Combined Task and Motion Planning Through an Extensible Planner-Independent Interface Layer

Siddharth Srivastava Eugene Fang Lorenzo Riano Rohan Chitnis Stuart Russell Pieter Abbeel

***Abstract*— The need for combined task and motion planning in robotics is well understood. Solutions to this problem have typically relied on special purpose, integrated implementations of task planning and motion planning algorithms. We propose a new approach that uses off-the-shelf task planners and motion planners and makes no assumptions about their implementation. Doing so enables our approach to directly build on, and benefit from, the vast literature and latest advances in task planning and motion planning. It uses a novel representational abstraction and requires only that failures in computing a motion plan for a high-level action be identifiable and expressible in the form of logical predicates at the task level. We evaluate the approach and illustrate its robustness through a number of experiments using a state-of-the-art robotics simulator and a PR2 robot. These experiments show the system accomplishing a diverse set of challenging tasks such as taking advantage of a tray when laying out a table for dinner and picking objects from cluttered environments where other objects need to be re-arranged before the target object can be reached.**

I. Introduction

In order to achieve high-level goals like laying out a table, robots need to be able to carry out high-level task planning in conjunction with low-level motion planning. Task planning is needed to determine long-term strategies such as whether or not to use a tray to transport multiple objects, and motion planning is required for computing the actual movements that the robot should carry out. However, combining task planners and motion planners is a hard problem because task planning descriptions typically ignore the geometric preconditions of physical actions. In reality, even simple high-level actions such as picking up an object have continuous arguments, geometric preconditions and effects. As a result, the approach of generating a sequence of tasks and then doing motion planning for each task fails.

The main contribution of this paper is an approach that provides an interface between task and motion planning (with fairly minimal assumptions on each planning layer), such that the task planner can effectively operate in an abstracted state space that ignores geometry. Geometric constraints discovered through reasoning in the continuous state space are translated and communicated to the task planner through our interface layer (Fig. 1).

We introduce the main ideas through a tiny example in $\mathbb{R}^2$ (Fig. 2). In this problem, a gripper can pick up a block if it is adjacent to the block and aligned with one of its sides; it can place a block that it is currently holding by moving to a target location and releasing it. The goal is to pick up block b_1. A discrete planning specification for this problem would

Computer Science Division, University of California, Berkeley CA 94720

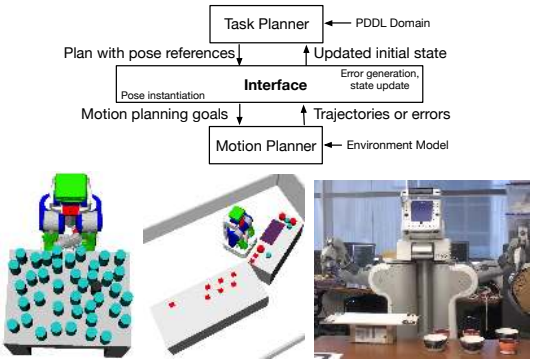


Fig. 1: Top: Outline of our approach. Bottom: Example test scenarios—(L) A cluttered table where the dark object has to be picked (there is no designated free space); (M) A dinner layout task where a tray is available but not necessary for transportation; (R) The PR2 starting a dinner layout.

describe actions *pick* and *place* with the following preconditions and effects: if the gripper is empty and $pick(b_1)$ is applied, the gripper holds b_1; if the gripper is holding b_1 and $place(b_1, S)$ is applied, the gripper no longer holds b_1 and b_1 is placed in the region S. However, this description is clearly inadequate because b_2 obstructs all trajectories to b_1 in the state depicted in Fig. 2, and $pick(b_1)$ cannot be executed. An accurate representation for this domain needs to include the geometric locations of objects and the gripper. The *pick* action's true arguments include *targetPose* and *traj*, denoting the target pose where picking should be done, and the trajectory along which the gripper should move to get to *targetPose*. The preconditions for picking b_1 require that *targetPose* be valid a gripping pose for b_1 and there be no obstruction in *traj*.

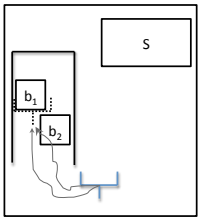


Fig. 2: Running example in $\mathcal{R}^2$: the gripper needs to pick b_1 after moving to the dotted pose.

Task planning domain descriptions require actions with discrete arguments and thus cannot directly handle this new representation. Discretizing the continuous variables for the high-level task planner is impractical, as even crude discretizations of the domains of the continuous variables quickly lead to computationally impractical problems.[1]

Our main contribution is an approach for communicating

[1]In a 2D world with only 10 sampled points along each axis, 5 objects, and considering only Manhattan paths that don't loop over themselves, we would need to precompute the truth values of close to $50,000$ "obstructs" predicates for the initial state. A similar discretization for one arm and the base for a PR2 robot would require $\sim 10^{11}$ facts. Even if this is done, the resulting problem instances will be too large to solve efficiently.

relevant geometric information to the task planner in terms of logical predicates, without discretization. In order to accomplish this we use *symbolic references* to continuous values [1], such as "grasping pose for b_1" and "trajectory for reaching grasping pose for b_1". We use an off-the-shelf task planner to produce plans of the form: "execute *pick* with a target pose which is a grasping pose for b_1 and has a feasible motion plan." Each of the symbols used in such plans needs to be instantiated, or refined into numbers. The interface layer does this by iterating through possible values for "grasping pose for b_1" and invoking the motion planner to compute a plan corresponding to each of them, thereby instantiating the trajectory reference (this constitutes the plan *refinement* process). Often however, a feasible instantiation may not exist. For instance, since b_2 is obstructing the gripper's path, a task plan of picking b_1 followed by placing it in the region S has no feasible refinement. Such failures can occur only if the geometric preconditions for a high-level action such as *pick* were false when it was attempted. These preconditions often concern the absence of obstructions, but may also refer to torque limits, stability properties of assemblies, etc. The key challenge is that the task planner cannot compute the truth values of such properties and the effects of actions on them. This is a natural consequence of using a representation suitable for task planners, and our approach is designed to handle it: it initializes the truth values of such properties to a set of default values and updates them if needed, during the plan refinement and generation process.

When a feasible instantiation is not found, the interface layer iterates through possible instantiations for the pose references. For each instantiation, it determines the conditions preventing a motion plan. Depending on the capabilities of the motion planner, a motion plan could also be selected so as to minimize these errors. For instance, the interface layer identifies b_2 as an obstruction error by removing all movable objects, invoking the motion planner, and identifying the collisions in the obtained trajectory. It translates this information into symbolic form, e.g. "b_2 obstructs trajectory for reaching grasping pose for b_1" and updates the task level state with it. A task planner is used to find a new plan for the updated state, e.g. "execute *pick* with a target pose from where b_2 can be grasped; execute place with a target pose from where releasing b_2 will place it on S; execute *pick* with a target pose from where b_1 can be grasped." For this plan the interface layer will find an instantiation of the continuous variables for which the motion planner can find a feasible motion plan and we are done. Effectively, our approach specifies a search problem for the interface.

The following sections formalize our algorithm and describe experiments on a number of tasks, including laying out a dinner table, which has millions of discrete states, and picking objects while replacing several obstructions on a tightly cluttered table. Videos of the experiments are available at:

`http://www.cs.berkeley.edu/%7Esiddharth/icra14.`

II. Background

A. Task Planning

The formal language PDDL [2] defines a fully observable, deterministic task planning problem as a tuple $\langle A, s_0, g\rangle$, where A is a set of parameterized propositional actions defined by preconditions and effects, s_0 is an initial state of the domain, and g, a set of propositions, is the goal condition. For clarity, we will describe both preconditions and effects of actions as conjunctive lists of literals in first-order logic, using quantifiers for brevity. The discrete *pick* action could be represented as follows:

pick(obj, gripper)
precon *Empty(gripper)*
effect *InGripper(obj), ¬Empty(gripper)*

A sequence of actions $a_0, \ldots, a_n$ executed beginning in s_0 generates a state sequence $s_1, \ldots, s_{n+1}$ where $s_{i+1} = a_i(s_i)$ is the result of executing a_i in s_i. The action sequence is a *solution* if s_i satisfies the preconditions of a_i for $i = 0, \ldots, n$ and s_{n+1} satisfies g.

B. Motion Planning

A motion planning problem is a tuple $\langle C, f, p_0, p_t\rangle$, where C is the space of possible configurations or poses of a robot, f is a Boolean function that determines whether or not a pose is in collision and $p_0, p_t \in C$ are the initial and final poses. A *collision-free motion plan* solving a motion planning problem is a trajectory in C from p_0 to p_t such that f doesn't hold for any pose in the trajectory. Motion planning algorithms use a variety of approaches for representing C and f efficiently. Throughout this paper, we will use the term *motion plan* to denote a trajectory that may include collisions. In some tasks, we may be interested in finding motion plans that ignore all of the movable obstacles. A solution that allows collisions only with a given set of movable objects in an environment may be obtained by invoking a motion planner by modifying f to be false for all collisions with such objects.

III. Abstract Formulation Using Pose References

Although high-level specifications like *pick* above capture the logical preconditions of physical actions they cannot be used in real pick-and-place tasks. A more complete representation of the *pick* action can be written with predicates *IsGP*, *IsMP* and *Obstructs* that capture geometric conditions: *IsGP(p, o)* holds iff p is a pose at which o can be grasped; *IsMP(traj, p_1, p_2)* holds iff *traj* is a motion plan from p_1 to p_2; *Obstructs(obj', traj,* obj*)* holds iff *obj'* is one of the objects obstructing a pickup of *obj* along *traj*. The argument *obj* need not be an argument in *Obstructs*; we include it for clarity.

pick2D(obj, gripper, pose$_1$, pose$_2$, traj)
precon *Empty(gripper), At(gripper, pose$_1$),*
IsGP(pose$_2$, obj), IsMP(traj, pose$_1$, pose$_2$),
∀obj'¬ *Obstructs(obj', traj, obj)*
effect *In(obj, gripper), ¬Empty(gripper),*
At(gripper, pose$_2$)

As noted in the Introduction, we adopt an abstract representation in which the "continuous" arguments—*pose*$_1$, *pose*$_2$,

and *traj* in this case—range not over the reals but over finite sets of *symbolic references* to continuous values.

We propose an abstract representation where continuous variables are replaced by ones that range over finite sets of symbols that are references to continuous values. This substitution can be viewed as a form of quantifier elimination (the full version presents a detailed analysis from this perspective [3]). The initial state contains a finite set of facts linking the references to plan-independent geometric properties they have to satisfy. Continuing with the example, pose variables range over pose references such as *initPose*, *gp_obj*$_i$, and *pdp_obj*$_i$*_S* for each object *obj*$_i$. Intuitively these references denote the gripper's initial pose, a grasping pose (gp) for *obj*$_i$ and a put-down pose (pdp) for placing *obj*$_i$ on surface S. For these references, the initial state includes facts: *at(gripper, initPose), IsGP(gp_obj*$_i$*, obj*$_i$*), IsPDP(pdp_obj*$_i$*_S, obj*$_i$*, S), IsMP(traj_pose*$_1$*_pose*$_2$*, pose*$_1$*, pose*$_2$*)*, where pose$_1$ and pose$_2$ range over the introduced pose references. The task planner can now use the *pick2D* specification defined above, but with variables that range over discrete, symbolic references to continuous variables. This leads to immense efficiency in representation, compared to discretization, and makes task planning practical.

Returning to the 2D example, the preconditions of *place2D* require two new predicates: *IsPDL(tloc, S)* indicates that *tloc* is a put-down location in S and *PDObstructs(obj', traj, obj, tloc)* indicates that *obj'* obstructs the trajectory *traj* for placing *obj* at *tloc*. In this simple example, we assert that once an object is placed in the region S, it does not obstruct any pickup trajectories.

place2D(obj, gripper, pose$_1$*, pose*$_2$*, traj, tloc)*
precon *In(obj, gripper), At(gripper, pose*$_1$*),*
IsPDP(pose$_2$*, obj, tloc), IsMP(traj, pose*$_1$*, pose*$_2$*),*
IsPDL(tloc, S),
$\forall$*obj'*$\neg$*PDObstructs(obj', traj, obj, tloc)*
effect $\neg$ *In(obj, gripper), At(obj, tloc), Empty(gripper),*
At(gripper, pose$_2$*),*
$\forall obj', traj'$*¬Obstructs(obj, traj', obj')*

As discussed in the introduction, the exact set of obstructions (or other geometric effects) caused by a high-level action cannot be determined using the pose references and logical reasoning capabilities available to the task planning layer. In our formulation the effect list for an action only needs to be *sound*: it needs to contain only those effects that can be guaranteed as a result of the action. Thus, when a free area is not available (as in our experiments) the *place2D* action's effects would not include a removal of all obstructions to pickup trajectories. Such effects are determined through the interface layer if needed (Sec. IV). Further representational optimization is possible by removing the action arguments that do not contribute any functionality to the high-level specification. Such arguments can be reintroduced in task plans prior to refinement. For instance, the *traj* argument of *pick2D* doesn't occur in its effects. It is used in *IsMP*, which is not changed by any high-level action and in *Obstructs*, which is changed by *place2D* for all trajectories, so that the effect is independent of *traj*. In other words, high-level solutions are not affected by this argument. The domain specification can be optimized to remove such arguments and predicates.

pr2Pick(obj$_1$ *gripper, pose*$_1$*, pose*$_2$*, traj)*
precon *Empty(gripper), RobotAt(pose*$_1$*),*
IsBPFG(pose$_1$*, obj),IsGPFG(pose*$_2$*, obj),*
IsMP(traj, pose$_1$*, pose*$_2$*),*
$\forall$ obj' $\neg$ *Obstructs(obj', traj, obj*$_1$*)*
effect *In*$(obj_1, gripper)$, ¬*Empty(gripper),*
$\forall obj', traj'$
¬*Obstructs*$(obj_1, traj', obj')$,
$\forall obj', traj', tloc'$
¬*PDObstructs*$(obj_1, traj', obj', tloc')$
pr2PutDown$(obj, gripper, pose_1, pose_2, traj, targetLoc)$
precon *In*$(obj, gripper)$ $\wedge$ *RobotAt*$(pose_1)$,
IsBPFPD$(pose_1, obj, targetLoc)$,
IsGPFPD$(pose_2, obj, targetLoc)$
IsMP$(traj, pose_1, pose_2)$, *IsLFPD*$(targetLoc, obj)$
$\forall obj'$¬*PDObstructs(obj', traj, obj, tloc)*
effect ¬ *In*$(obj, gripper)$, *At*$(obj, targetLoc)$
pr2Move$(pose_1, pose_2, traj)$
precon *RobotAt*$(pose_1)$, *IsMP*$(traj, pose_1, pose_2)$
effect ¬*RobotAt*$(pose_1)$, *RobotAt*$(pose_2)$

Fig. 3: Action specifications for robots with articulated manipulators.

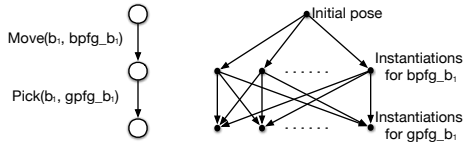
Our approach easily extends to real robots such as the PR2 (see Fig. 3). We can add predicates to capture base poses and gripper poses for grasping (*IsBPFG*, *IsGPFG*) and and for put-down (*IsBPFPD, IsGPFPD*). A base pose for grasping is a pose from which there is a collision-free IK solution to a gripper grasping pose if all movable objects are removed. A significant point of difference in this model is that when an object is picked up, it no longer obstructs any trajectories. Further, the predicate *IsLFPD* determines whether or not a location is one where objects can be placed. This can be true of all locations on surfaces that can support objects.

The truth values of ground atoms over references like *Obstructs(obj*$_{10}$*, traj_pose*$_1$*_pose*$_2$*, obj*$_{17}$*)* are set to defaults in the initial state. Domain-specific initializations can also be generated automatically to facilitate completeness guarantees [3]; during planning, interaction with the interface layer may add or remove such atoms from a state.

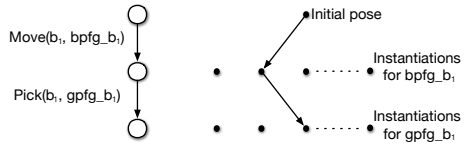
Conditional Costs The approach presented above applies seamlessly to actions whose costs depend on a finite number of geometric predicates over possibly continuous arguments. Further details can be found in the full version [3].

IV. Task and Motion Planning

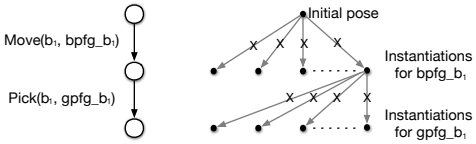
Before formally presenting our algorithm, we illustrate it in action on a simple example to communicate the main intuitions (Fig. 4). This example uses the specification in Fig. 3. Consider an initial task plan obtained using a task planner, and the search space for instantiations of the pose references used in it (Fig. 4a). In scenario 1 the interface layer finds instantiations that correspond to an error-free motion plan, thus solving the problem (Fig. 4b). In scenario 2 the interface layer is unable to find such an instantiation (Fig. 4c). It identifies partial solutions and attempts to extend them using a task planner. In scenario 2a this succeeds with the first partial motion plan (Figs. 4d). The interface layer generates logical facts capturing reasons for the failure and updates the high-level state where this failure occurred. It



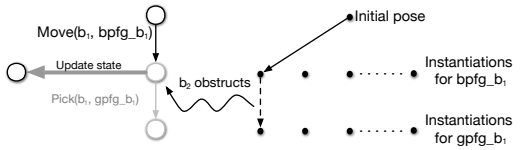
(a) On the left we show a task plan with pose references. On the right we show the search space of possible instantiations of these references: each row represents the space of possible instantiations for references in the preceding action, and each arrow represents a motion planning problem. The initial pose has a unique instantiation as that is the robot's current pose.



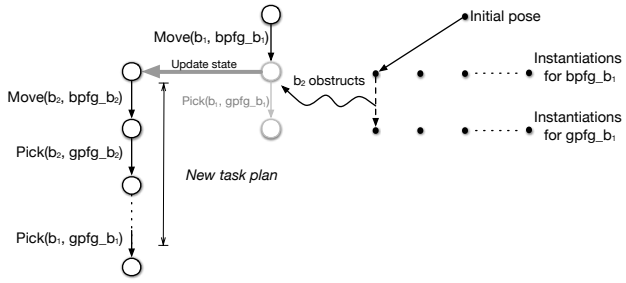
(b) In order to refine the task plan into a motion plan, the interface layer needs to find instantiations for all pose references, such that there is an error-free motion plan between each successive pair of poses. This subfigure captures Scenario 1, where the interface layer finds a set of pose instantiations for which there is an error-free motion plan. This completes the refinement process and the problem has been solved for this scenario.



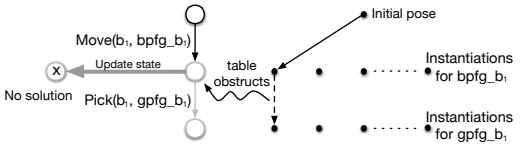
(c) Scenario 2. The interface layer completes backtracking search and finds no complete instantiation of poses with an error-free motion plan. The original task plan cannot be refined into a motion plan.



(d) Scenario 2a. The interface layer selects the first possible pose instantiation and invokes the motion planner, which finds a trajectory for the instantiation corresponding to the first action but not for the second. The motion planner is used to compute a trajectory allowing collisions (by removing movable objects) and the interface layer computes obstructions along it (a motion planner that reports possible collisions could also be used). The task level state is updated with this information represented using symbolic references (Sec. III), resulting in an updated high-level state.



(e) Scenario 2a (ctd.). The interface layer invokes a task planner on the updated state. The task planner generates a new plan, which now consists of first moving b_2 out of the way, and then picking b_1. At this point the interface layer will continue processing the new plan suffix, as described in (b).



(f) Scenario 2b. An alternate execution after scenario 2. The pose instantiation is such that the motion planner can compute an error-free trajectory for moving to *bpfg_b*1, but not from there to *gpfg_b*1. However, now the available trajectory for moving to *gpfg_b*1 is in collision with a table. The interface layer invokes the task planner with the updated state as in 2a, but this call fails because there is no high-level action for moving a table. The interface layer will iterate over pose instantiations until it finds one corresponding to a solvable high-level state. It will then proceed as in Scenario 2a.

Fig. 4: Illustration of the interface layer's refinement process. Action arguments have been abbreviated.

uses a task planner to obtain a new plan to solve the updated state (4e). In scenario 2b, the updated state is found to be unsolvable and the interface layer continues to search for a partial motion plan that corresponds to a solvable task planning problem.

We now describe two algorithms that constitute the interface layer. Alg. 1 describes the outer loop of refinement and regeneration of task plans that continues until a resource limit (e.g. time) is reached. The TryRefine subroutine (Alg. 2) describes the process of refining task plans into trajectories representing motion plans. The implementation of these algorithms uses careful bookkeeping to ensure that Alg. 1 can call TryRefine to either carry out an exhaustive search for an error-free refinement of the entire plan, or to make a sequence of calls to it, each returning a new partial refinement and the errors corresponding to it.

A. Overall Algorithm For the Interface

Alg. 1 begins by invoking a task planner with the given initial state of the task and motion planning problem to get *HLPlan*. In each iteration of the **while** loop, *TryRefine* (Alg. 2) is first called in line 6 in the error-free mode, which searches for a feasible instantiation of the pose references used in HLPlan. If this fails it is called in the partial trajectory mode (line 8). In this mode, repeated invocations of *TryRefine* return with the preconditions responsible for failure (*failCause*) in finding a motion plan corresponding to distinct pose instantiations. These errors are used to update the task level state. This is done by applying the effects of actions in *HLPlan* until *failStep* on the state for which HLPlan was obtained, and then updating the resulting state with *failCause*. In line 10 a task planner is invoked with this new state. If this state is unsolvable, *TryRefine* is used to compute the errors corresponding to the next pose instantiations for the same *HLPlan*. If on the other hand the state was solvable and *newPlan* was obtained, the entire process repeats with the updated plan (line 6). If an upper limit on the number of attempted refinements for *HLPlan* is reached (line 14) the refinement process starts over from the first action in the available plan after resetting the *PoseGenerator* used in *TryRefine*, and removing facts corresponding to pose instantiations.

Algorithm 1: Task and Motion Planning Algorithm

```
   Input: State, InitialPose
1  if HLPlan not created then
2      HLPlan ← callTaskPlanner(State)
3      step ← 1; partialTraj ← None; pose_1 ← InitialPose
4  while resource limit not reached do
5      if TryRefine(pose_1, HLPlan, step, partialTraj,
       ErrFreeMode) succeeds then
6          return refinement
7      repeat
8          (partialTraj, pose_2, failStep, failCause)
                   ← TryRefine(pose_1, HLPlan, step,
                         partialTraj, partialTrajMode)
9          state ← stateUpdate(State, failCause, failStep)
10         newPlan ← callTaskPlanner(state)
11         if newPlan was obtained then
12             HLPlan ← HLPlan[0:failStep] + newPlan
13             pose_1 ← pose_2; step ← failStep
       until NewPlan obtained or MaxTrajCount reached
14     if MaxTrajCount reached then
15         Clear all learned facts from initial state
16         Reset PoseGenerators with new random seed
17         Reset step, partialTraj, pose_1 to initial values
```

Algorithm 2: TryRefine Subroutine

```
   Input: InitialPose, HLPlan, Step, TrajPrefix, Mode
   /* local vars, pose-gens persist across calls */
1  if first invocation or new HLPlan then
2      index ← Step − 1; traj ← TrajPrefix
3      Initialize pose generators
4      pose_1 ← InitialPose
5  while Step − 1 ≤ index ≤ length(HLPlan) do
6      axn ← HLPlan[index]; nextAxn ← HLPlan[index+1]
7      pose_2 ← poseGen(nextAxn).next()
8      if pose_2 is not defined then
9          poseGen(nextAxn).reset()
10         pose_1 ← poseGen(axn).next()
11         index−−; traj ← traj.delSuffixFor(axn)
12     else if GetMotionPlan(pose_1, pose_2) succeeds then
13         if index = length(HLPlan)+1 then return traj
14         traj ← traj + ComputedPath; index++
15         pose_1 ← pose_2
16     else if Mode = PartialTrajMode then
17         return (pose_1, traj, index+1, MPErrs(pose_1, pose_2))
```

B. Refining Task Plans into Motion Plans

We assume without loss of generality that all HLPlans are zero-indexed lists with a NoOp in position 0.

1) TryRefine Subroutine: TryRefine (Alg. 2) can be invoked in two modes: in *ErrFreeMode* it carries out an exhaustive, backtracking search for feasible refinements of the input *HLPlan*; in *PartialTrajMode* it iterates through the possible instantiations for each pose reference used in *HLPlan*, and for each instantiation it returns the first action that has no error-free motion plan and the reason for infeasibility, which can include obstructions in motion plans and general geometric preconditions of actions. The latter are determined by dedicated modules. Such geometric properties are converted into logical facts in terms of pose references (independent of geometric values).

Starting with the input *InitialPose*, in each iteration of the loop, *TryRefine* invokes action-specific *PoseGenerators* to get a possible target pose for the next action (described below). In each iteration, if the *PoseGenerator* for an action runs out of possible poses, the algorithm backtracks (lines 8-11). If another target pose for the next action is available, a motion planner is called with it in line 12. If motion planning succeeds in the error-free mode, the iteration proceeds to the action after next. Otherwise, if *ErrFreeMode* holds, it obtains another target pose for the next action. If *PartialTrajMode* holds, it returns the reasons for failure (line 17).

As an optimization, our implementation of Alg. 2 invokes a motion planner only if IK solutions exist.

2) Pose Generators: The *PoseGenerator* for an action iterates over those values for pose references which satisfy geometric preconditions of that action. Thus, the space of possible values for pose generators can be constructed in a pre-processing step in a manner similar to approaches for precomputation of grasping poses. It is important to note that in our implementation these pose generators are not task-specific: the pose generator for picking up a bowl remains the same regardless of the planning goals, other actions and the rest of the environment.

In our implementation, PoseGenerators iterate over a finite set of randomly sampled values that are only *likely* to satisfy these properties. The random seed for generating these values is reset when *MaxTrajCount* is reached in Alg. 1. More specifically, a *PoseGenerator* generates (a) an instantiation of the pose references used in the action's arguments and (b) a target pose corresponding to each such instantiation. We also allow the pose generator to generate a tuple of target poses (waypoints) if needed, for multi-trajectory actions.

In this way, each PDDL action corresponds to a sequence of poses generated by its PoseGenerator, interleaved with gripper close and open events. The *GetMotionPlan* call in *TryRefine* succeeds for an action with a multi-target pose generator if it can generate a sequence of waypoints with a feasible motion plan linking all. We discuss specific examples of pose generators below.

PoseGenerator for *pr2Pickup* The *pr2Pickup* pose generator instantiates the pose references $bpfg\_obj_i$ and $gpfg\_obj_i$, which need to satisfy the geometric properties *IsBPFG* and *IsGPFG*. For $bpfg\_obj_i$ it samples base poses oriented towards obj_i in an annulus around the object. For $gpfg\_obj_i$, we need poses at which closing the gripper will result in a stable grasp of the object. Computation of effective grasping poses is an independent problem; we assume that such poses are known for each object class (e.g., bowl, can, tray etc.), and used an approach where every grasp pose corresponds to a pre-grasp pose, and a raise pose. The pose generator generates possible values for all of the intermediate poses as waypoints, while the arm always returns to a side pose at the end to enable an unobstructed view from the physical PR2's cameras. The latter could be avoided through a framework incorporating partial observability.

PoseGenerator for *pr2PutDown* The pose generator for *pr2PutDown* instantiates pose references of the form *tloc* (an abbreviation for *targetloc*), *bpfpd_obj$_i$_tloc*, and *gpfpd_obj$_i$_tloc* to satisfy the properties *IsBPFPD* and *IsGPFPD*. *tloc* values are sampled locations on supporting surfaces within a certain radius of the current base pose; values for *bpfpd_obj$_i$_tloc* are obtained by sampling base poses in an annulus around $tloc$ and oriented towards it. Gripper put-down poses of the form *gpfpd_obj$_i$_tloc* are sampled by computing possible grasping poses assuming the object was at *tloc*.

C. Completeness

We present a sufficient condition under which our approach is guaranteed to find a solution if one exists.

Definition 1: A set of actions is *uniform* wrt a goal g and a set of predicates R if for every $r \in R$,

1) Occurrences of r in action preconditions and goal are either always positive, or always negative.
2) Actions can only add r-atoms with the same sign as those used in preconditions and the goal g.

Theorem 1: Let $P = \langle A, s_0, g \rangle$ be a planning problem such that there are no reachable dead-end states w.r.t. g and A is a set of actions whose descriptions are sound w.r.t. continuous effects and uniform w.r.t. the g and the geometric predicates used in the domain. Let G be the pose generator for the pose references used in s_0. If all the calls to the motion planner terminate, then Alg. 1 will find a sequence of motion plans solving P if one exists using the sound descriptions and the pose references captured by G.

Intuitively, the result follows because under the premises, every time a state update takes place, missing geometric facts are added to the state and can only be removed by actions but not added again. We refer the reader to the full version [3] for the proof. Note that the conditions of Thm. 1 are not necessary. In particular, our empirical evaluation shows the algorithm succeeding in a number of tasks that *do not* satisfy the uniformity condition.

V. Empirical Evaluation

We implemented the proposed approach using the OpenRAVE simulator [4]. In all of our experiments we used Trajopt (multi-init mode), which is a state-of-the-art motion planner that uses sequential convex optimization to compute collision avoiding trajectories [5]. For every motion planning query, Trajopt returns a trajectory with a cost. A wrapper script determined collisions (if any) along the returned trajectory. We used two task planners, FF [6] and the IPC 2011 version of FD [7] in `seq-opt-lmcut` mode, which makes it a cost-optimal planner. FD was not appropriate for the first two tasks described below since they used negative preconditions and FD has known performance issues with negative preconditions. Domain compilations for eliminating negative preconditions are possible but impractical as they lead to large numbers of facts in the initial problem specifications. Since our system can work with any classical planner, we used FF for tasks where costs were not a concern. All the problems (Fig. 5) used an ambidextrous version of the PR2 actions shown in Fig. 3, with task-specific actions such as placing items on a tray and opening a drawer. All experiments were carried out on Intel Core i7-4770K machines with 16GB RAM, with two tests running in parallel at a time. All the success rates and times are summarized in Table I. As a baseline, we attempted to solve these problems using a discretization at the task level with all the predicates set at defaults. This effectively removed all geometric constraints. However the planners could not find solutions to these problems after running for more than 25 minutes. The source code and videos for all tasks are available at the URL noted in the introduction [8].

A. Object in a Drawer

In this domain the robot needs to open a drawer and retrieve an object inside it. An object in front of the drawer prevents its complete opening. The inner object's placement determines where the robot should place itself to avoid collisions with the outer object, and whether it is possible to solve the task without moving the outer object. This task illustrates the generality of our approach in going beyond pick-and-place tasks. We modeled it using an open-drawer action, whose pose generator generates random bounded values for the pull-distance. In the solution plans, our system chose to position the robot so as to open the drawer and access the inner object without moving the obstruction when possible. The results show average solution times for situations where removing the obstruction was optional (O) and necessary (N).

B. Cluttered Table

In this task, the objective is to pick up a target object from a cluttered table. There is no designated free space for placing objects, so the planning process needs to find spots for placing obstructing objects. We increased the number of objects up to 40 on a table of fixed dimensions. Fig. 6 shows the table with 40 objects. In order to make the problem more challenging, we restricted pickups to only use side-grasps. The robot's thick grippers create several obstructions and many of the pose instantiations lead to cyclic obstructions. Since placing objects adds obstructions, this task does not satisfy the premises of Thm. 1. In addition to the summarized results in Table I, Fig. 7 shows a histogram of the solution times. To the best of our knowledge, no other approach has

Problem	%Solved in 600s	Avg. Solution Time for Solved (s)
Drawer (O)	100	34
Drawer (N)	100	185
Clutter-15	100	32
Clutter-20	94	57
Clutter-25	90	69
Clutter-30	84	77
Clutter-35	67	71
Clutter-40	63	68
Dinner-2	100	63
Dinner-4	100	133
Dinner-6	100	209

TABLE I: Summary of the results. All numbers except for the cluttered table problem are from 10 randomly generated problems. Cluttered table problems showed greater variance and are averages of 100 randomly generated problems for each number of objects.

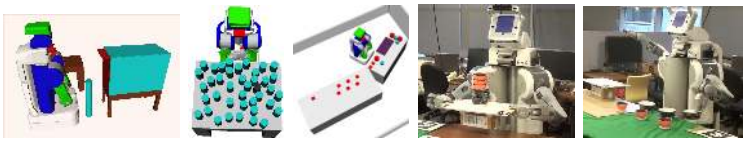


Fig. 5: Test domains from L to R: drawer domain, cluttered table with 40 objects where the dark object denotes the target object, and dinner layout. The rightmost images show the PR2 using the tray and completing the dinner layout.

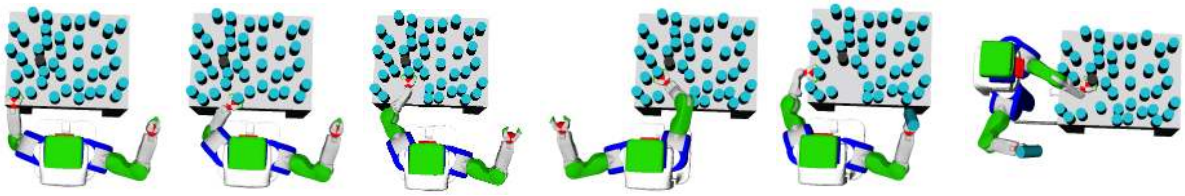


Fig. 6: Some of the grasps executed while solving an instance of the 40 object cluttered table with the dark object as the target.

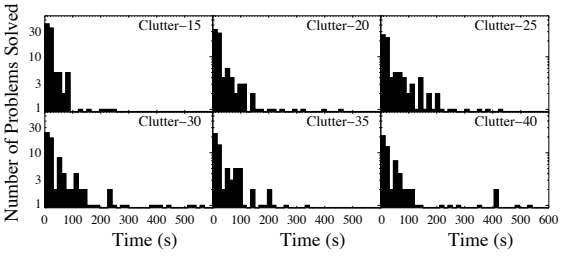


Fig. 7: Histograms of solution times for problems solved within 600s in the cluttered table domain. Y-axis is in log-scale.

been shown to perform at this level on randomly generated constrained problems without using specialized geometric reasoning routines.

C. Laying Out a Table for Dinner

The goal of this task is to lay out a dinner table. A tray is available, but not necessary for transportation. We modeled a scenario where the initial location of objects was far from the target location by asserting that these locations were in different rooms and associating a high cost to all task-level moves across rooms. The geometric properties in this domain were stackability and relative positions of objects (see below). Stackability was determined using object diameters. The test scenarios had 2, 4 and 6 objects (cups and bowls with equal numbers of each), placed at random locations on the table. Objects had random names to prevent the task planner from favoring any particular stacking order. The initial task planner specification allowed all objects to be stacked on each other. Optimal task planning is hard in this domain, as the number of reachable states exceeds 3 million with just 6 objects. We used FD as the task planner since plan cost was a consideration.

Our system appropriately used the tray to transport items. It used inefficient movements when the robot picked objects on its left with its right hand (and vice versa) as the task planner chose hands arbitrarily. We made two modifications to address this, both of which increase the complexity of the task planning problem by increasing its branching factor. We used a conditional cost formulation (Sec. III) to penalize actions which accessed an object or a location on the right (left) with the left (right) hand. We also added a *Handoff* action in the domain, which transfered an object from one hand to the other. The resulting behavior, though not guaranteed to be optimal, showed the system determining which hand to use for a particular grasp or putdown and whether or not a handoff should be done. To the best of our knowledge, no other approach has been shown to solve task and motion planning problems with such large high-level state spaces without using task-specific heuristics or knowledge beyond the set of primitive task-level actions.

D. Real-World Validation

For real-world experiments we used ROS packages for detecting object and table poses (`ar_track_alvar`) and for SLAM (`hector_slam`). A video of the PR2 laying out the table using this system is available at the URL noted in the introduction.

VI. Related Work

Our approach builds upon the vast literature of related work in robotics and planning. In particular, we leverage the immense advances made in task planning and motion planning. Various researchers have investigated the problem of combining task and motion planning [9]–[11]. However, few approaches are able to utilize off-the-shelf task planners and motion planners and most rely on specially designed task and/or motion planning algorithms. Existing approaches do not address the problem of correcting inaccurate task planning descriptions without resorting to discretization. In contrast, our approach (a) represents geometric information in a form that task planners can use and (b) corrects the task planner's representation with information gained through geometric reasoning, without discretization.

Cambon et al. [1] propose a framework that bears similarity to ours in using location references. The references in their approach however are not developed into a system for communicating geometric information to the task planner. Their framework requires the motion planner to use probabilistic roadmaps (PRMs) [12] with one roadmap per movable object, and per permutation of a movable object in each gripper for robots like the PR2. The utilization of task plans is minimal: only their lengths are used as inputs in a heuristic function for a separate search algorithm. However,

their algorithm is probabilistically complete. Kaelbling et al. [13] present a regression-based framework. They utilize as inputs a task hierarchy, action-specific regression functions and generators, and inferential attachments for carrying out limited logical reasoning. The overall framework is complete if the domain is *reversible* (a necessary condition for reversibility is that no dead-end state should be reachable from the initial task planning state), and the primitive actions, which include motion planner invocations, have sound and complete precondition and effect specifications. However, these conditions are only sufficient and not necessary.

Grasping objects in a cluttered environment is an open problem in robotics. Dogar et al. [14] propose replacing pick actions with push-grasps. This would be promising as a primitive action in our overall framework. Techniques have also been developed for navigation among movable obstacles (e.g., [15]), but they do not address the general problem of combining task and motion planning.

Reinvoking task planners relates to replanning for partially observable or non-deterministic environments [16], [17]. However, the focus of this paper is on the substantially different problem of providing the task planner with information gained through geometric reasoning. An alternate representation for dealing with large sets of relevant facts in the initial state would be to treat them as initially unknown and use a partially observable planner with non-deterministic "sensing" actions [18]. However, offline contingent solutions typically don't exist for all possible truth values of geometric predicates. Wolfe et al. [19] use angelic hierarchical planning to define a hierarchy of high-level actions over primitive actions. Our framework could be viewed as using an angelic interpretation: pose references in task plans are assumed to have *a* value that satisfies the preconditions, and the interface layer attempts to find such values. Planning modulo theories (PMT) [20] and planning with semantic attachments [21] also address related problems. In contrast to our objective of utilizing arbitrary task planners, these approaches do not use a discrete task planner. Instead, they include continuous fluents in the task planning specification and utilize search algorithms that make calls to external subroutines for computing the values of such fluents. Erdem et al. [22] also extend the task planner (an ASP solver) with external predicates implemented as arbitrary programs. They use a grid-based discretized representation for representing the task planning problem as well as for the geometric information gained at the continuous planning level. In contrast, this paper was focused on a method for communicating such information to an arbitrary task planner, without discretization.

VII. Conclusions

We presented an approach for combined task and motion planning that is able to solve non-trivial robot planning problems without using task-specific heuristics or any hierarchical knowledge beyond the primitive PDDL actions. Our system works with off-the-shelf task planners and motion planners, and will therefore scale automatically with advances in either field. We also presented a sufficient, but not necessary condition for completeness. We also demonstrated that our system works in several non-trivial, randomly generated tasks where this condition is not met and validated it in the real world with a PR2 robot.

Acknowledgments

We thank Malte Helmert and Joerg Hoffmann for providing versions of their planners that were helpful in early versions of our implementation. We also thank Ankush Gupta for help in working with the PR2 and John Schulman for help in setting up Trajopt. This work was supported by the NSF under grants IIS-0904672 and IIS-1227536.

References

[1] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *IJRR*, vol. 28, pp. 104–126, 2009.

[2] M. Fox and D. Long, "PDDL2.1: an extension to PDDL for expressing temporal planning domains," *JAIR*, vol. 20, no. 1, pp. 61–124, 2003.

[3] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "A modular approach to task and motion planning with an extensible planner-independent interface layer (full version)," 2013. [Online]. Available: http://www.cs.berkeley.edu/~siddharth/icra14/full_version.pdf

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

[5] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization," in *RSS*, 2013.

[6] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *JAIR*, vol. 14, pp. 253–302, 2001.

[7] M. Helmert, "The fast downward planning system," *JAIR*, vol. 26, pp. 191–246, 2006.

[8] "Source code and result videos." [Online]. Available: http://www.cs.berkeley.edu/~siddharth/icra14/

[9] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das, "The CLARAty architecture for robotic autonomy," in *Proc. of IEEE Aerospace Conference*, 2001, pp. 121–132.

[10] E. Plaku and G. D. Hager, "Sampling-based motion and symbolic action planning with geometric and differential constraints," in *ICRA*, 2010, pp. 5002–5008.

[11] K. Hauser, "Task planning with continuous actions and nondeterministic motion planning queries," in *Proc. of AAAI Workshop on Bridging the Gap between Task and Motion Planning*, 2010.

[12] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, pp. 566–580, 1996.

[13] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *ICRA*, 2011, pp. 1470–1477.

[14] M. Dogar and S. Srinivasa, "A framework for push-grasping in clutter," *RSS*, 2011.

[15] M. Levihn, J. Scholz, and M. Stilman, "Hierarchical decision theoretic planning for navigation among movable obstacles," in *WAFR*, 2012, pp. 19–35.

[16] K. Talamadupula, J. Benton, P. W. Schermerhorn, S. Kambhampati, and M. Scheutz, "Integrating a closed world planner with an open world robot: A case study," in *AAAI*, 2010.

[17] S. W. Yoon, A. Fern, and R. Givan, "FF-replan: A baseline for probabilistic planning," in *ICAPS*, 2007.

[18] B. Bonet and H. Geffner, "Planning with incomplete information as heuristic search in belief space," in *ICAPS*, 2000, pp. 52–61.

[19] J. Wolfe, B. Marthi, and S. J. Russell, "Combined task and motion planning for mobile manipulation," in *ICAPS*, 2010, pp. 254–258.

[20] P. Gregory, D. Long, M. Fox, and J. C. Beck, "Planning modulo theories: Extending the planning paradigm," in *ICAPS*, 2012.

[21] A. Hertle, C. Dornhege, T. Keller, and B. Nebel, "Planning with semantic attachments: An object-oriented view," in *ECAI*, 2012.

[22] E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, and T. Uras, "Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation," in *ICRA*, 2011, pp. 4575–4581.