cook Spam* tasks, which are similar to those in table I.

IX. Conclusions

We presented a replanning system for acting in partially-observable domains. By planning directly on beliefs, the planner can approximately compute the likelihood of detection given each movable object pose belief. Through plan structure constraints, we ensure our replanning policy makes progress towards the goal. And by deferring expensive stream evaluations, we enable replanning to be performed efficiently.

REFERENCES

[1] 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.

[2] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical Planning in the Now," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011. [Online]. Available: http://ieeexplore.ieee.org/document/5980391/

[3] 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," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014. [Online]. Available: http://ieeexplore.ieee.org/document/6906922/

[4] M. Toussaint, "Logic-geometric programming: an optimization-based approach to combined task and motion planning," in *International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI Press, 2015, pp. 1930–1936. [Online]. Available: https://www.ijcai.org/Proceedings/15/Papers/274.pdf

[5] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "FFRob: Leveraging symbolic planning for efficient task and motion planning," *International Journal of Robotics Research*, vol. 37, no. 1, pp. 104–136, 2018. [Online]. Available: https://arxiv.org/abs/1608.01335

[6] ——, "Sampling-based methods for factored task and motion planning," in *The International Journal of Robotics Research*, vol. 13, no. 13-14, 2017. [Online]. Available: https://arxiv.org/abs/1801.00680

[7] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *International Journal of Robotics Research (IJRR)*, 2013.

[8] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[9] S. W. Yoon, A. Fern, and R. Givan, "FF-Replan: A Baseline for Probabilistic Planning," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2007.

[10] A. Kolobov, "Planning with Markov decision processes: An AI perspective," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 6, no. 1, pp. 1–210, 2012.

[11] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "PDDLStream: Integrating Symbolic Planners and Blackbox Samplers," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2020. [Online]. Available: https://arxiv.org/abs/1802.08705

[12] C. Phiquepal and M. Toussaint, "Combined task and motion planning under partial observability: An optimization-based approach," in *RSS Workshop on Integrated Task and Motion Planning*, 2017.

[13] Y. Wang, S. Chaudhuri, and L. E. Kavraki, "Bounded policy synthesis for POMDPs with safe-reachability objectives," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 238–246.

[14] S. Srivastava, N. Desai, R. Freedman, and S. Zilberstein, "An anytime algorithm for task and motion mdps," *arXiv preprint arXiv:1802.05835*, 2018.

[15] D. Hadfield-Menell, E. Groshev, R. Chitnis, and P. Abbeel, "Modular task and motion planning in belief space," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 4991–4998.

[16] O. Madani, S. Hanks, and A. Condon, "On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems," in *AAAI/IAAI*, 1999, pp. 541–548.

[17] D. Silver and J. Veness, "Monte-Carlo planning in large POMDPs," in *Advances in neural information processing systems*, 2010, pp. 2164–2172.

[20] J. E. King, V. Ranganeni, and S. S. Srinivasa, "Unobservable monte carlo planning for nonprehensile rearrangement tasks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 4681–4688.

[18] A. Somani, N. Ye, D. Hsu, and W. S. Lee, "DESPOT: Online POMDP planning with regularization," in *Advances in neural information processing systems*, 2013, pp. 1772–1780.

[19] J. K. Li, D. Hsu, and W. S. Lee, "Act to see and see to act: POMDP planning for objects search in clutter," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 5701–5707.

[21] Y. Xiao, S. Katt, A. ten Pas, S. Chen, and C. Amato, "Online Planning for Target Object Search in Clutter under Partial Observability," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8241–8247.

[22] S. W. Yoon, A. Fern, R. Givan, and S. Kambhampati, "Probabilistic Planning via Determinization in Hindsight." in *AAAI*, 2008, pp. 1010–1016.

[23] R. Platt, R. Tedrake, L. Kaelbling, and T. Lozano-Perez, "Belief space planning assuming maximum likelihood observations," *Robotics: Science and Systems VI*, 2010. [Online]. Available: http://www.roboticsproceedings.org/rss06/p37.pdf

[24] R. Platt Jr and R. Tedrake, "Non-Gaussian belief space planning as a convex program," in *Proc. of the IEEE Conference on Robotics and Automation*, 2012.

[25] L. P. Kaelbling and T. Lozano-Pérez, "Implicit belief-space pre-images for hierarchical planning and execution," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 5455–5462.

[26] D. P. Bertsekas and J. N. Tsitsiklis, "An analysis of stochastic shortest path problems," *Mathematics of Operations Research*, vol. 16, no. 3, pp. 580–595, 1991.

[27] T. Schmidt, R. A. Newcombe, and D. Fox, "DART: Dense Articulated Real-Time Tracking." in *Robotics: Science and Systems*, vol. 2, no. 1. Berkeley, CA, 2014.

[28] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL: The Planning Domain Definition Language," Yale Center for Computational Vision and Control, Tech. Rep., 1998. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.9941

[29] M. Fox and D. Long, "PDDL2.1: An extension to PDDL for expressing temporal planning domains," *Journal of Artificial Intelligence Research (JAIR)*, vol. 20, p. 2003, 2003.

[30] E. P. D. Pednault, "ADL: Exploring the Middle Ground Between STRIPS and the Situation Calculus." *Kr*, vol. 89, pp. 324–332, 1989.

[31] J. J. Kuffner Jr. and S. M. LaValle, "RRT-Connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[32] S. Edelkamp, "PDDL2.2: The language for the classical part of the 4th international planning competition," *4th International Planning Competition (IPC'04), at ICAPS'04.*, 2004. [Online]. Available: https://pdfs.semanticscholar.org/4b3c/0706d2673d817cc7c33e580858e65b134ba2.pdf

[33] S. Thiébaux, J. Hoffmann, and B. Nebel, "In defense of PDDL axioms," *Artificial Intelligence*, vol. 168, no. 1-2, pp. 38–69, 2005.

[34] A. L. Blum and J. C. Langford, "Probabilistic planning in the graphplan framework," in *European Conference on Planning*. Springer, 1999, pp. 319–332.

[35] E. Keyder and H. Geffner, "The HMDPP planner for planning with probabilities," *Sixth International Planning Competition at ICAPS*, vol. 8, 2008.

[36] E. Coumans and Y. Bai, "PyBullet, a Python module for physics simulation for games, robotics and machine learning," 2016. [Online]. Available: http://pybullet.org

[37] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation, Robotics Institute, Carnegie Mellon University, 2010.

[38] C. Paxton, N. Ratliff, C. Eppner, and D. Fox, "Representing Robot Task Plans as Robust Logical-Dynamical Systems," *arXiv preprint arXiv:1908.01896*, 2019.