Object-Centric Task and Motion Planning in Dynamic Environments

Toki Migimatsu and Jeannette Bohg

Abstract— We address the problem of applying *Task and Motion Planning* (TAMP) in real world environments. TAMP combines symbolic and geometric reasoning to produce sequential manipulation plans, typically specified as joint-space trajectories, which are valid only as long as the environment is static and perception and control are highly accurate. In case of any changes in the environment, slow re-planning is required. We propose a TAMP algorithm that optimizes over Cartesian frames defined relative to target objects. The resulting plan then remains valid even if the objects are moving and can be executed by reactive controllers that adapt to these changes in real time. We apply our TAMP framework to a torque-controlled robot in a pick and place setting and demonstrate its ability to adapt to changing environments, inaccurate perception, and imprecise control, both in simulation and the real world.

I. INTRODUCTION

Robot manipulation tasks often require a combination of symbolic and geometric reasoning. For example, if a robot wants to grasp an object outside of its workspace, a symbolic planner might devise a plan to use a long stick to bring the object closer, while a motion planner would determine where to grasp the stick and how to use it to push the object. *Task and Motion Planning* (TAMP) algorithms address this challenge of interleaving symbolic and geometric reasoning. They often produce full trajectories for the robot to execute. However, due to the PSPACE-hard complexity of motion planning [1], generating these trajectories can take on the order of minutes, and if the environment changes, these trajectories may become invalid and require replanning.

In this paper, we combine TAMP with reactive control to enable robot manipulation in real-world scenarios where the environment may change during execution, perception may be inaccurate, and robot control may be imprecise. Instead of generating full joint-space trajectories like most TAMP algorithms, our approach returns desired object poses at key timepoints. These object poses are defined relative to target frames, which may be moving. Optimizing over relative poses instead of joint configurations at this stage of planning facilitates integration with local controllers that can react to changes in the environment in real time. For example, if the robot needs to pick up an object but specifies this pick action using a desired joint configuration, this joint configuration becomes invalid as soon as the object's position changes, leading to unintended contact or external disturbances. However, if the planner outputs a desired pose of the end-effector relative to the object, then this relative pose is still valid no matter where the object moves, eliminating
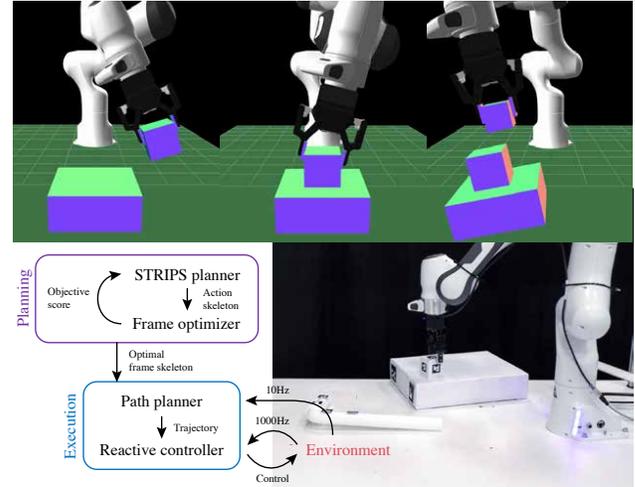
Fig. 1. The proposed TAMP algorithm plans over relative Cartesian frames for integration with reactive controllers that can adapt to changes in the environment in real-time. The top row demonstrates the robot's ability to complete the block stacking task without stopping to re-plan even when the poses of the grasped object and target location change. The bottom row summarizes our TAMP framework and shows an example execution on the real robot (see supplementary material for a video).

the need to stop and replan. Figure 1 provides an overview of the proposed framework.

In addition to facilitating integration with reactive controllers, planning over object poses also has the benefit of being a more natural representation for object manipulation. The geometric constraints for actions such as *pick*, *place*, and *push* are straightforward in Cartesian space, which simplifies the process of defining new manipulation actions.

The main theoretical contributions of this work are twofold. First, we introduce a formulation of TAMP using object-centric frames that work with reactive controllers to accommodate changing environments, inaccurate perception, and imprecise control. Second, we derive the $\mathbf{SE}(3)$ functions necessary for optimizing object-centric frames in pick and place applications. We then apply this TAMP framework to a manipulation setting where a robot can *pick*, *place*, and *push*, and demonstrate it working on a real robot where the environment state is visually tracked with RGB-D cameras.

II. RELATED WORK

Task and motion planning has been widely studied since the introduction of STRIPS, the first automated task planner [2]. One common approach of TAMP algorithms is to interleave the symbolic and geometric search processes by calling a motion planner at every step of the symbolic search and tentatively assigning geometric parameters to the current symbolic state before advancing to the next one [3, 4].

One issue that arises from interleaving symbolic and geometric search is when a state found in the search process is symbolically legal but geometrically infeasible due to a previous instantiation of geometric parameters. Lagriffoul et al. [5, 6] introduce the concept of geometric backtracking, where the planner must search for alternative geometric instances until a solution is found. They propose an algorithm that maintains interval bounds over placement parameters to reduce the search space of geometric backtracking. Bidot et al. [7] limit geometric backtracking with heuristics based on statistics for kinematic violations and collisions with movable objects. de Silva et al. [8] use *Hierarchical Task Networks* (HTNs) to perform symbolic search using hierarchically abstracted tasks, performing HTN backtracking whenever geometric backtracking fails. *Hierarchical Planning in the Now* (HPN) [9] similarly uses a hierarchical approach, but interleaves planning with execution, such that primitive actions at the lowest level of the hierarchy are executed as soon as they are reached by the search algorithm. This requires that the actions themselves be reversible when backtracking is necessary.

While integrating geometric search with symbolic search can prune out large sections of the symbolic space, calling motion planning with every symbolic search increment can be costly, and even disadvantageous if most states are geometrically feasible [10]. An alternative approach is to perform geometric search only on full candidate symbolic plans, or action skeletons. One method of conveying geometric feasibility back to the symbolic search is to construct predicates from the geometric parameters [11, 12, 13]. Dantam et al. [14] use an incremental satisfiable modulo theory (SMT) solver to incrementally generate action skeletons and invoke a motion planner in between for validation. Similarly, Zhang and Shah [15] formulate the TAMP problem as a *Traveling Salesman Problem* (TSP), using a motion planner in the inner loop of a symbolic planner to update the weights of the TSP.

While most TAMP algorithms use sampling-based motion planners, Toussaint et al. [16, 17] use nonlinear optimization to identify geometric parameters for given action skeletons. This approach, called *Logic Geometric Programming* (LGP), is what we extend in this work to handle dynamic environments. Advantages of using nonlinear optimization include the ability to consider all geometric parameters simultaneously without the need for backtracking, as well as the ability to encode complex goals in the objective such as minimizing the energy of the robot or maximizing the height of a stack of blocks.

A common limitation of the TAMP methods above is that planning is slow, taking tens of seconds to minutes. To this end, Garrett et al. [18] reduce the sampling space of motion planning by conditionally sampling on factoring patterns in the action skeletons. Wells et al. [19] use a *Support Vector Machine* (SVM) to estimate the feasibility of actions to guide the symbolic search, only calling the motion planner on action skeletons classified by the SVM as feasible.

Even with these improvements in efficiency, the theoretical complexity of TAMP makes it impractical to execute inside a real-time closed-loop controller. Thus, common assumptions in TAMP are that objects are static, object poses are accurately perceived, and robot commands are executed precisely. Otherwise, errors can accumulate, becoming problematic when a robot needs to grasp an object that it imprecisely placed earlier in the plan, for example. To deal with perception errors, Suárez-Hernández et al. [20] incorporate a symbolic action where the robot examines an object up close when perception uncertainty is high. However, this approach is still unable to handle dynamic environments.

We address these limitations by planning over relative object poses, so that an open-loop plan generated by the TAMP algorithm can still be executed by a real-time closed-loop controller even if object poses change during execution.

III. LGP Background

A. General LGP Formulation

An LGP is comprised of two subproblems: a STRIPS problem [2] specified using first-order logic that operates in a discrete domain, and a nonlinear trajectory optimization problem that operates in a continuous domain. The goal of an LGP is first to find a sequence of K discrete actions $a_{1:K}$ that results in a discrete state trajectory $s_{1:K}$ such that the final state s_K satisfies a set of goal propositions $\mathfrak{g}$. Second, the LGP needs to find a sequence of continuous control inputs $u(t)$ across time $t = [0, T]$ such that the resulting state trajectory $x(t)$ satisfies the requirements of $s_{1:K}$. This can be framed as the following optimization problem:

$$
\begin{aligned}
\underset{a_{1:K}, u(t)}{\arg \min} \quad & h(x(T)) + \int_0^T g(x(t), u(t)) \, dt \\
\text{subject to} \quad & \\
& x(0) = x_{init}, \quad s_K \vDash \mathfrak{g}, \\
& \dot{x}(t) = f_{path}(x(t), u(t), s_{k(t)}) \quad t \in [0, T], \\
& \dot{x}(t_k) = f_{switch}(x(t_k), u(t_k), a_k) \quad k = 1, \dots, K, \\
& s_k \in \text{succ}(s_{k-1}, a_k) \quad k = 1, \dots, K
\end{aligned}
\tag{1}
$$

h is the terminal cost and g is the trajectory cost typical in optimal control problems. f_{path} describes the continuous system dynamics, which change depending on the discrete state $s_{k(t)}$ at a given timestep t. An example of this dependency is when a robot manipulator throws a ball—while the manipulator is holding the ball, the ball's trajectory is determined by the manipulator's dynamics, but after release, it follows unconstrained projectile dynamics. f_{switch} describes the instantaneous system dynamics at timesteps t_k when the discrete action a_k changes. An example of such a constraint is when a robot uses a stick to hit a ball—at the time of impact, the ball's acceleration is determined by the instantaneous impulse imparted by the stick.

In STRIPS planning, each action a_k defines a set of preconditions that must be true before the action is performed and a set of postconditions that will be true after the action is performed. These discrete transition dynamics are encoded in the succ constraint using first-order logic.

While LGPs can be cast as Mixed-Integer Nonlinear Programs, such problems are in general undecidable and cannot be solved exactly [21]. Toussaint [16] proposes an algorithm that approximately solves LGPs by breaking it into a multi-part process. First, a tree search is performed in the STRIPS domain to find a candidate action skeleton $a_{1:K}$. Next, the action skeleton spawns a trajectory optimization problem whose constraints are defined by the action skeleton and corresponding symbolic states. This trajectory optimization is first solved over key timesteps t_k for $k = 1, \ldots, K$, and then subsequently over all timesteps $t = [0, T]$. This optimization will either produce a full trajectory $x_{1:T}$ with an objective score, or fail to find a solution, which means the candidate action skeleton is not physically feasible. The tree search continues until the user decides to terminate, at which point the trajectory with the smallest objective score gets returned.

B. Joint Space Formulation

Toussaint et al. [17] formulate the trajectory optimization subproblem as a k-order Markov Optimization (KOMO) problem, which discretizes time and represents time derivatives of position (up to kth order) with their discrete equivalents. The state space and control inputs are defined to be the manipulator configuration q extended with an extra 6-dof free body joint for each object the robot is manipulating. This augmented configuration is represented by $\bar{q}_t \in \mathbb{R}^{n+6m_t}$, where m_t is the number of objects the robot is manipulating at timestep t. Note that the size of $\bar{q}_t$ changes with t.

$$
\begin{aligned}
\underset{\bar{q}_{0:T}}{\arg\min} \quad & h(\bar{q}_T) + \sum_{t=0}^{T} g(\bar{q}_{t-2:t}) \\
\text{subject to} \quad & \bar{q}_{-2:0} = \bar{q}_{init}, \\
& f_{path_{k(t)}}(\bar{q}_{-2:t}) = 0 \quad t = 1, \ldots, T, \\
& f_{switch_k}(\bar{q}_{-2:t_k}) = 0 \quad k = 1, \ldots, K
\end{aligned}
\tag{2}
$$

The objective term $g(\bar{q}_{t-2:t})$ is a function that can take into account the position, velocity, and acceleration at each timestep. The sum of these functions across all timesteps results in a banded Hessian that can be inverted efficiently for Gauss-Newton methods. However, the constraint functions depend on the entire trajectory of joint positions—for example, the constraint associated with placing an object on a table is affected by how it was picked up at a previous timestep, since this determines the configuration of the object in the robot's end-effector. This coupling between timesteps makes the Jacobians for these constraints complex, since the constraint functions are dependent on all previous timesteps of the trajectory, not just the timestep of the constraint itself.

IV. Cartesian Formulation

In this section, we introduce the new Cartesian frame formulation that enables adaptation to dynamic environments. At the most basic level, a manipulation task can be reduced to controlling a point relative to some reference frame, e.g., controlling the tip of a hammer relative to the head of a nail. Instead of optimizing over augmented joint configurations of
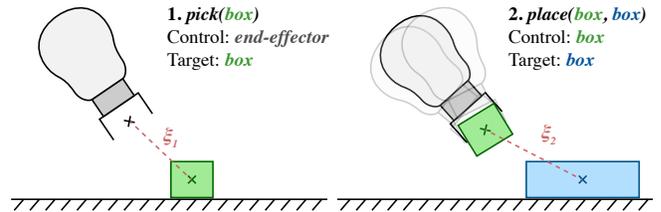

Fig. 2. Optimization variables ξ_t define the 6-dof pose of the control frame relative to the target frame. The place task is unaffected by the grasp executed by the preceding pick task, since it only involves the relative pose between the two boxes. This decouples the action constraints in the optimization problem.

the robot and manipulated objects, the optimization variables in this new formulation are defined to be the relative pose between the control and target frames at each timestep. In a real system, this pose could be observed from a combination of 3D pose estimation and forward kinematics of the robot.

Let $\text{control}(t)$ and $\text{target}(t)$ be the control and target frames determined by the action at time t in the action skeleton. The relative pose of $\text{control}(t)$ in $\text{target}(t)$ is represented by a 6-dof variable $\xi_t = \begin{pmatrix} \xi_{t_p} & \xi_{t_r} \end{pmatrix}^T$, where $\xi_{t_p} \in \mathbb{R}^3$ is the position and $\xi_{t_r} \in \mathbb{R}^3$ is the orientation in axis-angle form. The axis-angle representation is chosen for the orientation because it does not have any constraints over its parameters and does not suffer from kinematic singularities like gimbal lock. Although it has representation singularities at rotations of $k\pi$, this should not pose a problem for the optimizer. Note that ξ_t is a representation of the Euclidian lie algebra $\mathfrak{se}(3)$.

$$
\begin{aligned}
\underset{\xi_{0:T}}{\arg\min} \quad & h(\xi_{0:T}) + \sum_{t=1}^{T} g_t(\xi_{0:t}) \\
\text{subject to} \quad & \xi_0 = \xi_{init}, \\
& f_{path_{k(t)}}(\xi_t) = 0 \quad t = 1, \ldots, T, \\
& f_{switch_k}(\xi_{t_k}) = 0 \quad k = 1, \ldots, K
\end{aligned}
\tag{3}
$$

In this Cartesian formulation of the problem, we only consider the positions of the frames in the trajectory and ignore velocities and accelerations. Dynamic motions that require controlling velocities and accelerations are more likely to require high control frequencies that this stage of the trajectory planning cannot offer. Therefore, to enable real-time reactive behavior, the trajectory planner considers only key positions in the trajectory (e.g., at all the switch timesteps) and leaves the higher derivatives up to controllers designed for each action.

One advantage of optimizing over local frames, in addition to facilitating integration with local controllers and enabling reactive behavior to changing object positions, is that the optimization constraints become decoupled in time. For example, with joint configuration variables, the constraint for placing an object on a table depends on the configuration used to pick up the object, since this determines the pose of the object relative to the end-effector. However, in the Cartesian formulation, if we define the variable for the place constraint to be the relative pose between the object and the table, then this constraint no longer depends on the
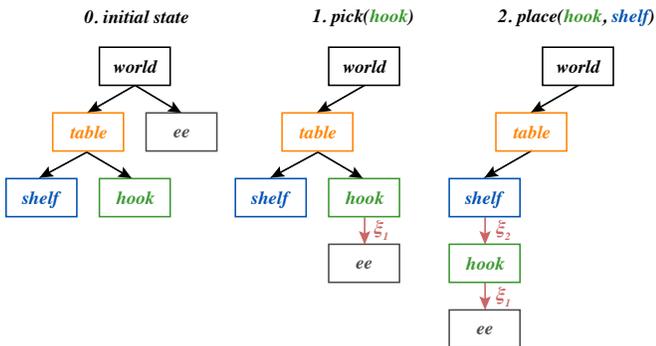
Fig. 3. Diagram of the kinematic tree. In the initial state, the hook and shelf are on the table, and the table and end-effector are positioned with respect to the world. During $pick(hook)$, ξ_1 defines the end-effector's pose relative to the hook such that the hook becomes the end-effector's parent. During $place(hook, shelf)$, ξ_2 defines the hook's pose relative to the shelf, but the end-effector is still fixed to the hook via ξ_1. Thus, computing the absolute pose of the end-effector at $t = 2$ involves composing the relative poses down the chain from the end-effector to the world.

configuration used to pick up the object, since the relative pose of the object in the end-effector is irrelevant (Fig. 2). This decoupling simplifies the Jacobians for the constraints, since each constraint is only a function of a configuration variable at a single time step.

Instead, the cross-time dependency gets pushed onto the objective function, which may measure something like the total distance travelled by the end-effector, described below.

A. Objective Function

The objective for our trajectory optimization problems measures a combination of the linear and angular distance travelled by the end-effector. Let $\xi \in \mathbb{R}^{6(T+1)}$ be the vector concatenation of $\xi_0, \ldots, \xi_T$. Given scaling factors $\alpha, \beta \in \mathbb{R}$, the combined distance between timesteps $t - 1$ and t is

$$
\begin{aligned}
g_t(\xi) = &\ \alpha \left\| x_{ee}(\xi; t) - x_{ee}(\xi; t - 1) \right\|_2^2 \\
&+ \beta \left\| \log \left(\phi_{ee}^{-1}(\xi; t - 1)\, \phi_{ee}(\xi; t) \right) \right\|_2^2
\end{aligned}
\tag{4}
$$

where $x_{ee}(\xi; t)$ and $\phi_{ee}(\xi; t)$ compute the absolute position and orientation, respectively, of the end-effector at time t given the entire history of relative pose variables ξ. $\log$ is the **SO**(3) logarithmic map that in this case computes the shortest axis-angle rotation between $\phi_{ee}(\xi; t - 1)$ and $\phi_{ee}(\xi; t)$.

Such an objective function requires mapping the relative variables ξ_t to the absolute pose of the end-effector in the world frame, which may depend on the control actions leading up to the current timestep. This makes the gradient for the objective function more complex, but this may be outweighed by the benefit of simpler constraint Jacobians, which greatly simplifies the process of defining new actions for the optimization framework. The mapping from relative poses to absolute poses is described in detail below.

B. Absolute Pose

The variable ξ_t defines the pose of $\text{control}(t)$ relative to $\text{target}(t)$. This means that $\text{control}(t)$ is kinematically attached to $\text{target}(t)$, because any pose change of $\text{target}(t)$ will result in the same pose change of $\text{control}(t)$. This means that computing the absolute world pose of $\text{control}(t)$ requires

computing the pose of $\text{target}(t)$ first. The pose of $\text{target}(t)$ may be defined relative to its own kinematic parent, forming a tree structure with the world frame at the root.

The kinematic tree changes at each timestep, when a manipulation action changes the kinematic parent of the control frame to a new target frame (Fig. 3). This change lasts for all subsequent timesteps until the same control frame is manipulated again. For example, an action that places a ball on the table at time t defines $\text{control}(t)$ to be the ball frame and $\text{target}(t)$ the table frame. Until the ball is manipulated by another action, such as $pick$, the ball will remain fixed to the table, with its relative pose defined by ξ_t.

At any given timestep t, the pose of frame i relative to its parent $\lambda(i; t)$ is either equivalent to its relative pose at the previous timestep or is defined by a new pose ξ_t. Formally, this can be written with the recursive definition

$$
{}^{\lambda(i)}x_i(\xi; t) = \begin{cases} \xi_{t_p} & i = \text{control}(t) \\ {}^{\lambda(i)}x_i(\xi; t - 1) & \text{otherwise} \end{cases}
\tag{5a}
$$

$$
{}^{\lambda(i)}R_i(\xi; t) = \begin{cases} \exp(\xi_{t_r}) & i = \text{control}(t) \\ {}^{\lambda(i)}R_i(\xi; t - 1) & \text{otherwise} \end{cases}
\tag{5b}
$$

where ${}^{\lambda(i;t)}x_i : \mathbb{R}^{6(T+1)} \mapsto \mathbb{R}^3$ and ${}^{\lambda(i;t)}R_i : \mathbb{R}^{6(T+1)} \mapsto \mathbb{R}^{3\times3}$ are the mappings from relative poses ξ to the position and orientation, respectively, of frame i in its parent $\lambda(i; t)$. $\exp$ is the **SO**(3) exponential map that converts the axis-angle representation ξ_r to its rotation matrix representation.

We can compute the pose of frame i in any frame j by applying (5) through the kinematic tree from i to j. The absolute world position and orientation $x_{ee}(\xi; t)$ and $\phi_{ee}(\xi; t)$ of the end-effector is computed in this manner.

Any objective function involving $x_{ee}(\xi; t)$ and $\phi_{ee}(\xi; t)$ requires the Jacobians $\frac{\partial x_{ee}(\xi; t)}{\partial \xi} \in \mathbb{R}^{3\times6T}$ and $\frac{\partial \phi_{ee}(\xi; t)}{\partial \xi} \in \mathbb{R}^{3\times6T}$ for gradient-based optimization methods. Because the combination of x_{ee} and ϕ_{ee} is a representation of the lie group **SE**(3), and ξ_t is a representation of the associated lie algebra $\mathfrak{se}(3)$, deriving these Jacobians is non-trivial. Their closed form solutions are given in Appendix A.

V. MANIPULATION PRIMITIVES

In this section, we define the constraint functions for each primitive manipulation action as visualized in Figure 4.

A. Pick

The $pick(a)$ action allows the manipulator to pick up a movable object a. The control frame is the end-effector and the target frame is object a. The preconditions are that the end-effector cannot be holding any objects before, and the postconditions are that a is in the hand and no longer placed on another object. The symbolic definition is:

Parameters: a: $movable$, obj
Precondition: $\forall b.\ \neg inhand(b)$
Postcondition: $inhand(a) \land \forall b.\ \neg on(a, b)$

The constraint for $pick$ requires that the end-effector control point be inside the object, or the signed distance between the control point and object is negative. Here, we
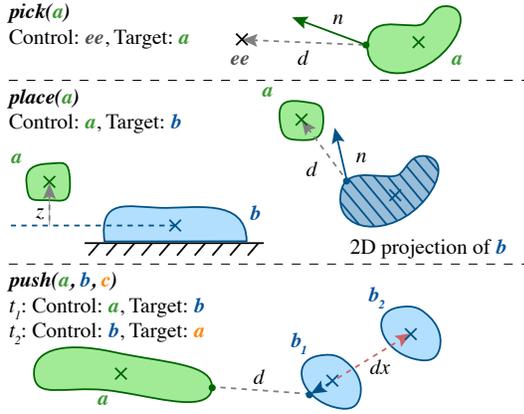
Fig. 4. The *pick* constraint requires that the signed distance between the end-effector and target is negative. The *place* constraint requires that a is above b and that its center of mass is inside the support area of b. The *push* constraint requires that the contact normal between a and b passes through the center of mass of b and points in the desired direction of the push.

assume that a lower level controller will be able to perform the actual grasp given a rough desired position on the object.

Given function $\text{proj}(x; a)$ that finds the closest point on the mesh of object a to point x, and function $\text{normal}(x; a)$ that finds the mesh normal of object a at point x, we can compute the signed squared distance as follows:

$$p := \text{proj}(\xi_{t_p}; a) \in \mathbb{R}^3 \tag{6a}$$

$$n := \text{normal}(p; a) \in \mathbb{R}^3 \tag{6b}$$

$$d := \xi_{t_p} - p \tag{6c}$$

$$f_{pick}(\xi_t; a) = \frac{1}{2} \text{sign}(d \cdot n) \|d\|_2^2 \leq 0 \tag{6d}$$

Note that ξ_{t_p} represents $^a x_{ee}(t)$. A closed-form solution for the Jacobian of the signed squared distance can be derived by linearizing the mesh at the projection point. The Jacobian points in the direction of the normal at the projection point.

$$\frac{\partial f_{pick}(\xi_t; a)}{\partial \xi_t} = \left(\|d\|_2 n^T \quad 0^{1 \times 3} \right) \tag{7}$$

B. Place

The $place(a, b)$ action allows the manipulator to place object a on object b. The control frame is object a and the target frame is object b. The preconditions are that a must be in the end-effector and b must be in the workspace of the robot. The postconditions are that a is no longer in the end-effector and a is on b. The symbolic definition is:

Parameters: a: *movable, obj*; b: *obj*
Precondition: $\neg eq(a, b) \wedge inhand(a) \wedge inworkspace(b)$
Postcondition: $\neg inhand(a) \wedge on(a, b)$

The f_{place} constraint is a concatenation of f_{touch}, $f_{support\_area}$, and $f_{support\_normal}$, which constrains the pose of a such that it is touching b, inside the support area of b, and supported by a contact normal opposite gravity.

1) Touch: Let $\text{contact}(\xi_t; a, b)$ be a function to compute the signed distance d between the closest points p_a and p_b between a and b (or the farthest points of overlap if they are

interpenetrating) as output by GJK [22]. The constraint is:

$$(d, p_a, p_b) := \text{contact}(\xi_t; a, b) \in (\mathbb{R}, \mathbb{R}^3, \mathbb{R}^3) \tag{8a}$$

$$f_{touch}(\xi_t; a, b) = \frac{1}{2}|d|d = 0 \tag{8b}$$

Like the *pick* action, the linear portion of the Jacobian can be computed by linearizing the mesh at the contact point. The angular portion is approximated by finite differencing.

$$n := \text{normal}(p_a, a) \in \mathbb{R}^3 \tag{9a}$$

$$\frac{\partial f_{touch}(\xi_t; a, b)}{\partial \xi_{t_p}} = |d|n^T \tag{9b}$$

2) Support Area: To ensure that a is supported by b, we check that the center of mass of a is above the 2d projection of b in the plane perpendicular to gravity. If we assume the frame of a is positioned at its center of mass, then ξ_{t_p} represents $^b x_{com_a}(t)$, the center of mass of a in b at timestep t. This constraint is then equivalent to f_{pick}, except in 2d.

$$p := \text{proj}\left(\xi_{t_p}; b\right) \in \mathbb{R}^2 \tag{10a}$$

$$n := \text{normal}(p; b) \in \mathbb{R}^2 \tag{10b}$$

$$d := \xi_{t_p} - p \tag{10c}$$

$$f_{support\_area}(\xi_t; a, b) = \frac{1}{2} \text{sign}(d \cdot n) \|d\|_2^2 \leq 0 \tag{10d}$$

In the case where a is non-convex, we split a into convex regions and compute the above function with each region's center of mass. The Jacobian is:

$$\frac{\partial f_{support\_area}(\xi_t; a, b)}{\partial \xi_{t_p}} = \left(\|d\|_2 n^T \quad 0 \right) \tag{11}$$

If a is non-convex, the angular portion of the Jacobian is computed with finite differencing. Otherwise, it is zero.

3) Support Normal: To ensure that the contact normal between a and b is pointing in the right direction (against gravity), we use a heuristic by constraining the height of a to be above the center of mass of b. This simplification assumes that b is convex and relatively flat, which means this constraint may produce a physically infeasible solution if b is something like a sphere, where a surface normal above the center of mass may be pointing sideways, not against gravity. However, objects with such geometry are also difficult to use as placement targets in the real world, and can be filtered out from the optimization with symbolic preconditions.

$$f_{support\_normal}(\xi_t) = -\xi_{t_z} \leq 0 \in \mathbb{R} \tag{12}$$

C. Push

The $push(a, b, c)$ action allows the manipulator to use a to push b on top of c in cases where b is outside the manipulator's workspace. The preconditions are that a has to be held by the end-effector, b has to be on c, and c has to be at least partially in the manipulator's workspace. The postcondition is that b is in the workspace, which assumes that the trajectory optimizer will be able to find a target push location inside the workspace. If such a location cannot be found, the optimizer will fail and a different action skeleton will be chosen. The symbolic definition is:

Parameters: a: *movable, obj*; b: *movable, obj*; c: *obj*

Precondition: $\neg eq(a,b) \;\wedge\; \neg eq(a,c) \;\wedge\; \neg eq(b,c) \;\wedge\; inhand(a) \wedge inworkspace(c)$

Postcondition: $inworkspace(b)$

This action is composed of two timesteps. First, the manipulator must position a so that it is ready to push b. Second, it must push b to the target position. In the first timestep, the control is a and the target is b. In the second timestep, the control is b and the target is c. Because a is a child of b, it will follow b wherever it is controlled.

The constraint at the first timestep is a concatenation of f_{touch} from $place$ and a f_{push_normal} constraint that forces the contact normal through the object's center of mass to be in the direction of the push. The constraint $f_{push\_direction}$ at the second timestep restricts the target push position to be along the surface of c, which is assumed to be flat. Because this action involves three frames a, b, and c across two timesteps, it introduces coupling between three timesteps to compute the relative poses of the three frames. The first timestep relates frame a to b, and the second relates b to c, but to relate a to c, it is necessary to look at when b was last placed in c. This coupling is limited to at most 3 timesteps, meaning the Jacobian depends on at most 18 variables.

1) Push Normal: Let $\mathrm{cast}(x, \delta x; a)$ be a raycasting function that finds the point of intersection between a ray with origin x and direction δx and object b.

$$\delta x := {}^b R_c(\xi; t)\, {}^c x_b(\xi_{t+1}) - {}^b x_c(\xi; t) \in \mathbb{R}^3 \tag{13a}$$

$$p_b := {}^a R_b(\xi_t)\, \mathrm{cast}\left({}^b x_{com_b}, -\delta x; b\right) \in \mathbb{R}^3 \tag{13b}$$

$$p_a := \mathrm{proj}(p_b; a) \in \mathbb{R}^3 \tag{13c}$$

$$d := p_b - p_a \tag{13d}$$

$$f_{push\_normal} = \frac{1}{2}\|d\|_2^2 = 0 \tag{13e}$$

The linear portion of the Jacobian at the first timestep can be simplified by assuming the contact point p_a will stay fixed to a for small changes in a's position. Then, the Jacobian is

$$\frac{\partial f_{push\_normal}(\xi_t; a, b)}{\partial \xi_{t_p}} = -d^T \tag{14}$$

The rest of the Jacobian can be computed with finite differencing in the remaining variables.

2) Push Direction: If we assume that c has a flat surface whose normal points in the z direction, this constraint can simply constrain movement along z and rotation about x and y to be zero. Let $s \in [-1, t)$ be the last timestep when b was positioned relative to c, where -1 indicates that b has not been manipulated yet. ξ_s is the corresponding variable, or if $s = -1$, then the initial pose of b in c.

$$d\xi := \xi_{t+1} - \xi_s \tag{15a}$$

$$d := \begin{pmatrix} d\xi_{p_z} & d\xi_{r_x} & d\xi_{r_y} \end{pmatrix} \tag{15b}$$

$$f_{push\_direction} := \frac{1}{2}\|d\|_2^2 = 0 \tag{15c}$$

The Jacobian for the second timestep is

$$\frac{\partial f_{push\_direction}(\xi_t; a, b)}{\partial \xi_{t+1}} = \begin{pmatrix} 0 & 0 & d^T & 0 \end{pmatrix} \tag{16}$$

and the Jacobian for s, if it exists, is the negation of (16).

D. Collision

Every manipulation action includes an additional constraint to ensure objects do not collide. The objects can be divided into two sets: the manipulated objects $\mathcal{M}$, which consist of all objects between the end-effector and the control frame, and the environment objects $\mathcal{E}$, which consist of all other objects. Let dist be the signed distance between the closest points between a and b (equivalent to the first value in the tuple returned by $\mathrm{contact}$. The constraint function is:

$$f_{collision}(\xi_{1:t}; a, b) = - \min_{a \in \mathcal{M},\, b \in \mathcal{E}} \mathrm{dist}(\xi_{1:t}; a, b) \le 0 \tag{17}$$

The Jacobian is computed with numerical differentiation.

VI. Implementation

A. Planning

Our STRIPS planner takes in a PDDL specification and performs a simple breadth first tree search to find candidate action skeletons. For more complex symbolic problems, a state of the art planner such as FD [23] might be considered.

Our Cartesian frame optimizer currently leverages two different nonlinear optimizers: IPOPT [24] and NLOPT [25]. Any state of the art nonlinear optimizer can be plugged in at this stage by writing wrappers for our LGP optimization API. IPOPT is an interior point method, while NLOPT features many methods. We chose the Augmented Lagrangian method with Sequential Quadratic Programming for its speed and robustness. All optimization variables are initialized to zero.

B. Control

The $pick$, $place$, and $push$ actions use operational space control [26]. An attractive potential field in Cartesian space is created between the control frame and target frame to allow the robot to track moving targets. Repulsive potential fields are used to avoid obstacles. We choose to use torque control so that the robot can safely interact with the environment even with large perception uncertainty. Details of this controller are specified in Appendix B.

For grasping, the frame optimization simply outputs a position inside the object. To convert this into a grasp pose for a two-fingered gripper, we use a heuristic that finds the top-down orientation at the optimized position with the smallest object width between the fingers. This heuristic may fail if the optimizer outputs a position that is not graspable, but it is sufficient for the simple objects in our application.

C. Perception

To track the 6-dof poses of objects, we use DBOT with particle filtering [27, 28] using depth data from a Kinect v2 downsampled to 320×180. Although DBOT can track objects with centimeter accuracy, even if they are partially occluded, it requires an initialization of the object poses.

TABLE I. IPOPT vs. NLOPT: Optimization Scores and Times

Solution	Optimal score		Time [s]	
	IPOPT	NLOPT	IPOPT	NLOPT
Tower on left plate	**0.444**	—	**3.63**	—
Tower on middle plate	**0.404**	—	**0.96**	—
Hook on table	**0.254**	0.274	1.18	**0.87**
Hook on shelf	**0.273**	0.358	3.71	**1.42**
Hook on box	0.287	**0.283**	1.14	**0.85**

Furthermore, the particle filter can lose sight of objects if they move too quickly while occluded. To alleviate these two issues, we use ArUco fiducial marker tracking [29] as a control input to DBOT to bias the tracking particles towards the expected global pose.

VII. RESULTS

We present results for two different planning problems: Tower of Hanoi and Workspace Reach. The actions used in both of these problems are the ones described in Section V. We test the framework on a 7-dof Franka Panda fitted with a Robotiq 2F-85 gripper in simulation, and then demonstrate the Workspace Reach problem in the real world. Videos of the results can be found in the supplementary material.

A. Tower of Hanoi

In the Tower of Hanoi problem, there are three plates upon which towers of blocks can be formed decreasing in size from bottom to top. The task is to transfer a tower from a plate on the right side to one of two other plates: in the middle or on the left. For a tower of three blocks, a minimum of 14 actions is needed to complete the task. This problem suffers from long-term coupling between timesteps, since the placement of blocks at the bottom of the stack affects later blocks. Using relative pose variables, however, removes this coupling from the pick and place constraints.

Given a maximum tree search depth of 14, STRIPS planning finds two candidate action skeletons, each one transferring the tower to a different plate. Transferring the tower to the middle plate yields a lower optimization score than transferring to the left. This difference is also reflected in the robot's execution time for each action skeleton: the middle takes 25s while the left takes 28s.

The combined TAMP optimization takes 4.02s for this problem with IPOPT as the nonlinear solver. As shown in Table I, NLOPT failed to produce a feasible solution.

B. Workspace Reach

In this problem, the robot's goal is to place a box on a shelf, but the box is outside its workspace. The world also contains a hook within the robot's workspace that can be used to bring the box closer.

With a maximum search depth of 5, there are three candidate action skeletons of the following form: 1. $pick(hook)$ 2. $push(hook, box, table)$ 3. $place(hook, \cdot)$ 4. $pick(box)$ 5. $place(box, shelf)$. The three skeletons differ in where the hook gets placed after $push$: on the table, shelf, or box. Frame optimization reveals that placing the hook on the table minimizes the total distance travelled by the end-effector.

The combined TAMP optimization takes a total of 3.76s for this problem with IPOPT. Figure 5 shows the convergence
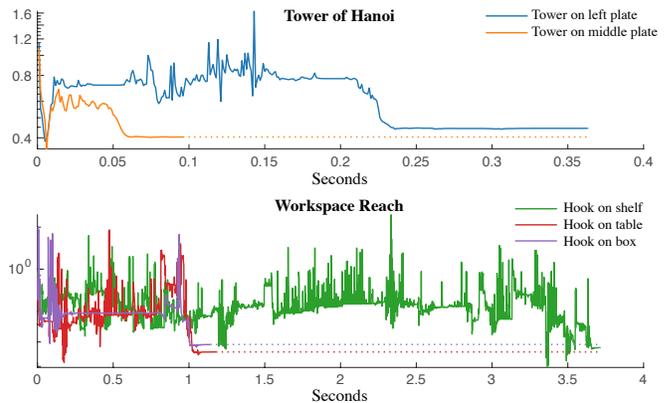

Fig. 5. Optimization convergence with IPOPT. Convergence is fast, but the highly nonlinear nature of the problem makes it susceptible to local minima. The spikes occur when the IPOPT algorithm tries to escape infeasible minima.

trends for the three different optimization subproblems, and Table I compares optimization scores and times between IPOPT and NLOPT. In general, we found IPOPT to produce more optimal results with fewer constraint violations. However, it is sensitive to the initial poses of objects in the environment; it can take up to a minute to find a feasible solution, if at all, depending on the problem instance.

VIII. CONCLUSION

In this paper, we proposed a new TAMP algorithm based on the optimization of Cartesian frames defined relative to target objects. By design, these plans can be easily integrated with reactive closed-loop controllers that can adapt to changes in the environment in real-time and handle imperfect perception and control. While previous works on TAMP are limited to simulation or tightly controlled environments with accurate perception, we demonstrate our approach working on a real robot with noisy perception from RGB-D cameras in an environment with moving objects subject to external disturbances.

One limitation of using Cartesian frames instead of joint configurations is that robot collisions and joint limits are difficult to encode. Our TAMP method only checks object-level collisions, relying on lower level motion planners or controllers to avoid collisions with robot links and joint limits. In extreme cases, however, this heuristic would not be sufficient, and an approach that combines Cartesian and configuration variables in an augmented representation as done in [11] may be more appropriate.

Although this paper treats the output of the TAMP optimizer as an open-loop plan, the planning would ideally run inside a closed loop (at a frequency on the order of seconds). This way, the optimal plan would be regularly recomputed with respect to new object poses during execution. Furthermore, this would allow the framework to respond to unexpected changes in the symbolic state beyond the scope of the reactive controllers, such as objects moving in and out of the workspace or the failure of a manipulation action during execution. Creating a perception system that could detect the symbolic state of the environment is a crucial area of future research.

This paper solves LGPs by combining STRIPS planning with Nonlinear Programming, but there is potential for simplifying the optimization further for increased efficiency and robustness, such as linearizing the constraints and using a quadratic objective [30] or decomposing the problem temporally and spatially into smaller subproblems [31].

Manipulation actions in our TAMP framework are defined by their STRIPS preconditions and postconditions, their control and target frames, and their constraint functions for optimization. The effort required to define new actions makes scalability an issue. Future work includes using learning-based approaches to alleviate the engineering bottlenecks, such as learning the preconditions and postconditions of the symbolic actions, as is considered by Huang et al. [32], or learning the constraint functions from demonstration.

REFERENCES

[1] J. H. Reif, "Complexity of the mover's problem and generalizations," in *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*. IEEE, 1979, pp. 421–427.

[2] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[3] E. Plaku and G. D. Hager, "Sampling-based motion and symbolic action planning with geometric and differential constraints," in *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 5002–5008.

[4] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *The International Journal of Robotics Research*, vol. 28, no. 1, 2009.

[5] F. Lagriffoul, D. Dimitrov, A. Saffiotti, and L. Karlsson, "Constraint propagation on interval bounds for dealing with geometric backtracking," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 957–964.

[6] F. Lagriffoul, D. Dimitrov, J. Bidot, A. Saffiotti, and L. Karlsson, "Efficiently combining task and motion planning using geometric constraints," *The International Journal of Robotics Research*, vol. 33, no. 14, pp. 1726–1747, 2014.

[7] J. Bidot, L. Karlsson, F. Lagriffoul, and A. Saffiotti, "Geometric backtracking for combined task and motion planning in robotic systems," *Artificial Intelligence*, vol. 247, pp. 229–265, 2017.

[8] L. de Silva, A. K. Pandey, M. Gharbi, and R. Alami, "Towards combining htn planning and geometric task planning," *arXiv:1307.1482*, 2013.

[9] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical planning in the now," in *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[10] F. Lagriffoul, L. Karlsson, J. Bidot, and A. Saffiotti, "Combining task and motion planning is not always a good idea," in *RSS Workshop on Combined Robot Motion Planning and AI Planning for Practical Applications*, vol. 4, no. 4, 2013, p. 6.

[11] T. Lozano-Pérez and L. P. Kaelbling, "A constraint-based method for solving sequential manipulation planning problems," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 3684–3691.

[12] J. Ferrer-Mestres, G. Frances, and H. Geffner, "Combined task and motion planning as classical ai planning," *arXiv:1706.06927*, 2017.

[13] 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 *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2014, pp. 639–646.

[14] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "Incremental task and motion planning: A constraint-based approach," in *Robotics: Science and systems*, AnnArbor, Michigan, June 2016.

[15] C. Zhang and J. A. Shah, "Co-optimizing task and motion planning," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 4750–4756.

[16] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[17] M. Toussaint, K. Allen, K. Smith, and J. B. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," in *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

[18] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Sampling-based methods for factored task and motion planning," *The International Journal of Robotics Research*, 2018.

[19] A. M. Wells, N. T. Dantam, A. Shrivastava, and L. E. Kavraki, "Learning feasibility for task and motion planning in tabletop environments," *IEEE robotics and automation letters*, 2019.

[20] A. Suárez-Hernández, G. Alenyà, and C. Torras, "Interleaving hierarchical task planning and motion constraint testing for dual-arm manipulation," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018.

[21] R. C. Jeroslow, "There cannot be any algorithm for integer programming with quadratic constraints," *Operations Research*, vol. 21, no. 1, pp. 221–224, 1973.

[22] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.

[23] M. Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[24] L. T. Biegler and V. M. Zavala, "Large-scale nonlinear programming using ipopt: An integrating framework for enterprise-wide dynamic optimization," *Computers & Chemical Engineering*, vol. 33, no. 3, pp. 575–582, 2009.

[25] S. G. Johnson, "The nlopt nonlinear-optimization package, 2014," *URL http://ab-initio. mit. edu/nlopt*, 2014.

[26] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE Journal on Robotics and Automation*, vol. 3, no. 1, 1987.

[27] M. Wüthrich, P. Pastor, M. Kalakrishnan, J. Bohg, and S. Schaal, "Probabilistic object tracking using a range camera," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 3195–3202.

[28] J. Issac, M. Wüthrich, C. G. Cifuentes, J. Bohg, S. Trimpe, and S. Schaal, "Depth-based object tracking using a robust gaussian filter," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 608–615.

[29] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, 2014.

[30] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints," *Automatica*, 1999.

[31] J. Jackson and I. Grossmann, "Temporal decomposition scheme for nonlinear multisite production planning and distribution models," *Industrial & engineering chemistry research*, 2003.

[32] D.-A. Huang, D. Xu, Y. Zhu, A. Garg, S. Savarese, L. Fei-Fei, and J. C. Niebles, "Continuous relaxation of symbolic planner for one-shot imitation learning," *arXiv:1908.06769*, 2019.