2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Graph-of-Constraints Model Predictive Control for Reactive Multi-agent Task and Motion Planning

Anastasios Manganaris, Jeremy Lu, Ahmed H. Qureshi, and Suresh Jagannathan

***Abstract*— Sequences of interdependent geometric constraints are central to many multi-agent Task and Motion Planning (TAMP) problems. However, existing methods for handling such constraint sequences struggle with partially ordered tasks and dynamic agent assignments. They typically assume static assignments and cannot adapt when disturbances alter task allocations. To overcome these limitations, we introduce Graph-of-Constraints Model Predictive Control (GoC-MPC), a generalized sequence-of-constraints framework integrated with MPC. GoC-MPC naturally supports partially ordered tasks, dynamic agent coordination, and disturbance recovery. By defining constraints over tracked 3D keypoints, our method robustly solves diverse multi-agent manipulation tasks—coordinating agents and adapting online from visual observations alone, without relying on training data or environment models. Experiments demonstrate that GoC-MPC achieves higher success rates, significantly faster TAMP computation, and shorter overall paths compared to recent baselines, establishing it as an efficient and robust solution for multi-agent manipulation under real-world disturbances. Our supplementary video and code can be found at https://sites.google.com/view/goc-mpc/home.**

I. INTRODUCTION

The effective deployment of multi-robot teams promises to automate significantly more complex and useful real-world tasks. For example, two robotic arms benefit from a substantially larger effective workspace, the ability to manipulate objects designed for two-handed human use, and the capacity to perform work in parallel. Scaling to larger teams of arms or mobile manipulators further increases throughput and task diversity. However, these capabilities come at the cost of much higher complexity in performing Task and Motion Planning (TAMP).

Despite this complexity, a variety of successful methods for TAMP have been developed and remain applicable in the multi-robot setting. A particularly well-studied family of approaches leverages modern constrained optimization to directly solve TAMP problems formulated as sequences of constraints [1]–[3]. Just as trajectory optimization underlies online control when used in a receding-horizon loop (e.g., with Model Predictive Control (MPC)), optimization-based formulations of TAMP have also been extended to reactive execution, enabling high-level symbolic plans and low-level continuous motions to be recomputed during runtime [4].

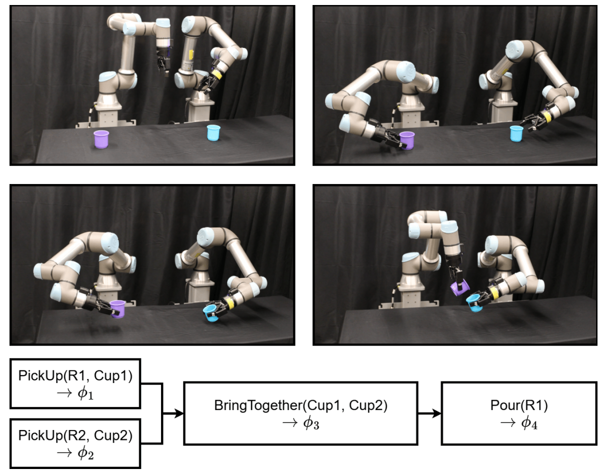


Fig. 1: Graphs-of-Constraints (GoCs) can naturally express partially-ordered multi-agent tasks using a Directed Acyclic Graph (DAG) of arbitrary system constraints. Our method, GoC-MPC, can use a GoC with constraints defined in terms of workspace keypoints to find optimal solutions for general multi-agent manipulation tasks using only visual observations. For this two-cup manipulation task, GoC-MPC finds an initial solution in 0.373 seconds, and all subsequent solutions in 0.065 seconds on average, demonstrating extremely fast performance.

This line of work demonstrates how re-solving optimization problems in response to new information provides a natural mechanism for handling failures and disturbances. Moreover, recent research has shown that defining constraints in terms of universally applicable geometric primitives—such as spatial keypoints tracked in the robot's workspace—enables general-purpose manipulation across a wide range of tasks. This allows such methods to achieve impressive generality and avoid the need for precise system models or extensive task-specific training data [5].

Although these methods have achieved success in single-agent domains, we identify two fundamental limitations that currently limit their usefulness for general multi-agent tasks. In particular, using singular sequences of constraints imposes a **total-order** on the steps carried out by the agents and it imposes a **static** assignment of agents to steps. By artificially imposing ordering constraints between steps that could be otherwise completed in parallel, such as the first two steps in Fig. 1, the system can unnecessarily halt the progress of one agent when waiting for the other to complete its task. Disturbances can exacerbate this issue arbitrarily, by

This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-24-1-0233. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force.

Anastasios Manganaris, Jeremy Lu, Ahmed H. Qureshi, and Suresh Jagannathan are with the Department of Computer Science, Purdue University, West Lafayette, IN, USA, 47907. Email: {amangana, lu1008, ahqureshi}@purdue.edu and suresh@cs.purdue.edu

preventing a single agent from completing an early assigned task, and therefore delaying the entire team, even if they are not affected by the disturbance at all. Static agent assignments can also contribute to this problem, as a poor choice in static assignments without considering the current state of the system can lead to unnecessary extra work or even complete infeasibility. Disturbances should also be able to prompt reassignments of agents if necessary.

In this work, we address these limitations by realizing the generalization of Sequences-of-Constraints to *Graphs*-of-Constraints (GoCs). GoCs are DAGs of constraints derived from symbolic, high-level actions, and therefore can naturally represent **partially-ordered** sets of task steps. Furthermore, constraints within a GoC are defined with respect to a **dynamic** assignment of agents to subtasks, which allows for optimizing over this assignment when obtaining the solution. To solve problems formulated with GoCs, we also contribute GoC-MPC, which decomposes and solves the optimization problem resulting from a GoC at real-time speeds, therefore enabling reactive multi-agent TAMP. We evaluate our algorithm on several bimanual manipulation tasks that take advantage of properties of the GoC formulation, including block stacking (featuring parallelized pick-ups and sequenced placements), transferring liquids in cups (featuring coordination between agents based on the objects they are manipulating), and tablecloth folding (featuring exact synchronization of agents).

II. RELATED WORK

In general, the field of TAMP is concerned with extending symbolic, discrete planning algorithms [6]–[8] with continuous motion planning [9], [10] to solve complex, long-horizon robotics tasks [11], [12]. Approaches to TAMP include constraint-based [13], sampling-based [11], and the aforementioned optimization-based methods [14]. The latter TAMP methods formulate a continuous or mixed-integer optimization problem [1]–[3], [14] and therefore benefit from naturally being able to incorporate diverse costs and constraints. Optimization-based approaches have also been extended to multi-agent settings [15].

Reactive TAMP is a closely related area to TAMP that, instead of assuming that TAMP solutions can be perfectly executed without failure or external disturbances, focuses on the ability to quickly recompute plans online during execution. Specific strategies include synthesizing hierarchical policies for symbolic actions via reactive synthesis from temporal logic specifications [16]–[18], defining object-relative plans that remain valid despite object motion [19], and re-solving optimization problems derived from symbolic skeleton plans [4], [20]–[22]. To our knowledge, we contribute the first multi-agent-capable reactive TAMP method belonging to this latter category.

We also note that this last class of solutions that reactively optimize with respect to a task skeleton has received increased attention as a way to relax the heavy modeling assumptions of traditional TAMP. In particular, Relational Keypoint constraints (ReKeps) [5] are constraints on the geometric relationships of keypoints within an environment, which enables the execution of general manipulation skills without requiring exhaustive world modeling or large, task-specific datasets as with other methods [23]–[25]. Our method is also designed to use this constraint representation, and therefore achieves similar generality, while still addressing the gaps in prior reactive TAMP methods for supporting dynamic agent assignment and parallelizable task structures.

III. METHODS

In this section, we present our approach for achieving reactive multi-agent TAMP. We begin by discussing the preliminaries for our method, including necessary notations and the limitations of prior methods in supporting multi-agent tasks (Section III-A). We then describe GoCs and how they specifically address these limitations, as well as the optimal control problem for TAMP that results from a GoC (Section III-B). Next, we describe the specific steps that are necessary to decompose and efficiently solve this problem within a receding-horizon MPC scheme to enable reactivity in multi-agent TAMP (Section III-C). The final algorithm, GoC-MPC, is summarized in Section III-D.

A. Preliminaries

As mentioned above, our method is based on the idea that robotic tasks can be defined in terms of ordered constraints on the robotic system state, inspired by methods such as [4], [5]. Let $X \subset \mathbb{R}^d$ denote the d-dimensional configuration space for a robot system. The configuration space $X = X_a \times X_p$ consists of an actuated component $X_a \subset \mathbb{R}^{d_a}$ for the controllable robot configurations and a non-actuated passive component $X_p \subset \mathbb{R}^{d_p}$ for the remaining scene elements (e.g., keypoints on objects). In a multi-agent system, the actuated configuration space $X_a = X_{a_1} \times X_{a_2} \times \ldots \times X_{a_M}$ can be split into M single-agent configuration spaces X_{a_j} for $j \in [1, M]$. For any configuration space X, we denote its tangent space as $\dot{X}$

Constraints are functions denoted by $\phi_i : X \to \mathbb{R}^{d_i}$ with $d_i \in \mathbb{N}$ where $\phi_i(x) \leq 0$ indicates satisfaction of the constraint under configuration x. A sequence of constraints, as defined in [4], [5], is then a pair $(\phi_{1:K}, \bar{\phi}_{1:K})$ of K "waypoint" constraint functions and K "path" constraint functions. Each waypoint is associated with a specific time t_i which satisfies $t_{i-1} \leq t_i$. Under this formulation, the objective is then to find an optimal path $\xi : [0, t_K] \to X$ that satisfies every constraint at the appropriate times.

Limitation of existing constraint representations. As mentioned above, the property that $\forall i \in [1, K]\ t_{i-1} \leq t_i$ requires that the steps in the trajectory ξ are totally-ordered. Therefore, it is impossible to specify stages that should be done in parallel or transposed as necessary for optimality. Furthermore, this formulation implies that multiple agents are always implicitly treated as a single "joint-agent", and every constraint ϕ must either operate over the joint-agent configuration or a static subset of that joint-agent configuration. We consider this static assignment issue the second major issue for the existing formulation.

B. Graphs of Constraints

We now define GoCs, which are a generalization of the sequences-of-constraints used in prior work and immediately address the limitations of a totally-ordered sequence and statically assigned constraints. For the former, we instead define a partial order for these constraints, which naturally results in defining a DAG structure. For the latter, we extend the definition of constraints to consider a dynamic set of assignments between M agents and K assignable "subtasks". We can represent these assignments with a matrix $A \in \{0,1\}^{K\times M}$ of binary decision variables with K rows and M columns. This matrix is constrained to be row stochastic to ensure that only a single agent is assigned to each of the K subtasks. A constraint function is now defined to be a function $\phi_i : \{0,1\}^{K\times M} \times X \to \mathbb{R}^{d_i}$ where $\phi(A, x) \leq 0$ indicates satisfaction of the constraint under assignment A and configuration x. The additional input allows now for the definition of *disjunctive* constraints that apply specifically to the agents selected for the relevant subtasks in A. In our implementation, these disjunctive constraints were implemented using big-M gating [26].

Let Φ denote the set of all such constraint functions. Then, a GoC is the tuple

$$\mathcal{G} = (V, E, \Phi_V, \Phi_E),$$

where $V \subset \mathbb{N}$ is a finite set of N nodes (i.e., step indices), $E \subseteq V \times V$ is a set of directed edges, $\Phi_V : V \to \Phi$ is a mapping from the nodes to "waypoint" constraint functions, and $\Phi_E : E \to \Phi$ is a mapping from the edges to "edge" constraint functions. For brevity, we denote $\Phi_V(v)$ as ϕ_v and $\Phi_E(e)$, where $e = (a, b)$, as $\bar{\phi}_e$ or $\bar{\phi}_{ab}$.

Given a GoC, the objective is then to find a set of agent assignments A, an optimal and constraint-satisfying system trajectory $\xi : [0, t_{\max}] \to X$, and a set of timings $t_{1:N}$ for each node in the GoC. The trajectory is defined over a time interval ending at the maximum time needed to finish the task $t_{\max} = \max_{1\leq i\leq N} t_i$ and is required to satisfy every waypoint and edge constraint in that interval at the appropriate times. Furthermore, we typically are interested in solving the task in the minimum amount of time and with a minimal amount of other costs. These constraints and this objective then can be expressed together as

$$\begin{aligned}
\min_{A,\xi,t_{1:N}} \quad & t_{\max} + \int_0^{t_{\max}} c(\xi(t), \dot{\xi}(t), \ddot{\xi}(t))\, dt && (1)\\
\text{s.t.} \quad & \xi(0) = x_0 \quad \dot{\xi}(0) = \dot{x}_0 \quad t_0 = 0 \\
& \forall k \in \{0, 1, \dots, K\} : \sum_{j=0}^{M} A(k, j) = 1 \\
& \forall v \in V : \phi_v(A, \xi(t_v)) \leq 0 \\
& \forall (a, b) \in E : t_a \leq t_b \wedge \\
& \qquad \forall t \in [t_a, t_b] : \bar{\phi}_{ab}(A, \xi(t)) \leq 0 \\
& \forall t : \dot{\xi}(t) = h(\xi(t)), && (2)
\end{aligned}$$

where c represents an arbitrary cost function that can penalize acceleration or collisions. The solution trajectory is constrained to originate at the current system state x_0 with the current velocity $\dot{x}_0$, and the task assignments A are constrained such that only one agent is assigned to a single task. The final constraint (2) represents the dynamical constraints of the system. For instance, the movement of a cube is constrained to match the movement of the robot assigned to hold that cube and is otherwise affected by gravity.

C. Problem-Decomposition

As with other methods leveraging optimization for long-horizon tasks, the full optimization problem is difficult to solve and impractical for a fast reactive planner. Instead, the problem can be decomposed and a satisfactory approximation can be obtained by solving for the three resulting components: (1) the waypoints and agent assignments, (2) the trajectory timings and velocities, and (3) a short horizon path for all agents. We describe this decomposition here and how each subproblem is extracted from a GoC.

a) Waypoints and Assignments Sub-Problem: The first component of the decomposed problem solves for an optimal set of waypoints, represented by a vector $W \in X^N$ of N configurations, along with the assignments of agents to tasks, A. Let $W(v)$ for $v \in V$ denote the waypoint at node v, and let V_{next} denote the topologically-first set of nodes. To encourage minimizing the total time as in (1), we use a surrogate objective based on the total geodesic distance $\bar{c} : X \times X \to \mathbb{R}$ between the waypoints on every edge of a GoC and between the initial system state x_0 to the nodes in V_{next}:

$$\begin{aligned}
\min_{W,A} \quad & \sum_{v\in V_{\text{next}}} \bar{c}(x_0, W(v)) + \sum_{a,b\in E} \bar{c}(W(a), W(b)) \\
\text{s.t.} \quad & \forall k \in \{1, \dots, K\} : \sum_{j=1}^{M} A(k, j) = 1 \\
& \forall v \in V : \phi_v(A, W(v)) \leq 0 \\
& \forall (a, b) \in E : \bar{\phi}_{ab}(A, W(a)) \leq 0 \wedge \bar{\phi}_{ab}(A, W(b)) \leq 0 \\
& \forall (a, b) \in E : \bar{h}(A, W(a), W(b)) \leq 0.
\end{aligned}$$

The remaining constraints adapt the original constraints for enforcing the assignment matrix is row-stochastic, every waypoint constraint is respected, and every edge constraint is respected. The joint optimization of waypoints and agents allows for resolving the subset of constraints that are dependent on agent assignments.

The final constraint, defined by $\bar{h}$, is meant to approximate the consequences of the dynamical constraints (2) in the transitions between waypoints. In our implementation, we reuse the information provided by the edge constraints Φ_E to determine across which edges are keypoints manipulated. Using this information, we define $\bar{h}$ by making two assumptions: (i) points not being manipulated by a robot remain fixed, and (ii) points being manipulated move rigidly with the robot and with each other. Note that these constraints are also dynamically gated depending on the agent assignments in A.

The resulting problem is a mixed-integer non-linear program, but is small enough to be efficiently, locally solved

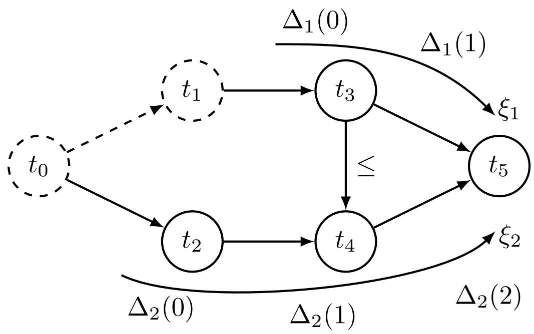


Fig. 2: After waypoints W and agent assignments A have been determined, splines for each agent are threaded through the appropriate waypoints in order. The current subgraph is determined by a subset of remaining nodes R (denoted with solid borders) that shrinks as tasks are completed (denoted with dashed borders). The graph's structure determines additional constraints that can enforce ordering or ensure synchronization between agents' actions.

with enumerating the possible integer variable solutions and solving for each assignment the non-linear program using Ipopt [27]. On the first instantiation of this problem, we initialize the waypoint decision variables $W(v)$ for all $v \in V$ with the initial state x_0, and on all subsequent iterations they are initialized with the previous iteration's solution.

b) Agent Splines Problem: After solving for the waypoints and agent assignments, we solve for an optimal multi-agent path passing through them. We represent this multi-agent path with a set of agent-specific cubic splines $\{\xi_1, \dots \xi_M\}$ collectively starting from x_0 with velocity $\dot{x}_0$ and passing through the appropriately ordered agent-specific waypoints $W_j \in X_{a_j}^{N_j}$ for $j \in [1, M]$. The number of waypoints relevant to agent j is denoted by N_j. Each spline also passes through its i^{th} waypoint with a velocity denoted by $V_j(i) \in \dot{X}_{a_j}$. To avoid the possibility of invalid timings, the timing of the spline is determined by a vector of time-deltas $\Delta_j \in \mathbb{R}^{N_j+1}$, where $\Delta_j(i)$ gives the time for agent j to travel between the i^{th} and the $i+1^{\text{th}}$ node on their spline. Based on these values, it is convenient to compute dynamical costs for the entire spline, denoted by $c(W_j, V_j, \Delta_j)$, including its total acceleration, max velocity, max acceleration, and max jerk. An illustration of the overall concept of splines threaded through a GoC is provided in Fig. 2.

We identify the ordered waypoints for each agent with a topologically-ordered breadth-first traversal of the GoC. For each traversed node v, agent j's component of the waypoint $W(v)$ is added to W_j for every agent j that is relevant to the constraints ϕ_v. Relevance is determined by either the static definition of ϕ_v or dynamic assignments in A. During this traversal, additional inter-agent timing constraints can be deduced. A set of "less-than" constraints $E_{\leq}$ is obtained from the traversal's non-tree edges (a, b), enforcing the relevant order of two agent's waypoints. Each edge is recorded as a tuple (j_a, l_a, j_b, l_b) where j_a, j_b are two agents that visit node a and b, and l_a, l_b are the indices of a and b in their respective trajectories. For nodes in which two trajectories converge, a similar set $E_=$ of tuples (j_a, l_a, j_b, l_b) can be obtained to enforce the dynamic synchronization of a pair of agents. For example, the edge from node 3 to node 4 in Fig. 2 forces agent 1 to reach its configuration in waypoint 3 before agent 2 reaches its configuration in waypoint 4. The convergence of both splines at node 5 require that agent 1 and agent 2 both arrive reach their configurations in waypoint 5 simultaneously. These constraints are represented by tuples $(1, 0, 2, 1) \in E_{\leq}$ and $(1, 1, 2, 2) \in E_=$ respectively.

In general, the agent spline subproblem is

$$\begin{aligned}
\min_{V_{1:m}, \Delta_{1:m}} \quad & \sum_{j=1}^{M} \sum_{i=0}^{N_j} \Delta_j(i) + c(W_j, V_j, \Delta_j) \\
\text{s.t.} \quad & \forall j \in \{1, 2 \dots M\} : V_j(0) = \dot{x}_{0_j} \wedge V_j(N_j) = 0 \\
& \forall (j_a, l_a, j_b, l_b) \in E_{\leq} : \sum_{i=0}^{l_a} \Delta_{j_a}(i) \leq \sum_{i=0}^{l_b} \Delta_{j_b}(i) \\
& \forall (j_a, l_a, j_b, l_b) \in E_= : \sum_{i=0}^{l_a} \Delta_{j_a}(i) = \sum_{i=0}^{l_b} \Delta_{j_b}(i),
\end{aligned}$$

which is a quadratic program that can be efficiently solved with the optimization routines implemented in MOSEK [28].

c) Short Receding-Horizon Problem: Finally, using the smooth reference trajectories computed in the last subproblem, aggregated into a single joint-agent trajectory $\xi^*(t) = \langle \xi_1(t), \xi_2(t), \dots \xi_M(t) \rangle$, we can compute a short receding horizon trajectory of H steps spaced out by a time interval dt that tracks the reference trajectory as close as possible while also taking into account more fine dynamical costs, represented by c_h, such as environment collision costs and reachability costs. This results in the objective

$$\begin{aligned}
\min_{\xi} \quad & \int_0^H ||\xi(t) - \xi^*(t)||^2 + c_h(\xi(t)) \\
\text{s.t.} \quad & \xi(0) = x,
\end{aligned}$$

which is again a quadratic program that is efficiently solved with MOSEK [28].

D. Graph-of-Constraints MPC

In this section, we describe the complete algorithm, shown in Algorithm 1, that iteratively solves the above three subproblems throughout execution in order to perform reactive TAMP. Apart from these optimization problems, the algorithm generalizes the phase progression and backtracking components of prior work [4], [5] so that progression occurs topologically forward or backward through the GoC and is monitored through a set of nodes $R \subseteq V$ that represent the remaining steps in the task.

To implement forward phase progression, we examine the computed time at which each agent is expected to cross the next node, which is given in $\Delta_j(0)$, and compare it to a small time-delta cutoff value τ. If the crossing is expected to occur within the cutoff period, all constraints associated with that node are checked and, if they are satisfied, the node is

Algorithm 1 GoC-MPC Cycle

Require: A GoC $\mathcal{G}$, the remaining nodes $R \subseteq V$, time-delta cutoff τ, constraint tolerance ϵ, current state and velocity x, $\dot{x}$

for $(a, b) \leftarrow$ CUTEDGES$(\mathcal{G}, R)$ **do** ▷ $E \cap ((V \setminus R) \times R)$
  if $\bar{\phi}_{ab} \geq \epsilon$ **then**
    $R \leftarrow R \cup a$ ▷ Phase Backtracking
    return
  end if
end for
$\mathcal{G}_s \leftarrow$ SUBGRAPH$(\mathcal{G}, R)$
$P_1 \leftarrow$ CONSTRUCTWAYPOINTPROBLEM$(\mathcal{G}_s, x)$
$W, A \leftarrow$ SOLVE(P_1) ▷ Sec. III a.
$\{W_j\}_{j=1}^{M} \leftarrow$ GETAGENTPATHS$(\mathcal{G}_s, W, A)$
$P_2 \leftarrow$ CONSTRUCTTIMINGPROBLEM$(\{W_j\}_{j=1}^{M}, x, \dot{x})$
$\{\Delta_j, V_j\}_{j=1}^{M} \leftarrow$ SOLVE(P_2) ▷ Sec. III b.
for $j \leftarrow 1$ to M **do**
  $v_j \leftarrow$ GETNEXTNODE(j)
  if $\Delta_j(0) \leq \tau$ **then**
    if $\phi_{v_j} \leq \epsilon$ **then**
      $R \leftarrow R \setminus v$ ▷ Phase Progression
    end if
  end if
end for
for $j \leftarrow 1$ to M **do**
  $\xi_j \leftarrow$ CUBICSPLINE(W_j, V_j, Δ_j)
end for
$P_3 \leftarrow$ CONSTRUCTSHORTPATHPROBLEM$(\{\xi_j\}_{j=1}^{M}, x)$
$\xi_h \leftarrow$ SOLVE(P_3) ▷ Sec. III c.
return ξ_h

removed from the set of remaining nodes. For backtracking, one instead examines the edge constraints for each edge crossing between one of the remaining nodes in R and the rest of the completed graph. Here, if any edge constraint is violated, the source node associated with that edge is added back into R and the cycle restarts. This makes it so that repeated cycles will backtrack until edge constraints are satisfied. Finally, the set of remaining nodes is used in each cycle to obtain a subgraph $\mathcal{G}_s$ from the overall GoC $\mathcal{G}$ that determines the structure of the waypoint optimization problem and of the splines relevant to each agent. As a result, the entire controller allows for flexible agent parallelization, progression through tasks, and recovery from failures.

IV. EXPERIMENTS

In this section, we describe the experiments conducted to validate GoC-MPC as an efficient and robust method for reactive TAMP. We evaluate the approach on three representative tasks:

- **Block-Stacking**: The robot begins with three colored blocks positioned flat on the workspace. The task requires grasping and stacking each block to construct a stable three-block tower.
- **Pick-and-Pour**: The system is initialized with a pouring cup and empty target container, both in upright positions. The robot must transfer the liquid between containers while maintaining precise control to prevent spillage.
- **Cloth-Folding**: The manipulation task involves a rectangular cloth initially laid flat on the work surface. The robot executes a series of grasping and folding motions to bring the cloth's corner regions inward, achieving a compact, organized configuration.

TABLE I: Comparison of our approach with ReKep on the **Block-Stacking** task shows that our method outperforms across all metrics. It achieves a 100% success rate and, on average, runs 70 × faster than ReKep, highlighting its effectiveness for reactive TAMP.

	GoC-MPC (Ours)	**ReKep**
Success Rate	$\mathbf{10/10}$	$7/10$
Max Time (s)	$\mathbf{0.520 \pm 0.311}$	8.38 ± 0.062
Avg. Time (s)	$\mathbf{0.108 \pm 0.012}$	7.11 ± 0.186
Total Length (m)	$\mathbf{2.54 \pm 0.216}$	4.75 ± 2.59

TABLE II: Comparison of our approach with ReKep on the **Pick-and-Pour** task. Our method achieves a 100% success rate, runs 40 × faster than ReKep, and produces significantly shorter path lengths than the baseline.

	GoC-MPC (Ours)	**ReKep**
Success Rate	$\mathbf{10/10}$	$6/10$
Max Time (s)	$\mathbf{0.512 \pm 0.328}$	12.01 ± 0.966
Avg. Time (s)	$\mathbf{0.216 \pm 0.064}$	8.49 ± 0.413
Total Length (m)	$\mathbf{1.94 \pm 0.742}$	3.20 ± 2.08

The visualization of these tasks is shown in Fig. 3.

Baselines. While several methods address TAMP tasks, they typically require complete world models and cannot operate directly from visual observations. To the best of our knowledge, ReKep [5] is the only method that solves reactive TAMP tasks directly from keypoints extracted from visual input, without relying on explicit world models. We therefore select ReKep as our primary baseline.

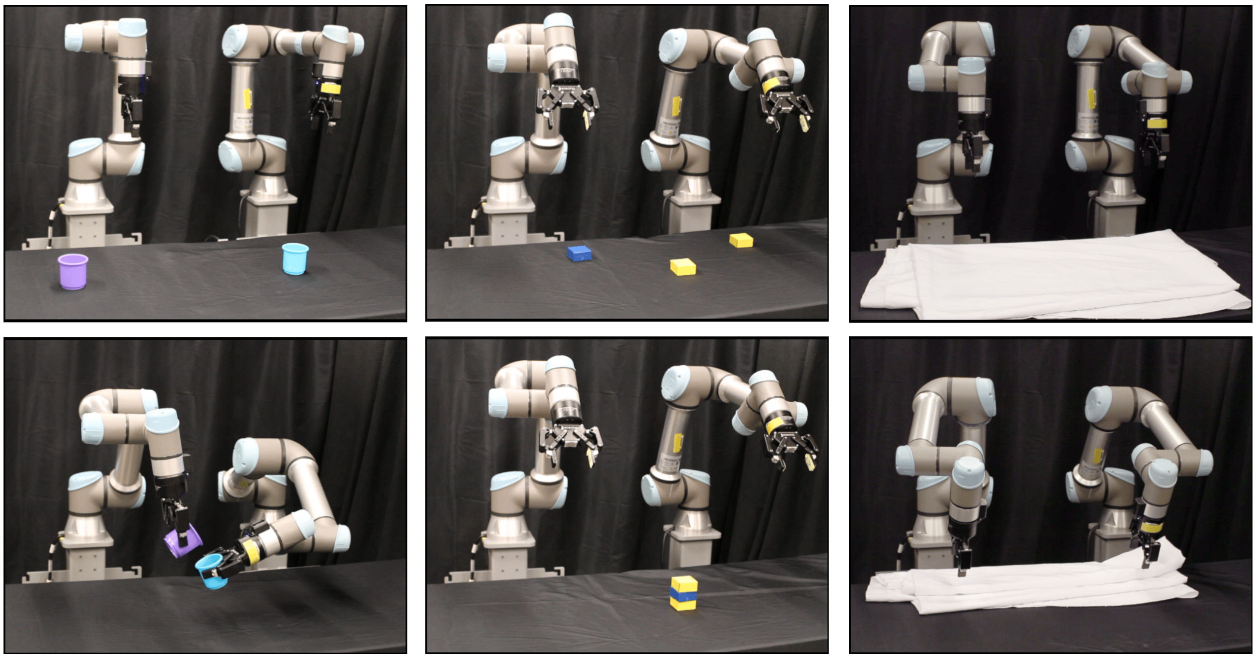
Experimental analyses. We conducted four sets of evaluations:

- **Static setting (without external disturbance):** Comparison against baselines in block-stacking and pick-and-pour tasks using visual observations (Section IV-A).
- **Disturbance setting:** Comparison against baselines on block-stacking tasks under external disturbances, where agents are explicitly disrupted (e.g., objects pushed from robot hands or object locations altered), requiring reactive TAMP to reobserve changes and dynamically adapt task assignments and motion plans (Section IV-B).
- **Scalability:** Analysis of performance against the number of agents and objects in block-stacking tasks (Section IV-C).
- **Real-world validation:** Demonstration of our approach on all three tasks in physical settings (Section IV-D).

Metrics. To quantify performance, we evaluate results using:

- **Success:** Proportion of trials completed successfully.
- **Max Time (s):** Maximum wall-clock execution time across all MPC cycles.
- **Average Time (s):** Average wall-clock execution time per MPC cycle.

Our static-setting experiments were simulated in Isaac-Sim [29] with OmniGibson [30], and our disturbance-setting



(a) Pick-and-Pour (b) Block Stacking (c) Tablecloth Folding

Fig. 3: We conducted experiments in two simulated and three real bimanual maniulation tasks with two UR5e arm robots. These included a distant-cup pick-and-pour task, a three-block stacking stacking task, and a tablecloth folding task (not simulated due to including a soft-body). These tasks demonstrate that our method is capable of handling partially-ordered tasks, coordination of multiple robots based on the objects they are manipulating, and synchronization between agents.

and scalability experiments were simulated using Drake [31]. All experiments were run on a local workstation with a 12th Gen Intel i7-12700F CPU and 32 GB RAM.

A. Comparisons to Existing TAMP Methods

In this study, we initially examine our method with ReKep in *static scenarios* without external disturbances, focusing on block-stacking and pick-and-pour tasks. Results are summarized in Tables I and II.

Our method achieves significantly lower computation times and total path lengths, along with higher success rates. While ReKep's longer computation times stem largely from its optimizer and cost formulations, our method's superior success rate and solution quality can be attributed to the advantages of the GoC structure.

In the block-stacking task, linearizing the GoC into a sequence suitable for ReKep introduces substantial idle time for the second robot, since placements must be explicitly interleaved. For the pick-and-pour task, although the DAG structure can be linearized while retaining some parallelism in cup handling, ReKep still suffers from single-sequence failure modes. In one case, a failed grasp triggers backtracking of the entire stage, unnecessarily resetting the other robot's successful grasp. In another, ReKep requires independent path planning for each agent, preventing joint optimization toward a central pouring location. This limitation makes some pouring locations infeasible and leads to misaligned pours. By contrast, our formulation inherently considers agent interactions and potential collisions during optimization, avoiding such failure modes and enabling more robust multi-agent execution.

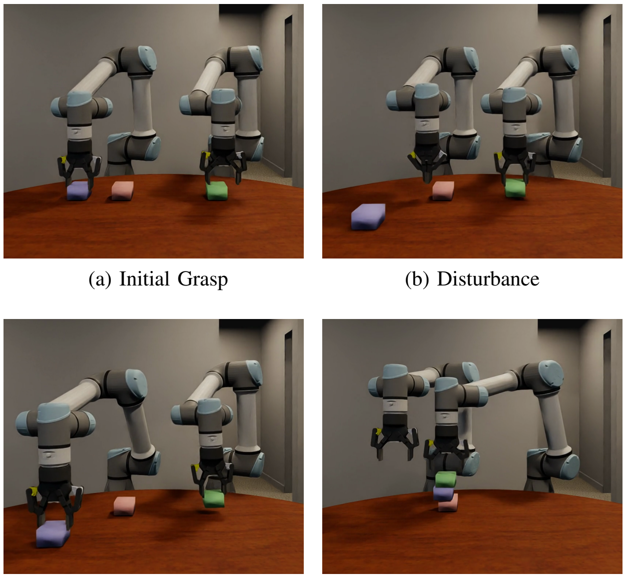
TABLE III: Comparison of our approach with ReKep on the block-stacking with external disturbances. While both methods achieved a 100% success rate, the *agent-independent backtracking* provided by GoCs saved on average 0.64 meters of unnecessary movement compared to ReKep, executing $80\,\times$ faster in maximum time.

	GoC-MPC (Ours)	**ReKep**
Success Rate	$\mathbf{10/10}$	$10/10$
Max Time (s)	$\mathbf{0.161 \pm 0.093}$	12.36 ± 6.180
Avg. Time (s)	$\mathbf{0.052 \pm 0.007}$	3.29 ± 0.331
Total Length (m)	$\mathbf{6.89 \pm 0.938}$	7.53 ± 0.892

B. Robustness to Disturbances

In multi-agent settings, disturbances are particularly challenging: external perturbations may force agent reassignment, delay current and future actions, or alter the set of feasible trajectories. To evaluate the robustness of GoC-MPC in such cases, we consider how it adapts to disturbances.

A key property of GoCs is the ability to backtrack when edge constraints are violated. Perturbations may trigger such violations and update the set of active stages—for example, one agent may need to reposition while another continues unaffected. Prior methods do not support coordinated backtracking across agents, whereas our approach does, resulting



(a) Initial Grasp (b) Disturbance

(c) Backtracking (d) Task Completion

Fig. 4: GoCs allow for agents to independently backtrack to previous stages when a disturbance is applied. In the block stacking task, a disturbance to the block in the left robot's grasp prompts the robot to loop back to picking up the block, and the right robot's path is slowed to the point where it smoothly reaches its original destination after the other robot has finished retrieving the disturbed block.

TABLE IV: Scalability performance of GoC-MPC. The number of agents and blocks in the environment vary across experiments. Mean and standard deviation are reported over 5 randomly initialized trials.

(# Obj, # Agents)	Success	Avg. Time (s)	Length (m)
$(5, 2)$	5/5	0.319 ± 0.055	10.95 ± 1.35
$(8, 2)$	5/5	0.976 ± 0.196	18.53 ± 3.83
$(11, 2)$	4/5	2.462 ± 0.516	27.532 ± 4.445
$(5, 3)$	5/5	0.365 ± 0.103	12.19 ± 1.14
$(8, 3)$	4/5	1.23 ± 0.363	21.50 ± 4.11
$(11, 3)$	4/5	2.650 ± 1.051	30.570 ± 3.783
$(5, 4)$	5/5	0.513 ± 0.184	14.22 ± 1.27
$(8, 4)$	4/5	1.43 ± 0.243	22.70 ± 2.57
$(11, 4)$	5/5	3.594 ± 1.052	33.480 ± 3.998

in shorter paths and better coordination. Fig. 4 illustrates this behavior in a block-stacking task: when a perturbation occurs, agent 1 backtracks to the pickup stage. Without agent 1's block, agent 2 cannot complete its subgoal and must delay execution until agent 1 finishes. In this way, GoC-MPC naturally handles interdependent delays and action reordering caused by disturbances.

The time metrics for GoC-MPC were higher in static settings (Table I–II) than in disturbance experiments (Table III). This is because, in static settings, GoC-MPC was initialized with additional stages to enhance robustness, which increased execution time compared to disturbance scenarios.

TABLE V: Real-world Success Rates and Computation Time for our method.

Task	Max Time (s) ↓	Avg. Time (s) ↓	Success ↑
Block-Stacking	0.493 ± 0.018	0.093 ± 0.027	$3/5$
Pick-and-Pour	0.373 ± 0.071	0.065 ± 0.012	$5/5$
Cloth-Folding	0.134 ± 0.009	0.057 ± 0.002	$5/5$

C. Scalability Analysis

To assess scalability, we evaluated GoC-MPC's robustness across different numbers of objects and agents in the scene. As the number of objects increased, the average time and path lengths naturally increased, but remained low-enough on average to still support online reactive execution. Success rates also remained high; however, it was occasionally observed with large problems that the waypoints problem could exhaust the allowed number of solver iterations, resulting in some cycles failing to update the planned waypoints. The majority of trials remained successful even with the increased scale.

D. Real-World evaluation

Finally, we deployed GoC-MPC on a dual-UR5e manipulator system to evaluate its performance on real-world bi-manual manipulation tasks. As shown in Fig. 3, the tasks included transferring liquid between two cups, stacking three blocks, and folding a tablecloth. Demonstration videos are available in the supplementary material.

Workspace keypoints were tracked from a 30 Hz RGB-D stream using a RealSense D455 camera. For block-stacking and pick-and-pour, block and cup centroids were tracked with SAM2 [32], while for tablecloth folding we additionally employed a Kanade–Lucas–Tomasi tracker [33], [34] to track corner points. We report maximum and average times per GoC-MPC cycle, along with task success rates across five randomly initialized trials (Table V). Overall, GoC-MPC achieved comparable performance in real-world experiments and simulation, succeeding in nearly all trials. The only failures occurred in two block-stacking trials due to severe block occlusions during execution. These results demonstrate that our approach transfers effectively to physical systems while retaining low computational times and short path lengths.

V. FUTURE WORK & CONCLUSIONS

In this work, we presented GoCs and GoC-MPC, a novel and effective algorithm for multi-agent reactive TAMP. As a generalization of prior work on sequences-of-constraints, GoCs automatically inherit existing constraint formulations that make them easily implemented for use in the real world without extensive modeling or data requirements. Our experiments additionally show that our approach can formulate and solve GoCs at almost real-time speeds. However, our method's performance can deteriorate depending on the external state estimation module and the initial plan skeleton.

We intend to explore integrating a learned perception module to overcome the former limitation, as well as discrete planning in-the-loop to overcome the latter.

REFERENCES

[1] M. Toussaint, "Logic-geometric programming: an optimization-based approach to combined task and motion planning," in *Proceedings of the 24th International Conference on Artificial Intelligence*, ser. IJCAI'15. AAAI Press, 2015, p. 1930–1936.

[2] T. Stouraitis, I. Chatzinikolaidis, M. Gienger, and S. Vijayakumar, "Online hybrid motion planning for dyadic collaborative manipulation via bilevel optimization," *IEEE Transactions on Robotics*, vol. 36, no. 5, pp. 1452–1471, 2020.

[3] T. Stouraitis, L. Yan, J. Moura, M. Gienger, and S. Vijayakumar, "Multi-mode trajectory optimization for impact-aware manipulation," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 9425–9432.

[4] M. Toussaint, J. Harris, J.-S. Ha, D. Driess, and W. Hönig, "Sequence-of-constraints mpc: Reactive timing-optimal control of sequential manipulation," 2022. [Online]. Available: https://arxiv.org/abs/2203.05390

[5] W. Huang, C. Wang, Y. Li, R. Zhang, and L. Fei-Fei, "Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation," 2024. [Online]. Available: https://arxiv.org/abs/2409.01652

[6] 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, pp. 189–208, 1971. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0004370271900105

[7] D. S. Weld, "An introduction to least commitment planning," *AI Mag.*, vol. 15, no. 4, p. 27–61, Dec. 1994. [Online]. Available: https://doi.org/10.1609/aimag.v15i4.1109

[8] J. Hoffmann, "Ff the fast-forward planning system," *AI Mag.*, vol. 22, no. 3, p. 57–62, Sep. 2001. [Online]. Available: https://doi.org/10.1609/aimag.v22i3.1572

[9] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[10] A. H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip, "Motion planning networks," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 2118–2124.

[11] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, no. 1, pp. 440–448, Jun. 2020. [Online]. Available: https://ojs.aaai.org/index.php/ICAPS/article/view/6739

[12] B. Vu, T. Migimatsu, and J. Bohg, "Coast: Constraints and streams for task and motion planning," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 14 875–14 881.

[13] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *Int. J. Rob. Res.*, vol. 37, no. 10, p. 1134–1151, Sep. 2018. [Online]. Available: https://doi.org/10.1177/0278364918761570

[14] Z. Zhao, S. Cheng, Y. Ding, Z. Zhou, S. Zhang, D. Xu, and Y. Zhao, "A survey of optimization-based task and motion planning: From classical to learning approaches," *IEEE/ASME Transactions on Mechatronics*, vol. 30, no. 4, pp. 2799–2825, 2025.

[15] L. Antonyshyn, J. Silveira, S. Givigi, and J. Marshall, "Multiple mobile robot task and motion planning: A survey," *ACM Comput. Surv.*, vol. 55, no. 10, Feb. 2023. [Online]. Available: https://doi.org/10.1145/3564696

[16] S. C. Livingston, R. M. Murray, and J. W. Burdick, "Backtracking temporal logic synthesis for uncertain environments," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 5163–5170.

[17] M. Guo, K. H. Johansson, and D. V. Dimarogonas, "Revising motion planning under linear temporal logic specifications in partially known workspaces," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 5025–5032.

[18] K. He, A. M. Wells, L. E. Kavraki, and M. Y. Vardi, "Efficient symbolic reactive synthesis for finite-horizon tasks," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE Press, 2019, p. 8993–8999. [Online]. Available: https://doi.org/10.1109/ICRA.2019.8794170

[19] T. Migimatsu and J. Bohg, "Object-centric task and motion planning in dynamic environments," *IEEE Robotics and Automation Letters (R-AL)*, vol. 5, no. 2, pp. 844–851, 2020.

[20] C. V. Braun, J. Ortiz-Haro, M. Toussaint, and O. S. Oguz, "Rhh-lgp: Receding horizon and heuristics-based logic-geometric programming for task and motion planning," 2022. [Online]. Available: https://arxiv.org/abs/2110.03420

[21] T. Xue, A. Razmjoo, and S. Calinon, "D-lgp: Dynamic logic-geometric program for reactive task and motion planning," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 14 888–14 894.

[22] Y. Zhang, C. Pezzato, E. Trevisan, C. Salmi, C. H. Corbato, and J. Alonso-Mora, "Multi-modal mppi and active inference for reactive task and motion planning," *IEEE Robotics and Automation Letters*, vol. 9, no. 9, pp. 7461–7468, 2024.

[23] L. Manuelli, W. Gao, P. Florence, and R. Tedrake, "Kpam: Keypoint affordances for category-level robotic manipulation," in *Robotics Research*, T. Asfour, E. Yoshida, J. Park, H. Christensen, and O. Khatib, Eds. Cham: Springer International Publishing, 2022, pp. 132–157.

[24] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-maron, M. Giménez, Y. Sulsky, J. Kay, J. T. Springenberg, T. Eccles, J. Bruce, A. Razavi, A. Edwards, N. Heess, Y. Chen, R. Hadsell, O. Vinyals, M. Bordbar, and N. de Freitas, "A generalist agent," *Transactions on Machine Learning Research*, 2022, featured Certification, Outstanding Certification. [Online]. Available: https://openreview.net/forum?id=1ikK0kHjvj

[25] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. C. Burchfiel, and S. Song, "Diffusion Policy: Visuomotor Policy Learning via Action Diffusion," in *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023.

[26] I. Grossmann and J. Ruiz, *Generalized Disjunctive Programming: A Framework for Formulation and Alternative Algorithms for MINLP Optimization*, 11 2012, vol. 154, pp. 93–116.

[27] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, Mar. 2006. [Online]. Available: https://doi.org/10.1007/s10107-004-0559-y

[28] M. ApS, *MOSEK Optimizer API for C 11.0.28*, 2025. [Online]. Available: https://docs.mosek.com/latest/capi/index.html

[29] NVIDIA, "Isaac Sim." [Online]. Available: https://github.com/isaac-sim/IsaacSim

[30] C. Li, R. Zhang, J. Wong, C. Gokmen, S. Srivastava, R. Martín-Martín, C. Wang, G. Levine, M. Lingelbach, J. Sun, M. Anvari, M. Hwang, M. Sharma, A. Aydin, D. Bansal, S. Hunter, K.-Y. Kim, A. Lou, C. R. Matthews, I. Villa-Renteria, J. H. Tang, C. Tang, F. Xia, S. Savarese, H. Gweon, K. Liu, J. Wu, and L. Fei-Fei, "Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation," in *Proceedings of The 6th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, K. Liu, D. Kulic, and J. Ichnowski, Eds., vol. 205. PMLR, 14–18 Dec 2023, pp. 80–93. [Online]. Available: https://proceedings.mlr.press/v205/li23a.html

[31] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019. [Online]. Available: https://drake.mit.edu

[32] N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma, H. Khedr, R. Rädle, C. Rolland, L. Gustafson, E. Mintun, J. Pan, K. V. Alwala, N. Carion, C.-Y. Wu, R. Girshick, P. Dollár, and C. Feichtenhofer, "Sam 2: Segment anything in images and videos," *arXiv preprint arXiv:2408.00714*, 2024. [Online]. Available: https://arxiv.org/abs/2408.00714

[33] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *IJCAI'81: 7th international joint conference on Artificial intelligence*, vol. 2, 1981, pp. 674–679.

[34] C. Tomasi and T. Kanade, "Detection and tracking of point," *Int J Comput Vis*, vol. 9, no. 137-154, p. 3, 1991.