Logic Geometric Programming: An Optimization-based Approach to Combined Task and Motion Planning

Shivam Vats (svats) and **Sayan Mondal** (sayanmon)

December 2020

Abstract

Sequential manipulation tasks are notoriously hard to plan for because they involve a combination of continuous and discrete decision variables. In this project we study and experimentally analyze Logic Geometric Programming (LGP), which is a trajectory optimization based approach to solve such problems.

1 Introduction

Sequential manipulation tasks, like building a tower with blocks or even making coffee, require sequencing multiple manipulation actions in the correct order so as to accomplish a high level task. Computing each action may itself require solving a complex motion planning problem such as pick and place or grasping. The former problem has been studied exhaustively and efficient methods in the form of symbolic search algorithms exist. The latter is also a well studied problem that has a variety of solutions, primarily involving graph search or trajectory optimization.

What makes sequential manipulation so hard is the fact that we need to solve both these problems in conjunction. The resulting problem can be viewed as a hybrid optimization problem involving bother discrete and continuous decision variables. Here, the discrete variables define the order and the type of actions that need to be taken, while the continuous variables define the actual trajectory to be executed by the robot. Both these domains (discrete and continuous) may have their own constraints.

In the discrete domain, we have two kinds of constraints:

- **Preconditions** define the circumstances in which an action can be taken. For example, we may not want to pick up a block if there is another block resting on it.
- **Effects** define the changes introduced in the world as a result of an action. For example, if the robot picks up a block from a table, the effect is that

the block is no more on the table and will be attached to the gripper from thereon. We can see that the effect of every action is to change the constraints on the object that the robot acted on - from *resting on the table*, the constraints changed to *attached to the gripper*.

The continuous domain encodes the physical (or geometric) state of the robot and every object in its world. Each object (and the robot) obeys different dynamics depending on their constraints. The constraints on the robot mostly stay the same, but those on other objects may be changed by the robot. For example, if a mug is on a table, then applying some force on it parallel to the table induces some motion in the mug. On the other hand, if the robot were to rigidly grasp the mug, then applying a similar force on the mug may not induce any motion.

Knowing how discrete actions change the dynamics of objects is hence crucial to extract the actual trajectory. The Logic Geometric Programming [1] (LGP) framework does this by explicitly tracking the changes in system dynamics introduced by each action allowing it to first generate a symbolic action plan and then refine it with trajectory optimization using the tracked system dynamics.

2 Related Work

Most other TAMP approaches [2] [3] use a sampling-based planner in the loop for motion planning. This greatly limits the problems that these approaches can solve as sampling-based planning suffers in high-dimensional spaces. [2] require designing action specific samples which can be challenging. Further, these approaches cannot deal with dynamic interactions such as throwing a ball. LGP overcomes some these problems by using trajectory optimization for motion planning. Not only does this scale well to high-dimensional problems, this also allows dynamic interactions to be planner for as long as the system dynamics can be written down. While this makes the framework much more powerful and flexible, it can also potentially make it more cumbersome that using a simulator.

3 Background

3.1 Monte Carlo Tree Search (MCTS)

MCTS is a popular search algorithm for decision making that can often work with little domain knowledge (figure 1). It relies on repeated randomized simulation of actions to compute estimates of the expected payoff for every action available at the current state. These estimates are then used to pick the next action.

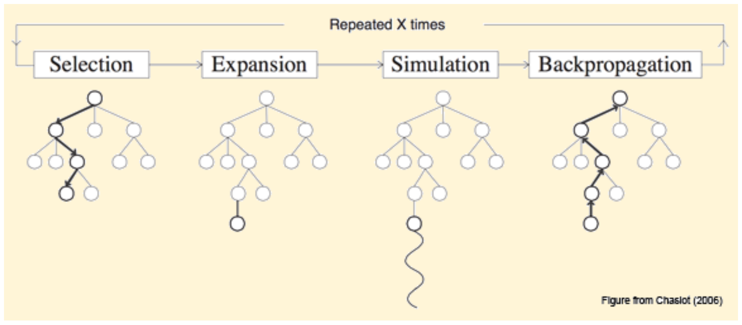

Figure 1: Overview of the MCTS algorithm.

3.2 KOMO - k-order Markov Optimization

KOMO [4] is a robot motion optimization framework that aims to draw on classical constrained optimization methods such as: Gauss-Newton (with adaptive stepsize and damping), Augmented Lagrangian and log-barrier. This optimization framework defines a very general class of robot motion problems, making it suitable for applying the optimization methods more efficiently.

Let $x_t \in R^n$ be a joint configuration and $x_{0:T} = (x_0, ..., x_T)$ be a trajectory of length T . Note that throughout this framework the trajectories are represented directly in configuration space. Thus, the optimization problems of a general “k-order non-linear sum-of-squares constrained” form can be mathematically structured as

$$\min_{x_{0:T}} \quad \sum_{t=0}^{T} f_t(x_{t-k:t})^\top f_t(x_{t-k:t}) + \sum_{t,t'} k(t,t') x_t^\top x_{t'}$$

$$\text{s.t.} \quad \forall_t : \quad g_t(x_{t-k:t}) \le 0, \quad h_t(x_{t-k:t}) = 0 \tag{1}$$

where $x_{t-k:t} = (x_{t-k}, .., x_{t-1}, x_t)$ are $k+1$ tuples of consecutive states. The functions $f_t(x_{t-k:t}) \in R^{d_t}, g_t(x_{t-k:t}) \in R^{m_t}$, and $h_t(x_{t-k:t}) \in R^{l_t}$ are arbitrary first-order differentiable non-linear k-order vector-valued functions. They define the cost terms, the inequality and the equality constraints respectively for each time step t. The optional kernel term $k(t,t')$ measures the correlation between time steps t and t'.

The cost vectors $f_t(x_{t-k:t})$ are very flexible in including various elements that can represent both transition and task-related costs. In order to elucidate this point with f_t, a few examples are shown below.

- To penalize transitional costs as square of velocities $k = 1$ is chosen. Thus, $f_t(x_{t-k:t}) = (x_t - x_{t-1})$.
- Similarly, in order to penalize transitional costs as square of accelerations, $k = 2$ is selected. This leads to $f_t(x_{t-k:t}) = (x_t - x_{t-1}) - (x_{t-1} - x_{t-2})$. Thus, $f_t(x_{t-k:t}) = (x_t + x_{t-2} - 2x_{t-1})$.

- It is even possible to penalize square of torques using $k = 2$. To display this let us consider the equations of motion $M\ddot{x} + F = \tau_t$ with $\ddot{x} \approx (x_{t+1} + x_{t-1} - 2x_t)$. Thus, with $f_t = (\sqrt{H}M(x_t - 2x_{t-1} + x_{t-2}) + F)$, where $\sqrt{H}$ is the Cholesky decomposition of a torque cost metric H and M is the mass matrix, we get costs $f_t^\top f_t = u_t^\top H u_t$.

The inequality and equality constraints g_t and h_t are equally general. Thus, the k-order optimization problem(1) can be rewritten as

$$\min_x f(x_{0:T})^\top f(x_{0:T}) \quad \text{s.t. } g(x) \le 0, \quad h(x) = 0 \tag{2}$$

where $f = (f_0; ..; f_T)$ is the concatenation of all f_t and $g = (g_0; ..; g_T), h = (h_0; ..; h_T)$. This defines a constrained sum-of-squares problem in which makes Gauss-Newton methods applicable. This general framework makes KOMO a very generic framework, where various optimizer can be applied.

4 Approach

The special feature about LGP-based combined task and motion planning approach compared to traditional TAMP approaches is that the objective is primarily given in terms of a cost function over the final geometric state, rather than a symbolic goal description. The goal information, $x(T)$, in the LGP framework is implicit in the evaluation of the final world configuration, $\psi(x(T))$.Thus, LGP-based TAMP allows to leverage optimization methods to inform search over potential action sequences, $a_{i:K}$.

The holistic problem formulation is a 1^{st} order non-linear constrained program over the full world trajectory, where the symbolic state-action sequence defines the (in-)equality constraints. The proposed way of solving such programs involves three levels of optimization.

4.1 Level 1: Optimization Over the Effective End Space

The first level of optimization (shown in Figure 2), which is the coarsest, introduces the concept of the effective end state kinematics, parametrically describing all possible end state configurations ($x(T)$) conditional to a given symbolic action sequence ($a_{1:K}$). This optimization over $x(T)$ is very fast, can inform symbolic search, and suitable as a search heuristic.

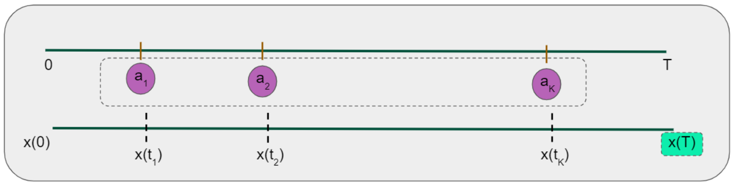

Figure 2: Schematic for level 1 optimization.

The effective end space $X^*(a_{1:K}) \subseteq X$ is defined as the set of all geometric configurations $x(T)$ that can be reached with a given symbolic action sequence $a_{1:K}$:

$$X^*(a_{1:K}) = \{x(T) \in X | a_{1:K}\}$$

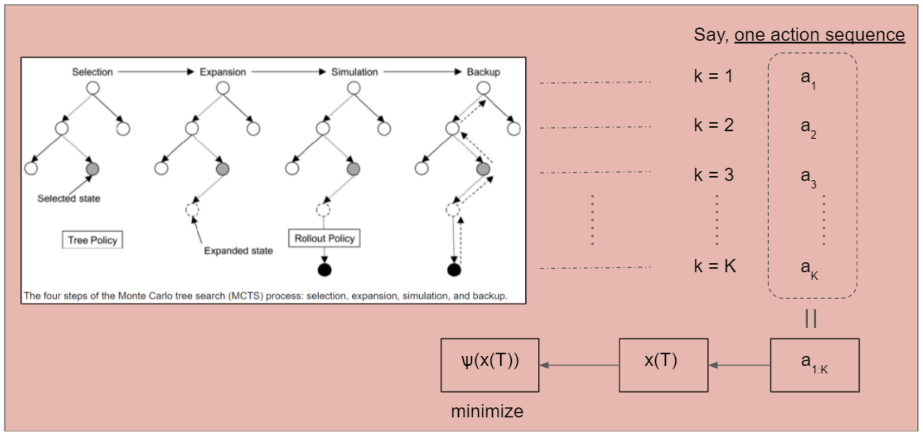

Figure 3: The pipeline of level 1 optimization.

The outline for level 1 optimization is shown in figure 3. In order to minimize the final cost $\psi(x(T))$, it undergoes a symbolic search over the action sequences $a_{1:K}$ using MCTS. We observe that, even without the explicit knowledge of the final configuration, $x(T)$, it aims to minimize it by minimizing $\psi(x(T))$.

Optimization in the end space is only an approximation to the optimization of the full path as it neglects constraints and the actual costs arising from actually performing the task in the constrained environment.

4.2 Levels 2: Optimizing At Kinematic Switches

Level 2 optimizes over interaction keyframes as shown in figure 4.

Here the optimization is jointly performed over all switch configurations $\{x(t) : t = t_1, t_2, .., t_K, T\}$ conditional to a given action sequence $a_{1:K}$. As a result it computes explicit geometric instantiations of the action operators. In the process it even optimizes the respective configurations to account also for long-term effects, e.g., optimizes over a grasp pose at $t = t_1$ to minimize also costs that arise several time steps later due to the choice of grasp pose.

4.3 Level 3: Optimizing Across Kinematic Switches over the Entire Path

Level 3 optimizes over the full world trajectory $x : [0, T] \rightarrow X$ conditional to a given action sequence $a_{1:K}$ as displayed in figure 5. This optimization occurs across interactions and thus, requires methods for trajectory optimization that

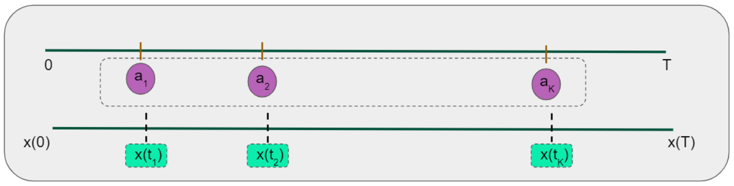

Figure 4: Schematic for level 2 optimization.

are able to cope with kinematic switches. KOMO, discussed earlier, is suitable for such a task.

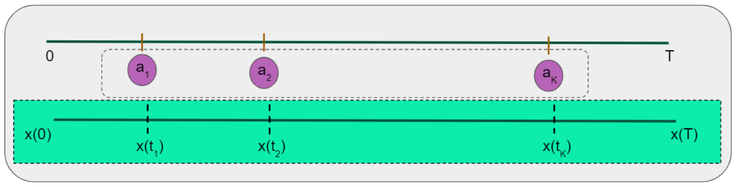

Figure 5: Schematic for level 3 optimization.

5 Results

We evaluated LGP on two types of tasks

1. **Pick and place:** Pick blocks from one table and place them in the goal region located on another table.
2. **Block stacking:** Pick blocks from one table and stack them in the goal region located on another table.

Table 1: Performance of LGP with different number of blocks.

Task	1 Block	2 Blocks	3 Blocks	4 Blocks
Pick and Place	✓	✓	✓	✗
Block Stacking	✓	✓	✓	✗

LGP was able to successfully solve these tasks for upto 3 blocks. Even though there was space in the goal region for 4 blocks, LGP found it hard to compute a plan because of the presence of other blocks in the way and restricted goal region. This suggests that even though LGP is good at successfully solving long horizon planning problems, it cannot handle problems that require solving a difficult motion planning problem, for example due to clutter.

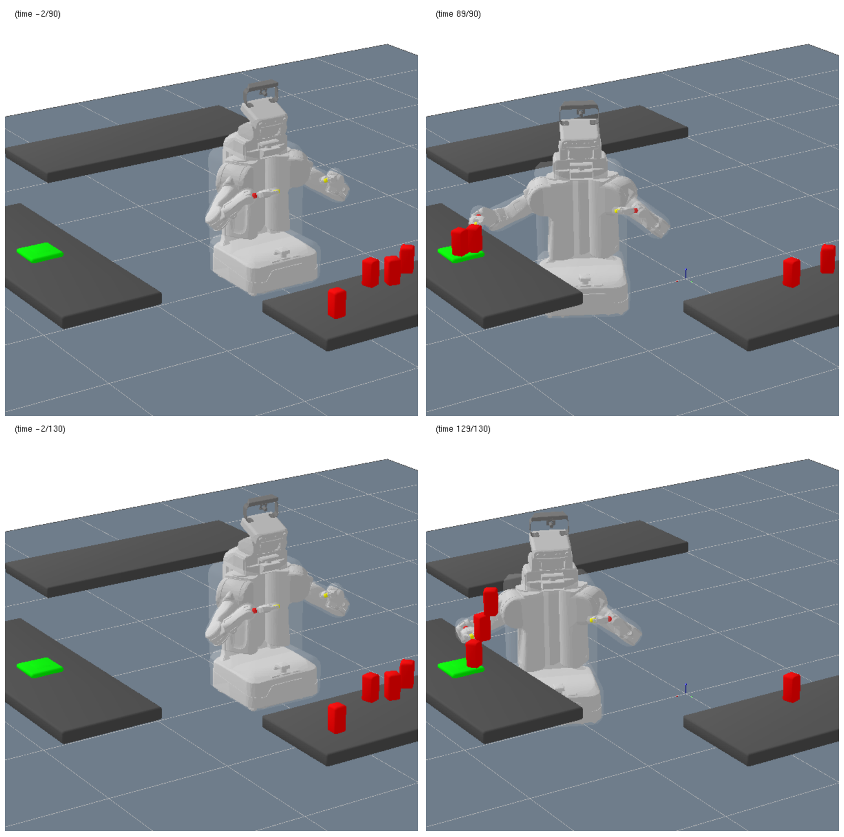

Figure 6: Result of planning for pick-place (top) and block stacking (bottom) tasks: (left) start states; (right) final states.

6 Conclusion

In this project we studied and experimentally evaluated an optimization-based task and motion planning (TAMP) approach, called Logic Geometric Programming (LGP). In particular, we looked at how it interleaved symbolic search and continuous trajectory optimization. TAMP is an exceedingly hard problem which involves solving multiple motion planning and graph search problems, that are themselves active research areas. We found that LGP is able to solve relatively long-horizon TAMP problems as long as they don't involve computing difficult motion plans. Finally, LGP in its current form assumes highly simplified dynamics for efficiency which limits its practical usage.

References

[1] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *IJCAI International Joint Conference on Artificial Intelligence*, volume 2015-Janua, pages 1930–1936, 2015.

[2] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Ffrob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research*, 37(1):104–136, 2018.

[3] Neil T Dantam, Zachary K Kingston, Swarat Chaudhuri, and Lydia E Kavraki. Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and systems*, volume 12, page 00052. Ann Arbor, MI, USA, 2016.

[4] Marc Toussaint. Newton methods for k-order markov constrained motion problems. *arXiv preprint arXiv:1407.0414*, 2014.