2017 IEEE International Conference on Robotics and Automation (ICRA)
Singapore, May 29 - June 3, 2017

Multi-Bound Tree Search for Logic-Geometric Programming in Cooperative Manipulation Domains

Marc Toussaint[1] Manuel Lopes[2]

***Abstract*— Joint symbolic and geometric planning is one of the core challenges in robotics. We address the problem of multi-agent cooperative manipulation, where we aim for jointly optimal paths for all agents and over the full manipulation sequence. This joint optimization problem can be framed as a logic-geometric program. Existing solvers lack several features (such as consistently handling kinematic switches) and efficiency to handle the cooperative manipulation domain. We propose a new approximate solver scheme, combining ideas from branch-and-bound and MCTS and exploiting multiple levels of bounds to better direct the search. We demonstrate the method in a scenario where a Baxter robot needs to help a human to reach for objects.**

I. Introduction

Planning manipulation sequences fundamentally involves both, reasoning about the smooth motion of all involved agents and objects as well as making categorial decisions about the type and order of manipulations and which objects are involved. Siméon et al. [14] was one of the first to pinpoint this combined geometric and logic structure of manipulation problems, which can also be viewed as a repeated alternation of piece-wise smooth paths and discontinuous kinematic switches [17].

The field of combined task and motion planning (TAMP) addresses this problem in the single agent settings and mostly from the perspective of finding feasible manipulation paths. Srivastava et al. [15] proposed a standardized interface between path finding algorithms (e.g. RRTs or PRMs) and a logic planner. [9], [5] presented a reduction to CSP methods and an adaptation of the FastForward planner to TAMP. These and similar approaches are impressive in terms of the demonstrated scaling to large number of objects. However, they rely on sampling in the configuration space (e.g., pre-sampling potential grasp configurations), which is good as it potentially inherits the convergence (probabilistic completeness) properties of sampling algorithms, but is less promising in terms of scaling to high-dimensional kinematics. Further, these methods focus on finding feasible manipulation sequences instead of optimal ones.

In contrast, in prior work [17] we proposed an optimization formulation of TAMP, on which our work builds. The formulation allows us to leverage non-linear mathematical programming (NLP) techniques to efficiently find smooth and locally optimal paths in high dimensional systems, while sampling is only used to search over the inherently discontinuous aspects of the problem. However, the concrete solver given in [17] is yet limited and tailored to situations where the so-called effective end-state kinematics is a good heuristic to explore manipulation sequences.

In this paper we address the problem of finding (near) optimal solutions to multi-agent (in our case, four manipulators) sequential manipulation problems with high-dimensional kinematics (in our case, 43 dimensions in total) where additional degrees-of-freedom (dofs) become subject to optimization whenever objects are manipulated. To tackle such problems we generalize the solver of [17] to account for multiple agents, and to exploit a hierarchy of bounds (as in branch-and-bound) that can be computed using NLP methods and allow us to prune branches of the search tree. The extension to the multi-agent setting is based on prior work on representing cooperative multi-agent manipulation processes as semi-MDPs [19].

The resulting method is less powerful in terms of scaling to many objects than the above mentioned feasibility approaches to TAMP, but scales well to the high-dimensional kinematics. The optimality formulation entails a series of sub-problems that have not yet been considered part of TAMP, but are fundamental and yet un-addressed challenges in itself. Our method equally tackles those problems in one coherent formulation:

(1) When grasping an object the grasp parameters (e.g., object-hand transformation) have a strong influence on the optimality of later actions with this object, e.g. when it is required to place the object upside-down, or when the object is used as a tool. In general, such long-term dependencies of optimization variables are non-trivial to formulate exactly in NLP formulations while still retaining a form that is efficient to solve. We propose a novel approach to represent such optimization variables by adding and deleting effective dofs to the configuration kinematics at different time slices, depending on the manipulation sequence. This is an exact formulation of optimizing such action parameters that correctly accounts for the long term dependencies while retaining the Markovian structure of the path optimization problem that is essential to ensure the linear-in-T complexity of computing Newton steps in the NLP.

(2) The path optimization method we exploit can be viewed as a standard optimal control method, e.g., as used in model-predictive control (MPC). However, we extended the solver to handle optimization across kinematic switches, where the kinematics and configuration space dimensionality may vary across time steps. To our knowledge, this is the first optimal control method we are aware to handle this case.

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013 and by the EU FP7-ICT project 3rdHand under grant agreement no 610878.

[1]Machine Learning and Robotics Lab, University of Stuttgart, Germany. marc.toussaint@informatik.uni-stuttgart.de

[2]INESC-ID, Instituto Superior Técnico, Universide de Lisboa, Portugal. manuel.lopes@tecnico.ulisboa.pt

After discussing related work we first recap the logic-geometric programming framework and the present in detail our novel solver scheme based on multiple levels of bounds. We then discuss the specific path optimization methods used and the extension to the multi-agent setting before reporting on experiments.

II. Related Work

Concerning Combined Task and Motion Planning (TAMP), a number of approaches [9], [8], [7] rely on a discretization of the configuration space or action/skeleton parameter spaces to leverage CSP methods. Siméon et al. [14] describe complex, multi-interaction planning of the manipulation of a single object, but does not bridge to relational/logic representations of environments with many objects. Others [15], [5] devise a symbolic description that includes predicates to abstract geometric feasibility conditions and represent action operator preconditions on the symbolic level. For a given task plan, the predicates are evaluated on demand, as well as the `Obstructs` predicate added depending on which objects make a path finder fail. Such backtracking depending on geometrical reasoning is also the core idea in [11], [3], [1]. To our knowledge our prior work [17] is the first to propose a full optimization formulation of TAMP.

Concerning multi-agent cooperative manipulation, in [19] we presented a reduction of concurrent multi-agent decision processes to semi-MDPs. This work considered only the symbolic level—here we focus on the geometric optimality. [4], [6] describe work on multi-robot cooperative assembly, similar to our setting. However, the symbolic planning and geometric execution phases are rather decoupled, and the system does not aim for (locally) optimal concurrent robot manipulation paths.

Path optimization across manipulation sequences has been demonstrated by [10] based on a contact-invariant optimization approach that originated in locomotion research. Equally impressive, but not aiming at sequential manipulation planning, are recent methods on trajectory optimization through contacts [12]. Both methods, however, do not address optimization over paths where the kinematics and configuration space really changes, as it is the case in our formulation, and ensure that effective dofs (or skeleton parameters) are jointly optimized with the full multi-agent path. They also neglect symbolic search over alternative manipulation sequences.

III. Logic-Geometric Programming

We recap the Logic-Geometric Programming (LGP) formulation of sequential manipulation of [17]. As a starting point, let us recap the notion of 'kinematics': Given a system with configuration space $\mathcal{X}$, the system kinematics describe all possible paths of motion in $\mathcal{X}$. We may more concretely define it as the collection of tangent spaces

$$T_x\mathcal{X} = \{\dot{x} \mid \dot{x} = \dot{x}(x,u),\ u \in \mathbb{R}^n\}$$

where u are some controls that articulate the system.

In our case we are concerned with m rigid objects and n-articulated joints of potentially multiple agents; the configuration space is $\mathcal{X} \in \mathbb{R}^n \times SE(3)^m$. However, not all paths are possible in this space: only n joints are articulated, and the maximally n-dimensional tangent space $T_x\mathcal{X}$ depends on which objects and manipulators are in contact or connected in the configuration x. We want to capture the state space kinematics in terms of path constraint functions

$$h_{\text{path}}(x,\dot{x}) = 0\ , \quad g_{\text{path}}(x,\dot{x}) \le 0\ ,$$

which must hold for any x and imply $T_x\mathcal{X}$.

When during manipulation a contact or connection is created or destroyed, this implies a *discontinuity* in the constraint functions $h_{\text{path}}, g_{\text{path}}$: e.g., the row space of their Jacobians instantly changes to span other dimensions, which can also be viewed as a flip of the tangent space $T_x\mathcal{X}$. In our view, these discontinuities are the core of why sequential manipulation optimization is hard, in particular the combinatorics implied by such discontinuities.

In this paper we use a first-order logic language $\mathcal{L}$, similar to PDDL, to describe kinematic structure. This means that we require two properties to hold: First, we have a mapping from every configuration x to a discrete relational state[1] $s(x) \in \mathcal{L}$ such that the constraint functions h, g

$$h_{\text{path}}(x,\dot{x} \,|\, s(x)) = 0\ , \quad g_{\text{path}}(x,\dot{x} \,|\, s(x)) \le 0$$

are smooth in $x, \dot{x}$ for a constant $s(x)$. This describes a partitioning of the configuration space. Second, there exist first order rules (e.g., PDDL-like) that enumerate all possible successor states $s_k \in \text{succ}(s_{k\text{-}1})$, that is, all possible *kinematic switches* from s. This describes the connectivity of configuration space partitions. We further assume that the boundary between two partitions (which corresponds to a kinematic switch such as creating/destroying a contact/connection) can be described by smooth constraint functions

$$h_{\text{switch}}(x(t_k) \,|\, s_k, s_{k\text{-}1}) = 0\ , \quad g_{\text{switch}}(x(t_k) \,|\, s_k, s_{k\text{-}1}) \le 0\ .$$

In our experiments we will use a PDDL-like logic modified to represent concurrent cooperative manipulation domains, as described in [19], which contains predicates `grasp`, `place`, `handover` that imply geometric constraints as described in Sec. V-C.

In essence, we introduced a relational state to make the conditional problem smooth, leading to piece-wise smooth paths while $s_k \in \mathcal{L}$ is constant, and categorial decisions about kinematic switches $s_k \in \text{succ}(s_{k\text{-}1})$. Based on this, the overall sequential manipulation optimization problem can be formulated as a Logic-Geometric Program of the form

$$\begin{aligned}
\min_{x, s_{1:K}, t_{1:K}} \quad & \int_0^T c(x(t),\dot{x}(t),\ddot{x}(t))\ dt + f_{\text{goal}}(x(T)) \\
\text{s.t.} \quad & h_{\text{goal}}(x(T)) = 0,\ g_{\text{goal}}(x(T)) \le 0 \\
& \forall_{t\in[0,T]}\ h_{\text{path}}(x(t),\dot{x}(t) \,|\, s_{k(t)}) = 0 \\
& \forall_{t\in[0,T]}\ g_{\text{path}}(x(t),\dot{x}(t) \,|\, s_{k(t)}) \le 0 \\
& \forall_{k=1}^{K}\ h_{\text{switch}}(x(t_k) \,|\, s_k, s_{k\text{-}1}) = 0 \\
& \forall_{k=1}^{K}\ g_{\text{switch}}(x(t_k) \,|\, s_k, s_{k\text{-}1}) \le 0 \\
& \forall_{k=1:K}\ s_k \in \text{succ}(s_{k\text{-}1}) \\
& s_K \models \mathfrak{g}_{\text{goal}}
\end{aligned}$$

[1]By relational state we denote a conjunction of grounded literals.

Here, $c(x, \dot{x}, \ddot{x})$ are typical control costs, and the terms with subscript $_{\text{goal}}$ specify (optional) goal aspects, including geometric costs and constraints and a symbolic goal constraint $s_K \models \mathfrak{g}_{\text{goal}}$ (e.g., which contacts/connections are to be established). The path and switch constraints are as above. Finally we mention that an LGP differs from a mixed-integer NLP in the same way that a PDDL problem differs from an integer program.

IV. A Multi-Bound Tree Search Approach to Solving LGPs

A. Multi-Bounds as search heuristics

[17] considered a basic solver for LGPs that focuses only on the final effective kinematics to decide on candidate symbolic sequences. Only for the best such symbolic sequences the full sequence and path optimization is performed.

This is not sufficient to solve the more complicated problems considered in this paper, where geometric cost estimates of decisions are essential to guide tree search early on. Tree search should be organized in a way that decisions which are found to be geometrically infeasible or very costly are avoided during tree search. For instance, in our example domains there could be ~ 20 possible decisions in the start state, while geometrically only about 4 of them are feasible. Interweaving geometric feasibility and costs systematically in tree search can therefore significantly reduce the branching factor.

The approach we take here is a mixture of basic ideas from branch-and-bound, admissible heuristics of A*-search, and Monte-Carlo Tree Search. We start by defining our notion of a lower bound.

A NLP $\mathcal{P} = (f, g, h)$ is a tuple of a cost function f and two constraint functions (g, h). An NLP $\hat{\mathcal{P}}$ is a lower bound of another NLP $\mathcal{P}$ iff

$$\mathcal{P} \text{ feasible } \Rightarrow \hat{\mathcal{P}} \text{ feasible } \wedge \ \hat{f}^* \leq f^* \ , \qquad (1)$$

where $f^* = \min_x f(x)$ s.t. $g(x) \leq 0, h(x) = 0$ is the optimum of $\mathcal{P}$, and analogously for $\hat{f}^*$.

In the LGP, every terminating symbolic sequence $s_{1:K}$ with $s_K \models \mathfrak{g}$ implies a remaining NLP $\mathcal{P}(s_{1:K})$. To guide search we are not only interested in bounds given a full terminating sequence $s_{1:K}$, but also in *sequential* bounds, which roughly state that sub-sequences must be cheaper. Let $\mathcal{P}(s_{1:k})$ assign an NLP to any symbolic sub-sequence $s_{1:k}$. We say that $\mathcal{P}$ bounds itself sequentially iff, for any symbolic sub-sequence $s_{1:k}$ and any possible continuation $s_{k+1:K}$, $\mathcal{P}(s_{1:k})$ is a lower bound of $\mathcal{P}(s_{1:K})$, that is,

$$\begin{aligned} &\mathcal{P}(s_{1:K}) \text{ feasible } \Rightarrow \\ &\qquad \mathcal{P}(s_{1:k}) \text{ feasible } \wedge \ f^*(s_{1:k}) \leq f^*(s_{1:K}) \ . \end{aligned} \qquad (2)$$

In words, if $\mathcal{P}$ finds that the sub-sequence $s_{1:k}$ leads to an infeasible NLP, then there can be no continuation sequence $(s_{1:k}, s_{k+1:K})$ that would lead to a feasible solution. Further, the optimal cost of a sub-sequence is less than the optimal cost of any continuation sequence.

Finally, we call a set of bounds $(\mathcal{P}_1, .., \mathcal{P}_L)$ a multi-bound, if for each i, $\mathcal{P}_i$ is lower bound of $\mathcal{P}_{i+1}$, each $\mathcal{P}_i$ bounds itself sequentially, and $\mathcal{P}_L$ is the full NLP (path cost) of the original LGP.

In the case of sequential manipulation it is natural to construct such multi-bounds: Concerning the sequentiality of bounds, it is clear that the cost of grasping and placing something must be greater than that of grasping only. Concerning the bound levels $i = 1, .., L$, we detail below how exactly we choose them. However, there is a simple generic way to construct such bounds based on the following trivial observation:

Lemma 1: Given an NLP $\mathcal{P}$ with an objective function $f(x)$ that is a *sum of positive terms*. It holds that (i) dropping terms from f leads to a lower bound $\hat{\mathcal{P}}$, and (ii) dropping constraint function terms from g or h leads to a lower bound $\hat{\mathcal{P}}$.

Proof: (ii) Dropping a constraint term, the feasible set of $\hat{\mathcal{P}}$ is a superset of that of $\mathcal{P}$. As f is unchanged, the optimum can only decrease or stay equal. (i) As g, h are unchanged x^* is also feasible for $\hat{\mathcal{P}}$. As $\hat{f}$ drops a positive term, $\hat{f}(x^*) \leq f(x^*)$ and therefore also $\hat{f}^* \leq f^*$. ■

In the LGPs we consider, the objective function $f(x)$ is always a sum-of-squares. Therefore, our approach to constructing a series of bounds $\mathcal{P}_1, .., \mathcal{P}_L$ is to drop terms. One special case of dropping terms it to choose a course time-discretization of the path:

Corollary 1: If $x = x_{1:T}$ is a path and $\mathcal{P}$ an NLP over the path. Let $\hat{x} = x_{1:\hat{T}}$ be a sub-sample of the path, with coarser time discretization. If all constraint terms depend only on individual time slices x_t, and those cost function terms that depend on higher-order tuples (velocities, accelerations) guarantee $f(\hat{x}) \leq f(x)$, then the NLP over the coarse path $\hat{x}$ which drops the corresponding single-time slice terms is a lower bound of $\mathcal{P}$.

The specific choice of $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ we use in the experiments will follow exactly these constructions.

B. Multi-Bound Tree Search (MBTS)

Our algorithm builds a tree of nodes, each node n containing:

- the list of child nodes
- the symbolic sub-sequence $s_{1:t}(n)$ of relational states from root to this node,
- the ten best returns $R_j(n)$, $j = 1, .., 10$ from MC rollouts, where $R_j \leq R_{j+1}$ through this node (see details below)
- the series $\mathcal{P}_i(n)$, $i = 1, .., L$ of NLPs, where $\mathcal{P}_i(n) = \mathcal{P}_i(s_{1:t}(n))$, and $\mathcal{P}_{i=1}$ is the coarsest bound and $\mathcal{P}_{i=L}$ the original full LGP
- for each i, whether $\mathcal{P}_i(n)$ has been given to an optimizer yet ($a_i(n) \in \{0, 1\}$), whether the optimizer found a feasible solution ($b_i(n) \in \{0, 1\}$), and the minimum found by the optimizer $f_i(n) \in \mathbb{R}$

While a parallelised implementation would improve computation time, for simplicity we chose a round robin scheme to schedule which computations are made in each round. Specifically, in each round the algorithm:

1) selects k_E leaf nodes (detailed below) to be expanded
2) selects k_R leaf nodes to contribute a new MC rollout
3) and for each $i = 1, .., L$, selects k_i nodes (if adequate ones exist, detailed below) to pass the NLP $\mathcal{P}_i(n)$ to

an optimizer

The numbers $k_{E,R,1,..L}$ determine how much computation time we dedicate to the various heuristics. In our experiments we chose $k_E = k_2 = k_3 = 1$, $k_1 = 5$ and $k_R = 50$.

In this scheme there are two essential ingredients to the algorithm: how exactly the nodes are selected in each round, and how detected infeasibilities (when a $\mathcal{P}_i(n)$ is found infeasible) feed back to the tree itself to prune branches.

Concerning node selection, in 1) and 2) we use the same soft-max tree policy to choose a leaf node. We descend the tree sampling the child c according to

$$p(c) \propto \exp\{-\beta R_1(c)\}$$

where $R_1(c)$ is the best return found by MC rollouts through child c, and β a temperature. We choose $\beta = 2$, where returns are typically in the range $[2, 10]$ (returns are costs that we minimize).

Selecting nodes in 3) for optimization could also be described in terms of tree policies, but we find it simpler to present the selection process in terms of prioritized candidate queues. For each bound level i we maintain a list of candidates that are adequate to be passed to the NLP solver. Depending on the bound we may require that a node is adequate for optimization only when its parent has been optimized before (we impose this on the level of pose optimization $\mathcal{P}_1$); or when the node is a symbolic terminal node (we require this for the sequence level $\mathcal{P}_2$ and the finest/full level $\mathcal{P}_3$). These rules define current sets of candidate nodes for each optimization level i. These sets are now prioritized simply by $f_{i\text{-}1}(n)$, that is, the minimum found by the next coarser optimization level.

C. Bound generalization across branches

To motivate another mechanism we first give an example. When agent A aims to grasp a screwdriver which is initially out of reach, the above method quickly finds that directly reaching for the screwdriver is infeasible, e.g., with $\mathcal{P}_1(n)$ where n represents the direct grasp. However, this prunes only the branch that *starts* with the direct grasp. Combinatorially many other branches exist which have the direct grasp in second or third place, with fully unrelated actions in the first step, e.g., agent B first grasping for an apple, then A for the screwdriver. Clearly we know that this is still infeasible. The bound $\mathcal{P}_1(n)$ (e.g., its infeasibility) should therefore transfer to all branches that include the direct grasp *if* the screwdriver or agent A has not been moved by a preceding action.

For simplicity we adopted the approach of [15] to generate such a type of generalization of bounds across branches, which however is only able to generalize the symbolic knowledge of infeasibility. Future research should try to generalize also the optimistic costs. The approach of [15] artificially introduces an `infeasible` predicate of the respective decision in the relational state that blocks the precondition of the decision. An interesting issue is where exactly the predicate is introduced: introducing it at the root state may be incorrect as it might have been a later action that rendered a decision infeasible (e.g., the object was placed on a distant table), and therefore the infeasible predicate only holds after that action. We insert the infeasible predicate at the tree node that last manipulated the object related to n, when $\mathcal{P}_i(n)$ finds an infeasibility.

Inserting an infeasible predicate somewhere in our search tree implies a change of the structure of the tree (some branches become symbolically unreachable) and therefore the symbolic returns from MC rollouts have to be recomputed. This is the reason for why we explicitly store the ten best MC returns at each leaf: In the respective branches we delete all MC scores in internal nodes and backup all explicitly stored returns from reachable leafs to the internal nodes, without need to repeat these rollouts.

V. Path Optimization and the Specific Multi-Bound Levels of Approximation

A. Path optimization and kinematic switches

The MBTS algorithm out-sources sub-problems $\mathcal{P}_i(s_{1:k})$ to an NLP solver. These sub-problems are path optimization problems that try to find a feasible and optimal path consistent with the first k symbolic decisions. We describe the specific bound approximations made for $i = 1, 2, 3$ below. Here we describe the path optimization problem and solver used. It turns out that correct and efficient path optimization across kinematic switches raises a number of interesting issues.

As backbone we use the k-order Motion Optimization (KOMO) method described in [18], which addresses problems of the form

$$\begin{aligned} \min_{x_{0:T}} \quad & \sum_{t=0}^{T} f_t(x_{t-k:t})^\top f_t(x_{t-k:t}) \\ \text{s.t.} \quad & \forall_t : \; g_t(x_{t-k:t}) \le 0 \;, \quad h_t(x_{t-k:t}) = 0 \;. \end{aligned} \tag{3}$$

Here, x_t is the system configuration (not phase state) in time slice t, and $x_{t-k:t}$ is a k+1 tuple of consecutive configurations. This form assumes that the cost and constraint terms may only depend on such k+1 tuples of consecutive configurations (e.g., on finite difference velocities and accelerations for $k = 2$). The cost function is assumed to be a sum-of-squares.

This particular formulation shares with other path optimization methods (e.g., iLQG [16], DDP [2], CHOMP [13]) that (quasi-) Newton steps can be computed in complexity *linear in* T, due to the bandedness of the Hessian, see details in [18]. KOMO additionally handles constraints using the Augmented Lagrangian method.

Below we specify how symbolic decisions $s_{1:k}$ translate to specific objective and constraint functions that reflect the grasps, placements and hand-overs implied by $s_{1:k}$. However, a more fundamental issue beyond these specific objectives is how to handle kinematic switches accurately and generically within this framework.

Consider a sequence where a robot picks up a screwdriver, then places it on a table, then a human picks up the screwdriver. On the first grasp a rigid connection is created between the robot end-effector and the screwdriver (literally switching the kinematic tree); however, the parameters of this

connection are subject to optimization and heavily influence the costs that arise when the robot places the screwdriver again. Similarly, the parameters of where the screwdriver is placed are subject to optimization and heavily influence the costs that arise when the human grasps the screwdriver.

Approached naively, this raises issues in the context of KOMO and any other methods that exploits banded Hessians and local gradients, because the costs depend non-locally on configurations: early (grasp) configurations have influence on costs that appear much later in a (placing) configuration. Technically, path optimizers require correct gradients and the question arises how gradients are correctly "propagated along" the trajectory to account for non-local effects of such kinematic switches.

B. Effective kinematics and two complementary zero velocity constraints to ensure correct gradient propagation

Our approach is to optimize paths over *effective kinematics*[2] as follows. When an object is grasped we introduce a 6D *free* joint between object and end-effector in the kinematics, thereby increasing the configuration space dimension; when an object is placed on a table we introduce a 3D $xy\varphi$-joint (planar translation and rotation) between object and table. These novel dofs represent what is classically called *action parameters* [9], the geometric parameters of a grasp or placement. Introducing them as effective dofs in the kinematics allows us to optimize over them consistently and jointly with the overall path. Technically we had to extend the KOMO code to deal with the fact that the configuration state space may frequently change dimensionality (while transition costs, e.g., are still well-defined), and that the effective dofs behave as expected. We achieve this by introducing two complementary equality constraints:

(1) *Effective joints are non-actuated and constrained to zero velocity:* All created and destroyed effective joints are not articulated by true motors. Therefore, they must remain fixed throughout their existence: a screwdriver placed on the table remains fixed relative to the table; an object grasped remains fixed relative to the hand. However, note that in the first time slice of the joint existence, the respective dofs are "free" and subject to optimization: where the screwdriver is placed on the table, or how the object is placed into the hand; equally the dofs in the last time slice of the joint existence are subject to optimization: where the screwdriver is picked from the table; how the object is positioned in the hand when releasing it. The zero velocity constraint implies that the start and end relative poses are optimized subject to being equal. This is the underlying key of how gradients of long-term geometric dependencies are correctly propagated while still being conforming to our KOMO framework and the implied banded Hessian that leads to efficient Newton steps. However, this constraint does not ensure consistency at creation/destruction of effective joints: objects would still "jump".

(2) *(De-)linked bodies have same relative velocities:* In time slice t we detect all joints present in time slice t-1 but not in time slice t and vice versa (all created and destroyed joints). For each pair of bodies (i, j) for which a joint is created or destroyed we compute their difference in linear velocity $[p_i(t) - p_i(t\text{-}1) - p_j(t) + p_j(t\text{-}1)]/\tau$ as well as their difference in angular velocity (derived properly from their quaternion finite differences). We constrain these velocities to be zero.

Note that the first constraint is formulated in configuration space, while the second is in the involved objects' pose spaces. These two complementary constraints ensure consistent handling of switching kinematics.

Numerically one might be concerned that zero-velocity constraints on effective joints are imprecise and small errors accumulated along the existence of an effective joints could lead to significantly different start and end poses. However, this turns out not to be the case: We are computing exact Gauss-Newton steps which is, as shown in [18] analogous to Dynamic Programming (Riccati sweeps) and computing exact (subject to the local LQ approximation) cost-to-go functions. With respect to the effective joints this means that the start configuration of an effective joints "sees" the exact cost-to-go function w.r.t. the equality constraint and therefore w.r.t. the end configuration. In this sense, there are no "errors accumulating along the zero-velocity constraint". The iterates correctly compute gradients and Hessians across time and kinematic switches.

C. Modeling grasp, place, and hand-over constraints

The path constraints $h_{\text{path}}, g_{\text{path}}$ in our LGP include joint limit and collision constraints as well as the just explained generic constraints for consistency across kinematic switches. We now explain the specifics of how we defined the switch constraints $h_{\text{switch}}, g_{\text{switch}}$ conditional to a grasp, place and handover decisions in our experiments.

A decision `grasp(t,e,o)` states that a time slice t end-effector e wants to grasp object o. This represents a switch from $s_{k\text{-}1}$ to s_k where a joint from table to object is destroyed and an effective ball joint between the end-effector's grasp center and object created. Geometrically it additionally imposes

- a small-weighted sum-of-squares costs to align the end-effector vertically downward,
- sum-of-squares costs to enforce downward and upward velocity just before and after the grasp.

Note that the effective ball joint identically enforces the object to be positioned in the grasp center after the grasp; and the 2nd consistency constraint transfers this to before the grasp.

A decision `place(t,e,o,p)` states that a time slice t end-effector e wants to place object o onto p. This represents a switch from $s_{k\text{-}1}$ to s_k where the effective joint between hand and object is deleted and a new effective 3D $xy\varphi$-joint between table and object is created. Geometrically it additionally imposes

- end-effector vertical alignment and up and down velocities as for `grasp`,
- inequality constraints (four of them for a table) that indicate whether the object's center is within the table's support.

[2]The concept was introduced only for final configurations in [17]. Here we generalize it to kinematic switches across the path

Note that the $xy\varphi$-joint identically enforces the object touching the table after the placement; and the 2nd consistency constraint transfers this to before the placement.

A decision `handover(t,e1,o,e2)` is the straightforward combination of grasp and place, where the object is placed into the 2nd end-effector $e2$ and a new effective 6D joint between $e2$ and o is created. Note that in this case it is perfectly feasible that the handover is happening *in flight:* The 2nd consistency constraint only requires the relative velocities $e1$–o and $e2$–o to be equal.

D. Used path optimization bounds

We use three levels $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$. The finest level $\mathcal{P}_3$ is the full path optimization with fine time steps that reflects the original LGP. The next courser level $\mathcal{P}_2$ is exactly the same KOMO problem as $\mathcal{P}_3$, but with a much coarser time resolution, namely only two time steps[3] for one symbolic decision. This means that $\mathcal{P}_2$ only optimizes over the sequence of key frames plus one intermediate frame. E.g., a manipulation sequence with 5 manipulation decisions amounts to an NLP over only 10 configurations x_t, which is comparably fast to optimize. The coarsest bound $\mathcal{P}_1(n)$ does not optimize over sequences at all and relies on the notion of effective kinematics. It optimizes two configurations: the one associated with node n and its predecessor configuration associated with the parent node. Importantly, the parent configuration is optimized over its effective kinematics, that is, assuming all effective joints to be articulable, fully neglecting the costs and feasibility to actually reach such a configuration with a previous path. E.g., it assumes that in the parent configuration a screwdriver has been ideally placed on its table to now being grasped by an agent, neglecting the costs that a previous action raises to place it like this. This bound is particularly optimistic and focuses on quickly evaluating feasibility of n, presuming optimal preparation of previous actions.

VI. Multi-Agent Cooperative Manipulation

Our description of methods above did not mention multi-agent aspects specifically. Using the appropriate problem representations, all methods directly transfer.

Concerning the path optimizations, we consider the system of a human and a robot, each with two end-effectors, as a single kinematic system with 43 dimensions. I.e., all paths and configurations are optimized in the full-dimensional configuration space. From the perspective of KOMO it makes no difference whether the system's path constraints have single- or multi-agent semantics.

Concerning the logic $\mathcal{L}$ which enumerates possible kinematic switches (actions) we have to introduce a relational domain where all objects and each agent (the four end-effectors) are constants. We adopt the RAP formulation of [19] to formulate concurrent cooperative manipulation processes on the symbolic level.

Note that our methods naturally leads to multi-agent paths with concurrent movement of all agents. E.g., if a decision states that at time $t = 1$ the robot grasps an object, while at time $t = 3$ the human grasp another object, then both start moving already from $t = 0$ on as this is the optimal overall movement of both agents. The same is true when they jointly and concurrently move through a handover.

[3]We required two time steps instead of just one in order to define transition costs consistently with $\mathcal{P}_3$ and Corollary 1.

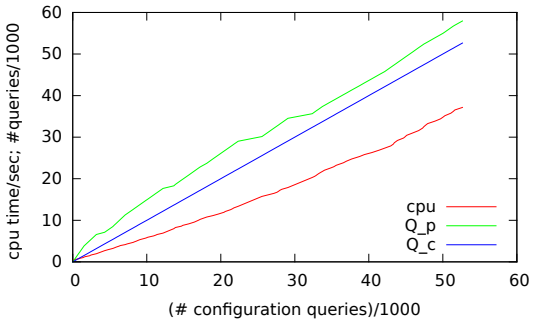

Fig. 1. Evaluation metrics: Relation between CPU time, *path queries* Q_p, and *configuration queries* Q_c. We will use $Q_c/1000$ to report on the later results.

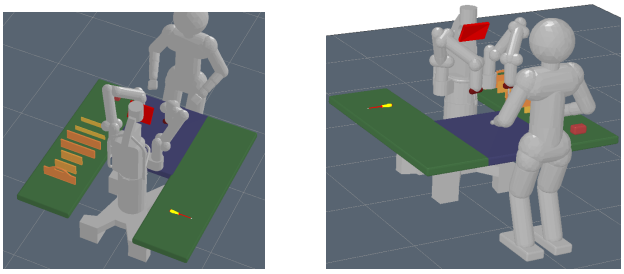

Fig. 2. The scene we consider in our evaluations. A humanoid and Baxter robot with two manipulators each (43 articulated joints in total) are located around three tables. A screwdriver is placed far back on the right (w.r.t. Baxter) table, and several pieces that make a wooden IKEA box on the left table.

VII. Experiments

A. Metrics

As performance metric we considered the number of *path queries* Q_p, i.e., how often a path x has been evaluated, including $f(x), g(x), h(x)$ and their Jacobians, within line search or Newton steps. As the length of paths varies depending on the heuristic (sequence vs. full path) and tree depth, we also considered the number of *configuration queries* Q_c, i.e., how often the forward kinematics and all task variables were computed from a configuration x_t. Fig. 1 shows the relations between these query metrics and CPU time. We find that CPU time is strongest correlated with Q_c, with about 0.698ms CPU time per configuration query; therefore, $Q_c/1000$ is a slight overestimate of CPU time in seconds. Configuration queries are rather slow in our code compared to others, mainly because we use not particularly efficient distance computations using SWIFT++ for all convex hulls. We will use $Q_c/1000$ as main metric to report on results.

B. Path optimization across kinematic switches

The scene we consider throughout our evaluations is given in Fig. 2. We first briefly demonstrate the path optimization

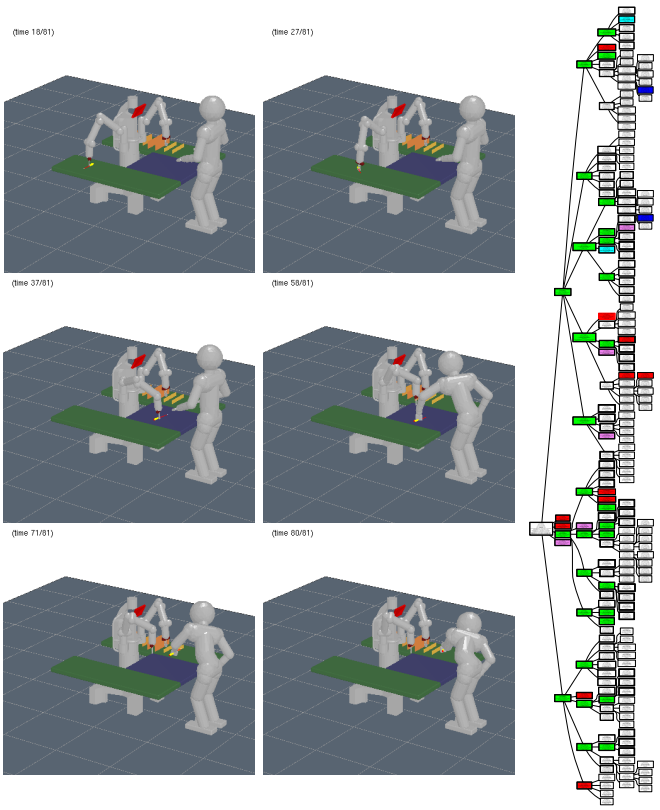

Fig. 3. LEFT: Example sequence of an optimization across kinematic switches where the robot places the screwdriver optimally for the human to be grasped and placed again. RIGHT: A search tree generated after ∼20sec. Please zoom the pdf to see details. Coarsely: The root node is on the left, red nodes are labeled infeasible (the infeasible predicate is annotated in respective ancestor), $\mathcal{P}_2$-feasible nodes are green, symbolically terminal nodes ($s_K \models \mathfrak{g}$) are blue; $\mathcal{P}_2$-feasible and symbolically terminal are cyan and passed to the last level $\mathcal{P}_3$. At this stage, two feasible manipulations (cyan nodes) are found. Nodes scheduled for $\mathcal{P}_2$ are double framed.

$\mathcal{P}_3$ across kinematic switches for a pre-specified manipulation sequence: Baxter grasps the screwdriver from the right table with its right hand, places it on the center table, the human grasps it with its left hand, then places it on the left (from Baxter) table. Fig. 3(LEFT) illustrates the resulting sequence. The sequence optimization $\mathcal{P}_2$ required 0.47sec with $Q_c = 1020$, while full path optimization $\mathcal{P}_3$ required 2.1sec with $Q_c = 3280$. We also tested a direct handover from Baxter to human, which requires one step less for this task. While computation time was comparable the total cost reduced from 3.90 to 3.57. We are not aware of a comparable method that can generate such consistently optimal paths of multi-agent sequential manipulation across kinematic switches.

C. Getting a screwdriver that is initially unreachable

We applied our MBTS algorithm on the same problem instance, now without prior knowledge of a feasible manipulation sequence. Fig. 3(RIGHT) displays the tree that is generated in the first ∼20sec; the pdf can be zoomed if desired. The tree illustrates the attributes associated with nodes. Fig. 4 displays what solutions are found by MBTS, overlaying 5 randomized trials. All trial evaluated between

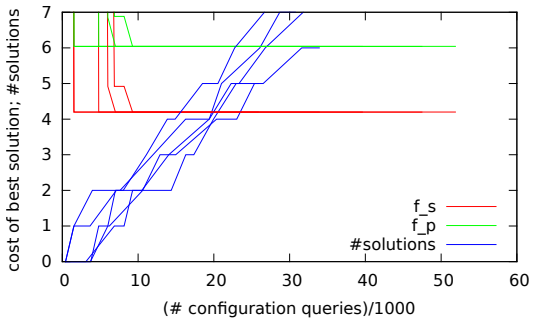

Fig. 4. Performance overlaying 5 randomized trials. MBTS reliably finds optimal manipulation sequences within the first 10 000 queries (<10sec). Later search finds more solutions, longer sequences that are non-optimal.

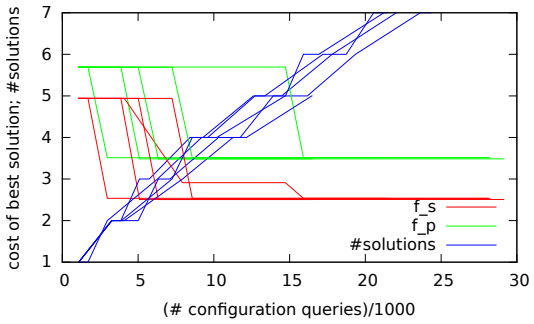

Fig. 5. Performance of 5 trials, where the screwdriver can directly be reached (early solution found), but a longer sequence yields less cost.

20 and 30 geometric sequences ($\mathcal{P}_2$) and 5-12 fine paths ($\mathcal{P}_3$). Randomization enters MBTS at two places: the softmax tree policy and random rollouts of MC are randomized, and the initialization of all path optimization problems is randomized.

D. Getting a screwdriver that is costly to reach

As a similar experiment we placed the screwdriver more to the front of the right table, where it is reachable by the human but at high costs. The optimal manipulation sequence is for the robot to place it closer to the human. Fig. 5 shows that MBTS reliably first finds the sub-optimal solution (in <1sec), and later (<20sec) the optimal solution.

E. Getting a distant screwdriver and placing a box

In the same domain we consider the target of grasping the screwdriver, which is out of reach (the robot has to place it first), and placing the screw-box in the center table. The optimal sequence requires 5 manipulations. Fig. 6 shows that MBTS requires more computation (∼50k configuration queries), but finds a (locally) optimal and concurrent path for all agents to achieve the task. The accompanying video displays these multi-agent manipulation sequences.

VIII. Discussion

In comparison to existing sampling-based approaches to TAMP our proposed method has limitations. *If* we could solve each NLP $\mathcal{P}_i(s_{1:k})$ (for any symbolic decisions $s_{1:k}$ and approximation level i) exactly, the MBTS approach itself

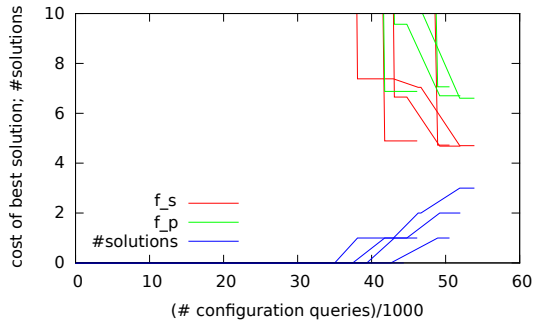

Fig. 6. Performance of 5 trials for the full far-screwdriver-and-place-box task.

would inherit an A-star like proof of optimality. However, the NLPs we deal with here are non-convex, non-linear programs, and the KOMO solver we use aims for fast downhill Newton steps into a local optimum. This local optimality of our geometric optimizations breaks the completeness and global optimality of our methods—as is typically the case for optimization-based vs. sampling-based path planning approaches. However, we think that a lack of global optimality guarantees does not render path optimization methods superfluous. Instead, they have shown to be effective especially in high-dimensional kinematics and where proneness to local optima can be alleviated by random restarts.

Our approach to TAMP is very much build up from a path optimization perspective and therefore formulates TAMP as a mathematical program, in contrast to the mentioned existing TAMP methods. While it cannot guarantee globally optimal solutions, that fact that it returns a locally optimal path, including all geometric action parameters of the sequence, is a feature that these existing methods lack.

The method aims for efficiency in high-dimensional kinematics. The demonstration in Sec. VII-D considers a rather complex sequential manipulation task, with four manipulators executing 11 actions in a 43-dimensional kinematic system, which is solved with about 50k configuration queries, including the search over about a hundred alternative symbolic sequences.

While the title and abstract highlight the LGP formulation of multi-agent TAMP and the MBTS approach as core contributions of this paper, the methods we developed to efficiently solve the conditional NLPs $\mathcal{P}_i(s_{1:k})$ are essential to this work. In the path optimization and optimal control literature, correct optimization across kinematics switches, i.e. correctly handling the temporally non-local effect of geometric action parameters, has hardly been addressed in general. The proposed concept of effective dofs (implying variable configuration space dimensionality over time) and the two respective consistency constraints (Sec. V-B) are an elegant way to correctly represent such problems and were crucial to develop a generic solver for the conditional NLPs. The proposed notion of effective kinematics was also essential to define the coarsest and most efficiently pruning bound $\mathcal{P}_1$, as well as the heuristic used in our earlier work [17]. In our view, these are contributions to the field of path optimization in itself, independent of TAMP.

The source code for the experiments can be found on the author's webpage.

References

[1] S. Alili, A. K. Pandey, E. A. Sisbot, and R. Alami. Interleaving symbolic and geometric reasoning for a robotic assistant. In *ICAPS Workshop on Combining Action and Motion Planning*, 2010.

[2] David Jacobson and David Mayne. *Differential Dynamic Programming*. 1970.

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

[4] M. Dogar, A. Spielberg, S. Baker, and D. Rus. Multi-robot grasp planning for sequential assembly operations. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 193–200. IEEE, 2015.

[5] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling. FFRob: An efficient heuristic for task and motion planning. In *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2014.

[6] R. A. Knepper, T. Layton, J. Romanishin, and D. Rus. Ikeabot: An autonomous multi-robot coordinated furniture assembly system. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 855–862. IEEE, 2013.

[7] 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*, 2014.

[8] F. Lagriffoul, D. Dimitrov, A. Saffiotti, and L. Karlsson. Constraint propagation on interval bounds for dealing with geometric backtracking. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 957–964. IEEE, 2012.

[9] T. Lozano-Pérez and L. P. Kaelbling. A constraint-based method for solving sequential manipulation planning problems. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 3684–3691. IEEE, 2014.

[10] I. Mordatch, Z. Popović, and E. Todorov. Contact-invariant optimization for hand manipulation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 137–144. Eurographics Association, 2012.

[11] A. K. Pandey, J.-P. Saut, D. Sidobre, and R. Alami. Towards planning human-robot interactive manipulation tasks: Task dependent and human oriented autonomous selection of grasp and placement. In *Biomedical Robotics and Biomechatronics (BioRob), 2012 4th IEEE RAS & EMBS International Conference on*, pages 1371–1376. IEEE, 2012.

[12] M. Posa and R. Tedrake. Direct trajectory optimization of rigid body dynamical systems through contact. In *Algorithmic Foundations of Robotics X*, pages 527–542. Springer, 2013.

[13] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 489–494. IEEE, 2009.

[14] T. Siméon, L. Jean-Paul, J. Cortés, and A. Sahbani. Manipulation Planning with Probabilistic Roadmaps. *The International Journal of Robotics Research*, 23(7-8):729–746, Aug. 2004.

[15] 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 *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 639–646. IEEE, 2014.

[16] E. Todorov and W. Li. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference, 2005. Proceedings of the 2005*, pages 300–306. IEEE, 2005.

[17] M. Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI 2015)*, 2015.

[18] M. Toussaint. A tutorial on Newton methods for constrained trajectory optimization and relations to SLAM, Gaussian Process smoothing, optimal control, and probabilistic inference. In J.-P. Laumond, editor, *Geometric and Numerical Foundations of Movements*. Springer, 2016.

[19] M. Toussaint, T. Munzer, Y. Mollard, L. Y. Wu, N. A. Vien, and M. Lopes. Relational activity processes for modeling concurrent cooperation. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA 2016)*, 2016.