Self-Supervised Learning of Scene-Graph Representations for Robotic Sequential Manipulation Planning

Son-Tung Nguyen[1], **Ozgur S. Oguz**[1,2], **Valentin N. Hartmann**[1,3], **Marc Toussaint**[2,3]
[1]Machine Learning & Robotics Lab, University of Stuttgart,
[2]Max Planck Institute for Intelligent Systems,
[3]Learning and Intelligent Systems Group, TU Berlin

Abstract: We present a self-supervised representation learning approach for visual reasoning and integrate it into a nonlinear program formulation for motion optimization to tackle sequential manipulation tasks. Such problems have usually been addressed by combined task and motion planning approaches, for which spatial relations and logical rules that rely on symbolic representations have to be predefined by the user. We propose to learn relational structures by leveraging visual perception to alleviate the resulting knowledge acquisition bottleneck. In particular, we learn constructing *scene-graphs*, that represent objects ("red box"), and their spatial relationships ("yellow cylinder on red box"). This representation allows us to plan high-level discrete decisions effectively using graph search algorithms. We integrate the visual reasoning module with a nonlinear optimization method for robot motion planning and verify its feasibility on the classic blocks-world domain. Our proposed framework successfully finds the sequence of actions and enables the robot to execute feasible motion plans to realize the given tasks.

Keywords: Visual Reasoning, Representation Learning, Task & Motion Planning

1 Introduction

Sequential robotic manipulation tasks require long-term reasoning and planning capabilities. Autonomous agents have to keep track of available internal and external states and reason about how those states change if they interact with the environment. The spatial relations within the environment, including both the robot and the objects, determine the feasible actions and thus the sequence of decisions. Due to this interdependence, planning methods have to consider the logical state transitions and the geometric constraints jointly while searching for a solution. Even though classical planners in AI literature are very effective for high-level action planning [1, 2], geometric constraints and visual representations have been largely neglected [3, 4]. In essence, structured visual representations enable additional reasoning capabilities for autonomous robots to support their long-term decision-making skills.

Sequential manipulation problems require (*i*) planning high-level discrete actions that have to satisfy pre- and post-conditions, simultaneously with (*ii*) finding continuous motion paths that have to satisfy low-level constraints such as kinodynamic limits, or collision avoidance. To tackle this hybrid nature of such problems, existing combined task and motion planning (TAMP) methods rely on symbolic representations and first-order predicate rules for the action planning while using either sampling-based [5, 6, 7, 8, 9] or optimization-based [10] planners for computing the robot motion path. Predicates for the spatial relationships and the action transition rules that determine the logical and geometric constraints are defined a priori by the user. To enhance their autonomy, robots should learn and plan based on relational representations, transition rules, and their implied constraints by relying on their sensory data. Acquisition of such relational models poses a major challenge for providing more autonomy for the robots and also for scaling TAMP frameworks. Planning high-level actions autonomously could be facilitated by learning spatial relations in a self-supervised fash-

4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.

ion. The relational structures, in turn, can be integrated with effective motion planners to realize sequential manipulation tasks.

We present a high-level planner that perceives visual inputs as commands and realizes the high-level action sequence. Our planner first captures the symbolic representations of the input scenes in the form of *scene graphs* (SGs) [11]. This is possible via an encoder network that transforms visual input into a local (relative to the view) coordinate system. Inside this imaginary coordinate system, we use the k-means algorithm to group coordinates into clusters, then effectively infer spatial relationships among clusters and objects in each of the cluster. These spatial relationships are built into triples which are combined to form a SG. Since these SG representations are symbolic, we can use generic graph exploration methods to search for the optimal action sequence to reach the goal state from an initial scene. This action sequence is then passed to a motion-planner to find a plan and subsequently execute it. We demonstrate our approach on the classical example of the Blocks-world task [12], i.e., given several stacks of different objects, the agent reorders them autonomously to obtain a final configuration. Thus, as the main contributions of this work:

- We implement an encoder network as a visual module which allows for learning relative coordinates in a *self-supervised* way,
- We exploit this visual module to map a scene image into a *scene-graph* which is an invariant representation feasible for planning,
- We present a unified framework comprising *learning* and *TAMP* directly from visual data to solve sequential robotic manipulation tasks effectively.

2 Related Work

Symbolic Planning Our work is partially inspired by the use of symbolic planning in various domains [4, 13, 14]. Symbolic planning with graph search algorithms are usually optimal and fast [15], thus particularly suitable for long-horizon planning. However, symbolic planners require discrete state representations, and hence do not work out of the box with continuous representations. Acquiring such a representation only from visual input is a challenging task. To overcome this challenge, recent work either relaxes the discrete requirement of symbolic planners [14] or enforces discrete state spaces [3, 4] using a Gumbel-Softmax [16, 17] function. However, since both methods work with non-symbolic, unstable representations, the constructed action models are suboptimal [14] and not robust [3, 4]. Since symbolic representations are the core component of such planners, we propose a method to acquire these representations reliably.

Sequential Manipulation Planning Sequential planning in robotics has been mostly investigated in two largely disconnected subfields: *task and motion planning* and *reinforcement learning*. Task and motion planning methods tackle the problem of integrating high-level actions with low level constraints effectively by relying on the domain knowledge. Such a method incorporates action constraints as (in)equality constraints into either a sampling-based motion planner [5, 6, 7, 8, 9] or a non-linear program (NLP) [10, 18]. This program then jointly optimizes the action feasibility and the motion trajectory to output a motor plan. However, TAMP formulations require the user to prespecify possible actions and their pre- and post-conditions to impose constraints. Most importantly, the input state and the desired output state need to be *symbolic*, rather than more flexible definitions, such as images in recent reinforcement learning approaches.

Deep reinforcement learning approaches have recently been successfully used for robot skill acquisition tasks by learning control policies for motor skills from image pixels [19, 20, 21]. While showing great results in learning *manipulation skills*, these approaches are not particularly suitable to find a sequence of high-level actions that fulfill a set of goals (e.g., given as a set of spatial relations, *"yellow cylinder on red box"* and *"red box on first stack"*), from a given initial state that requires multiple different actions, such as *pick*, *place*, *push*, etc. This is partially due to the fact that each of the actions impose different geometric and dynamic constraints, leading to an impractical amount of data needed to train such a policy.

Combining the two approaches helps to learn heuristics to guide the exploration of the possible actions [22, 23, 24], or to learn new action-primitives [25]. In this work, we propose a learning approach to find a representation of the scene that supplements combined task and motion planning.

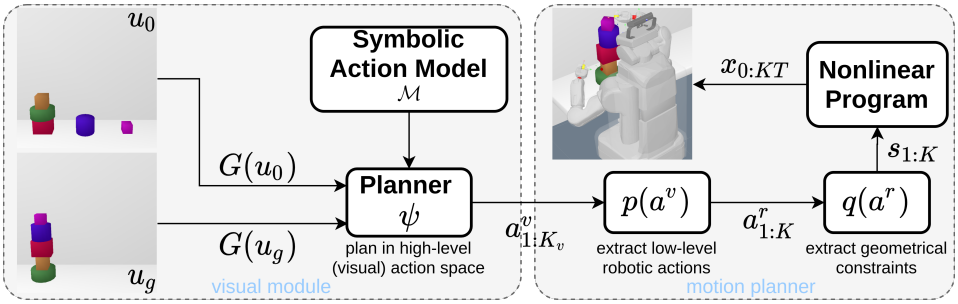

Figure 1: Flow of the proposed framework. *Left*: High-level action planning: a visual reasoning module consisting of a scene graph (SG) representation (Sec. 4.1.3) and an action model (Sec. 4.2), and *Right*: Motion optimization: a nonlinear programming (NLP) component (Sec. 3).

3 Problem Statement

This works relies on the NLP formulation of the Logic-Geometric Programming (LGP) framework [10, 26] to combine visual reasoning for task planning with an effective constraint optimization method for path planning. LGP formulates a constrained optimization problem based on the logical and geometric constraints imposed while solving a combined task and motion planning problem. We focus on solving similar TAMP problems for sequential robotic manipulation. However, our objective is to complement physical reasoning by visual perception for high-level action selection. In essence, the logical planning is delegated to a visual reasoning component, whereas existing TAMP methods, including the LGP, use first-order predicate language rules. Accordingly, in our formulation only the geometric constraints are imposed by this high-level planning on the NLP. This modification on planning still allows us to use the original NLP formulation of the LGP only with slight changes (Fig. 1). Let $\mathcal{X} \subset \mathbb{R}^d \times SE(3)^N$ be the configuration space of N rigid objects and a robot with d degrees-of-freedom with initial condition x_0. Given a goal description g, we aim to find a sequence of actions $a^r_{1:K}$ and a path $x : [0, KT] \to \mathcal{X}$ that minimizes a state-dependent cost:

$$
\begin{aligned}
&\min_{\substack{x,K,\\ a^r_{1:K}, s_{1:K}}} \int_0^{KT} c(\overline{x}(t), s_{k(t)})\,\mathrm{d}t \\
&\text{s.t.} \quad x(0) = x_0, \qquad s_0 = \tilde{s}_0, \qquad h_{\text{goal}}(x(KT), g) = 0 \\
&\qquad \forall t \in [0, KT]: \quad h_{\text{path}}\big(\overline{x}(t), s_{k(t)}\big) = 0, \quad g_{\text{path}}\big(\overline{x}(t), s_{k(t)}\big) \le 0 \\
&\qquad \forall k \in 1, \dots, K: h_{\text{switch}}(\hat{x}(t), a^r_k) = 0, \quad g_{\text{switch}}(\hat{x}(t), a^r_k) \le 0 \\
&\qquad\qquad a^r_k \in \mathcal{A}^r(s_{k-1}), \qquad s_k \in q(s_{k-1}, a^r_k),
\end{aligned} \tag{1}
$$

where $\overline{x} = (x, \dot{x}, \ddot{x})$ and $\hat{x} = (x, \dot{x})$. $\tilde{s}_0$ describes the initial geometric constraints, g_{path} and h_{path}, the (in)equality constraints for the motion path, and h_{switch} and g_{switch}, the transition conditions between kinematic modes, respectively. Symbolic state s_k defines the constraints at each phase, and determines the available actions from the set $\mathcal{A}^r$. The key difference is that we partially decouple the logical action search component from the NLP. As the spatial relations are defined by visual representations generated from images, high-level action planning is solved by the graph search algorithm acting on the SGs (Sec. 4). In particular, the goal of the visual planner ψ is to find an action sequence $a^v_{1:K^v}$, given an image of an initial scene u_0, and a desired scene u_{g},

$$
\begin{aligned}
&\operatorname{argmin}_{a^v_{1:K^v}} \sum_{i=1}^{K^v} c^v(z_{i-1}, a^v_i) \\
&\text{s.t. } z_0 = \mathcal{G}(u_0), \qquad z_N = \mathcal{G}(u_{\text{g}}), \\
&\qquad z_{i+1} = \mathcal{M}(z_i, a^v_i), \quad a^v_i \in \mathcal{A}^v
\end{aligned} \tag{2}
$$

where c^v defines the optimality criterion, z_0, z_N is the initial and final SG representations encoded by the visual module $\mathcal{G}$, $\mathcal{M}$ is a symbolic action model, $\mathcal{A}^v$ is a fixed set of high-level actions, and K^v is the number of high-level actions. Given the action sequence found by the visual planner, we extract low-level actions for the robot $a^r_{1:K} = p(a^v_{1:K^v})$ and the geometric constraints $s_{1:K} = q(a^r_{1:K})$

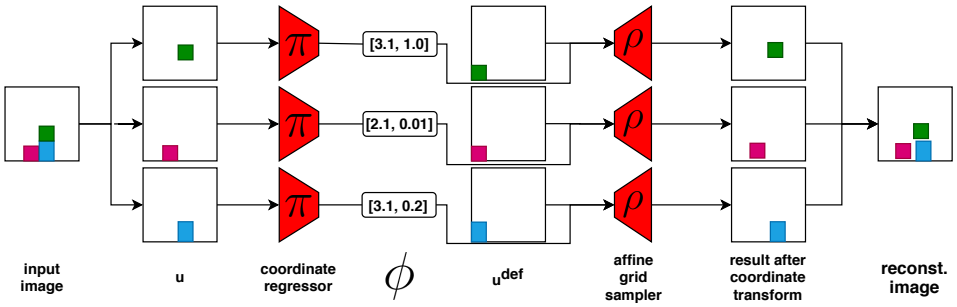

Figure 2: A forward pass from the input scene image to the reconstructed scene image, through the visual module $\mathcal{G}$ with the coordinate regressor π and the affine grid sampler ρ.

(Fig. 1). Note that K does not necessarily have to be equal to K^v, e.g., a single high-level *move* action might be broken down to two low-level robot *pick* and *place* actions. Those actions $a^r_{1:K}$ along with their implied geometric constraints $s_{1:K}$ form the constrained optimization problem to be solved to compute the optimal motion path for the given plan that realizes the goal.

4 Method

We present a unified framework to solve sequential robotic manipulation tasks. The three main components of this framework are a visual module $\mathcal{G}$, a symbolic action model $\mathcal{M}$, and an NLP solver for motion planning. The basic forward pass from an input scene image to a goal scene image is as follows: (*i*) we capture scene graphs (SGs) from the image pair (u_0, u_{goal}) using the visual module $\mathcal{G}$ (Fig. 2), then (*ii*) use the action model $\mathcal{M}$ to realize a high-level plan from u_0 to u_{goal}, and last, (*iii*) exploit the NLP formulation to convert the high-level plan into atomic actions along with their implied constraints and compute the optimal motion path for the realization of the plan by the robot.

4.1 Visual module $\mathcal{G}$

The visual module $\mathcal{G}$ assumes that input scene images contain bounding boxes of N scene objects. This assumption is reasonable since such bounding boxes are readily available within a simulation environment, and also there exists several robust methods to acquire them for real-world tasks [27, 28, 29]. After computing the bounding boxes, we split the scene image into N different images. Each of these images contains exactly one bounding box, i.e., a mask, of one scene object. The module $\mathcal{G}$ then consumes one image at a time to encode a coordinate vector for this specific object. This coordinate vector lives in a relative coordinate system which has the origin at the bottom left corner of image. We proceed to present first, how to realize $\mathcal{G}$, and then how to train the visual module in a complete self-supervised manner, and last, how to transform N coordinate vectors to a SG describing spatial relations between N objects.

4.1.1 Architecture overview

The visual module $\mathcal{G}$ consists of two sub-modules: coordinate regressor π and affine grid sampler ρ (illustrated by the red boxes in Fig. 2). Note that the visual module $\mathcal{G}$ is similar to the Spatial Transformer Network by Jaderberg et al. [30]. The only difference is that $\mathcal{G}$ only works with the translation vector ϕ instead of the full 3×3 transformation matrix. Next, we explain the details of those components and how they are combined.

π - Coordinate regressor: The coordinate regressor infers a vector ϕ from a mask image u_i, i.e., $\phi = \pi(u_i)$ where $\phi \in \mathbb{R}^2$, $u_i \in \mathbb{R}^{128\times128\times3}$. The vector ϕ denotes how far the object should be from the bottom-left corner (depicted in the fourth column of Fig. 2). The backbone of the π module is a Res-Net34 network [31]. This module is pre-trained [1] on the ImageNet dataset [33]. We replace the last layer with a fully-connected layer to output ϕ. Note that it is possible to train ϕ using the gradient feedback from ρ (detailed below) via Eq. (3).

[1] The pre-trained weights are available in the TorchVision library [32].

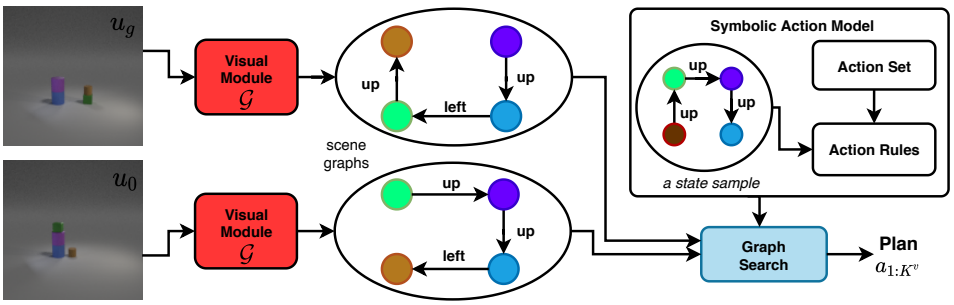

Figure 3: An overview of how the graph search is done using the symbolic action model $\mathcal{M}$ and the SG representation.

ρ **- Affine grid sampler:** The affine grid sampler takes the vector ϕ and transforms a default matrix u^{def} (depicted in the fifth column of Fig. 2) into the desired location which is implied by ϕ, i.e., $\tilde{u}_i = \rho(\phi, u_i^{\text{def}})$. The module ρ is differentiable, hence providing feedback on how to *spatially* transform the default matrices u_i^{def}, i.e., optimizing ϕ from π according to a label signal.

4.1.2 Training objective

We train the visual module in a complete *self-supervised* manner. Our training objective is

$$l = l_{\text{mse}}\left(\sum_{i=1}^{N}\tilde{u}_i, \sum_{i=1}^{N} u_i\right) + \sum_{i=1}^{N} l_{\text{mse}}\left(\tilde{u}_i, u_i\right) \tag{3}$$

where l_{mse} is the mean squared loss, u_i and $\tilde{u}_i$ are the ground-truth and predicted images, which contain only the bounding box of scene object o_i. $\sum_{i=1}^{N}\tilde{u}_i$ and $\sum_{i=1}^{N} u_i$ are the reconstructed and true scene images, respectively. We compute $\tilde{u}_i$ by $\tilde{u}_i = \rho(\pi(x_i), u_i^{\text{def}})$ where u_i^{def} is the default matrix in which the bounding box of o_i is located at the bottom left corner (Fig. 2).

4.1.3 Coordinates to SG

Our SGs are a set of triples $\{o_1, e, o_2\}$, where $e \in \{\text{left,up}\}$ describing the spatial relations between object o_1 and object o_2 (Fig. 3). We first use the k-means clustering algorithm [34] to group horizontal coordinates into m clusters, where m is the number of stacks. Here, m is prespecified, however can be discovered automatically (see sec. 7.1 of the sup. mat.). Then, within each cluster, we simply construct *up* relations by comparing vertical coordinates. Finally, we add *left* relations by comparing horizontal coordinates of the bottom objects of each cluster. Note that *down* and *right* relations are covered as well thanks to this SG representation of triples.

4.2 Symbolic action model

After capturing the SGs for the initial and goal scene images, we deploy an action model to search for the action sequence. Note that this action model is *oracular* since we pre-define a set of actions $\mathcal{A}^v = \{(p, q) | 1 \leq p, q \leq m\}$, where m is the number of stacks in the scene. This means that if the visual module $\mathcal{G}$ correctly captures the SGs for the image pair (u_0, u_{goal}), the action model is guaranteed to find the optimal plan if the goal can be satisfied. This is possible as our planner relies on SG representation for intermediate states. To find the action sequence, we start from the input SG, use this action model to explore the neighboring graphs from the action set, and add these neighboring graphs into a queue-like data structure (Fig. 3). The search can be *breadth-first* or *depth-first*, although we use the former in our experiments. The search stops when it reaches the goal SG z_{goal} which is either encoded by u_{goal} or user-specified (see Fig. 5 for sample plans).

4.3 Motion planner

Actions from the high-level plan of the visual module are divided into micro actions, $a_{1:K}^r = p(a_{1:K^v}^v)$, which in turn define the geometric constraints $s_{1:K} = q(a_{1:K}^r)$ (Sec. 3) [10]. Given those

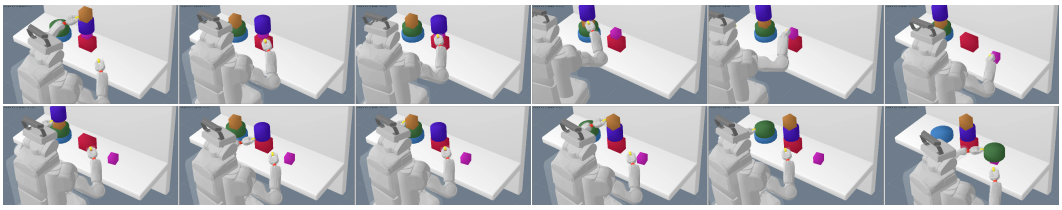

Figure 4: Keyframes for each *pick* and *place* action starting from (top-left) the initial state till (bottom-right) the goal.

Table 1: Performance on different evaluation datasets.

Dataset	**Accuracy**	**IoU**
$\mathcal{D}_{\text{pbw-all}}$ (6 objects, 3 stacks)	0.889	0.740
$\mathcal{D}_{\text{sim-all}}$ (6 objects, 3 stacks)	0.867	0.739
$\mathcal{D}_{\text{sim-20k}}$ (7 objects, 3 stacks)	0.884	0.714
$\mathcal{D}_{\text{sim-20k}}$ (6 objects, 3 stacks, camera view $+2°$)	0.882	0.584
$\mathcal{D}_{\text{sim-20k}}$ (6 objects, 3 stacks, camera view $+10°$)	0.881	0.561
$\mathcal{D}_{\text{sim-20k}}$ (6 objects, 3 stacks, camera view $+15°$)	0.843	0.529
$\mathcal{D}_{\text{sim-all}}$ (6 objects, 3 stacks, trivial COCO [37] masks)	0.873	0.693

constraints, we use KOMO (k-order Markov optimizer) [35], and the changes introduced in [26] which make the solver more robust, to solve the TAMP problem (Eq. (1)). KOMO represents the trajectory directly in configuration space, and uses a Gauss-Newton solver to solve the discretized optimization problem.

5 Experiments

As a representative sequential decision-making and manipulation scenario, we tested our methods on a stacking problem with the PR2 robot in simulation (Fig. 4). For the *visual module*, we also evaluated its robustness and generalization capabilities on a photorealistic image dataset in detail. Since our method is self-supervised, and is able to re-calibrate to adapt to novel scenes, we do not aim for a universal generalization of all scene objects or camera views. Having said that, we perform experiments to investigate the robustness properties for practical use-cases. This analysis can be useful to understand when the model needs to be re-calibrated, which can be achieved automatically using the *Intersection-over-Union* (IoU) indicator [36].

5.1 Setup

We evaluate our framework using the Blocks-world task [12] where we fix the number of stacks $m = 3$, and the set of actions $\mathcal{A}^v = \{(1,2),(1,3),(2,1),(2,3),(3,1)(3,2)\}$. The action $a_i^v = (p,q)$ means *move* the object at the top of stack p to stack q. We utilize two different graphics engine to collect the data. The first engine [38] is used to render photorealistic images to collect the dataset $\mathcal{D}_{\text{pbw}}$. The second dataset $\mathcal{D}_{\text{sim}}$ [10] is obtained from our own simulator which is visually simpler but allows for robotic manipulation planning and control experiments. We train two $\mathcal{G}$'s on the state space of the 5-object task and evaluate on the 6-object task on $\mathcal{D}_{\text{pbw}}$ and $\mathcal{D}_{\text{sim}}$.

5.2 Results

Since our action model $\mathcal{M}$ is oracular, the planning accuracy depends on whether $\mathcal{G}$ correctly captures the scene graphs (SGs) of the desired scene images. Therefore, we focus our analysis on the performance of $\mathcal{G}$. We report the IoU metric (Fig. 9) and the SG prediction accuracy for our experiments (Tab. 1). The IoU is computed between the true u_i and the predicted $\tilde{u}_i$ mask images.

Large scale analysis: We show that our method, when trained on the task of 5 objects and 3 stacks, generalizes well to higher dimensional task, e.g., 6 or 7 objects. This is reflected by the good SG prediction accuracy and IoU metric (first 3 rows of Tab. 1). During plan execution, the robot may

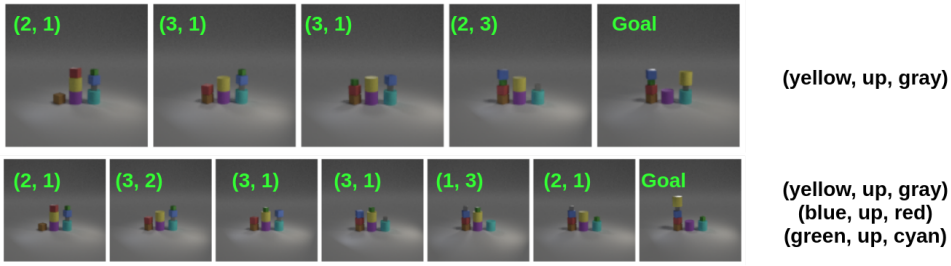

Figure 5: Three sample plans. Initial images u_0 are the left-most images of each row. Desired relations z_{goal} (the right-most column) are specified as textual commands instead of desired images u_{goal}, which is also possible (see Fig. 12 in Appendix). (*Green, top-left corner*) Action that transitions the current state to the next. Action notation follows Sec. 4.2 and Sec. 5.1.

change its camera view, thus we also analyze whether this change may affect our method. We test the same $\mathcal{G}$ on datasets with different views. We observe a decreasing trend in IoU, however, the performance is still sufficient to retain accuracy in predicting SGs (Tab. 1 rows 4-6). We also replace the bounding boxes of objects in $\mathcal{D}_{\text{sim-all}}$ with random object masks from the COCO dataset [37]. We train the same $\mathcal{G}$ using these random masks, and obtain similar results (see last row Tab. 1). This suggests that our framework works with arbitrary object appearance.

Fine scale analysis: We conduct fine-scale experiments to understand whether our model generalizes to different shapes or colors. We artificially construct objects of 3 different shapes (circle, square, and triangle) and 14 different colors. We evaluate the visual module $\mathcal{G}$ which is trained on $\mathcal{D}_{\text{sim}}$, and report the variance of the IoU as 0.0055 for shape and 0.001 for color changes. This analysis suggests that our model is more sensitive to change in color rather than in shape.

Robot manipulation experiments: We test the proposed framework on a robotic sequential manipulation task. The setting is similar to the blocks-world problem but constructed in our simulator. The robot successfully executes plans proposed by the visual reasoning module (see supplementary video). Even though the original LGP formulation struggles to solve problems involving many objects (e.g., # of objects $N \geq 5$) and which requires long action sequences (e.g., action sequence length $K \geq 6$) [22], our framework efficiently solves such problems in this blocks-world domain, where we have 6 & 7 objects in the scene and the solutions require action sequences with length $K \geq 6$. We note that comparing the runtime performance of our method to the original LGP is not informative on its own, as this work is based on a more robust version of the optimizer described in [26]. We merely want to point out that the visual representations learned by our method proves to be a useful and feasible component to support task and motion planning approaches for practical scenarios where the robot has to rely on its visual perception.

Extension to realistic tasks: For applicability of our approach to real-world settings, we show a couple of straightforward extensions. First, we incorporate geometric constraints in addition to spatial relations among objects, e.g., spherical objects cannot be used as a base (Fig. 6). These constraints help to reason about stacking stability, thus allow for safer manipulation. Since each object has different geometric and dynamical properties, these can also be inferred by more advanced object detection methods, which in turn supports the intuitive physical reasoning of the autonomous agent [39]. We suggest one possible extension following this idea in Sec. B.2 of the appendix. Second, our method can be further developed to reason about the 3-D world using a depth sensor. By leveraging this depth information, we implemented a *pre-checking* subroutine to handle occlusion. Fig. 7 illustrates some planning samples under this setting. We detail the pre-checking subroutine in Sec. B.3 of the appendix.

6 Discussion

Here we discuss the main limitations and assumptions of our work, and provide possible directions for future work to address those issues.

We make two assumptions within this work. First, we assume that we have access to the bounding boxes of scene objects rather than just raw scene images. From such a segmented scene image, we

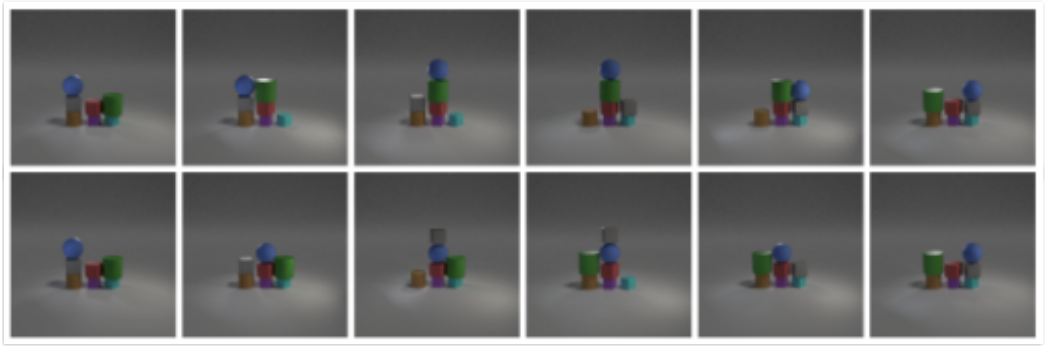

Figure 6: A sample plan when stability is enforced (*top*) and not enforced (*bottom*). Note that the former plan does not involve stacking any object on top of the sphere.

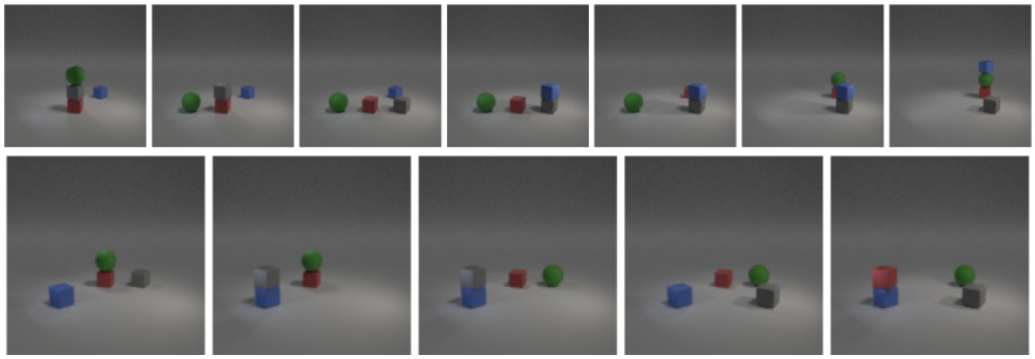

Figure 7: Plan sequences for 2 scenarios with 3-D relationships, starting from (*left-most*) the initial scene till reaching (*right-most*) the goal.

then extract individual objects to infer the respective coordinates. Even though this is a reasonable assumption as there are many state-of-the-art object detection methods [27, 28, 29], it can be relaxed by adapting an unsupervised scene decomposition technique [40].

Second, our current setup is highly structured, where we assume a fixed setup of 3 stacks in the scene, thus set $k = 3$ in the k-means algorithm. However, this is not a hard requirement since it is possible to infer k using a simple image processing algorithm (see Sec. B.1).

Our experimental results demonstrate a good accuracy for novel environmental settings (6 & 7 objects). To deal with a possibly incorrectly predicted image-pair, two approaches are feasible: *(i)* Our planner can output a planning solution with a confidence score using the IoU metric, which indicates the plan reliability. This might trigger either a view change on the robot for better observation, or a query for clarifying the goal, and *(ii)* our framework can be extended to work in a reactive way by closing the loop from the robotic action execution to the visual reasoning part. In essence, the visual planner can verify if the plan is executed properly and adjust the plan whenever necessary.

7 Conclusion

This paper proposes a simple yet useful visual reasoning method which we combine with a nonlinear program to solve sequential robot manipulation tasks. The scene-graph representations are learned in a self-supervised fashion, which encourages the integration of such spatial relation learning approaches into task and motion planning methods. These representations not only are interpretable, but also have the potential to generalize to novel settings due their invariance property. Relational structures learned autonomously by robots have the potential to improve the long-term autonomy of intelligent agents.

Acknowledgments

The research has been supported by the German Research Foundation (DFG) under Germany's Excellence Strategy – EXC 2120/1 – 390831618, and via the Max Planck Fellowship (MPI for Intelligent Systems).

References

[1] J. Hoffmann and B. Nebel. The ff planning system: Fast plan generation through heuristic search. *J. Artif. Int. Res.*, 14(1):253–302, May 2001. ISSN 1076-9757.

[2] M. Helmert and C. Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In *ICAPS*, 2009.

[3] M. Asai and A. Fukunaga. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *AAAI*, pages 6094–6101. AAAI Press, 2018.

[4] M. Asai. Unsupervised grounding of plannable first-order logic representation from images. In *ICAPS*, pages 583–591. AAAI Press, 2019.

[5] T. Lozano-Pérez and L. P. Kaelbling. A constraint-based method for solving sequential manipulation planning problems. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, September 14-18, 2014*, pages 3684–3691. IEEE, 2014.

[6] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 639–646. IEEE, 2014.

[7] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling. Ffrob: An efficient heuristic for task and motion planning. In *Algorithmic Foundations of Robotics XI*, pages 179–195. Springer, 2015.

[8] K. Hauser and J.-C. Latombe. Multi-modal motion planning in non-expansive spaces. *The International Journal of Robotics Research*, 29(7):897–915, 2010.

[9] Z. Kingston, M. Moll, and L. E. Kavraki. Sampling-based methods for motion planning with constraints. *Annual Review of Control, Robotics, and Autonomous Systems*, 2018.

[10] M. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. In *Proc. of Robotics: Science and Systems*, 2018.

[11] J. Johnson, R. Krishna, M. Stark, L. Li, D. A. Shamma, M. S. Bernstein, and F. Li. Image retrieval using scene graphs. In *CVPR*, pages 3668–3678. IEEE Computer Society, 2015.

[12] N. Gupta and D. S. Nau. On the complexity of blocks-world planning. *Artif. Intell.*, 56(2-3): 223–254, 1992.

[13] D. Driess, O. S. Oguz, and M. Toussaint. Hierarchical task and motion planning using logic-geometric programming (HLGP). *RSS Workshop on Robust TAMP*, 2019.

[14] D. Huang, D. Xu, Y. Zhu, A. Garg, S. Savarese, L. Fei-Fei, and J. C. Niebles. Continuous relaxation of symbolic planner for one-shot imitation learning. In *IROS*, pages 2635–2642. IEEE, 2019.

[15] M. Helmert. The fast downward planning system. *CoRR*, abs/1109.6051, 2011.

[16] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. In *ICLR (Poster)*. OpenReview.net, 2017.

[17] C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR (Poster)*. OpenReview.net, 2017.

[18] S. Zimmermann, G. Hakimifard, M. Zamora, R. Poranne, and S. Coros. A multi-level optimization framework for simultaneous grasping and motion planning. *IEEE Robotics and Automation Letters*, 5(2):2966–2972, 2020.

[19] C. Finn and S. Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793, 2017.

[20] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems, 2020.

[21] A. Srinivas, M. Laskin, and P. Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning, 2020.

[22] D. Driess, O. S. Oguz, J.-S. Ha, and M. Toussaint. Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning. In *Proc. of the International Conference on Robotics and Automation (ICRA)*, 2020.

[23] R. Chitnis, D. Hadfield-Menell, A. Gupta, S. Srivastava, E. Groshev, C. Lin, and P. Abbeel. Guided search for task and motion plans using learned heuristics. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 447–454. IEEE, 2016.

[24] B. Kim, Z. Wang, L. P. Kaelbling, and T. Lozano-Pérez. Learning to guide task and motion planning using score-space representation. *The International Journal of Robotics Research*, 38 (7):793–812, 2019.

[25] Z. Wang, C. Reed Garrett, L. Pack Kaelbling, and T. Lozano-Pérez. Learning compositional models of robot skills for task and motion planning. *arXiv e-prints*, art. arXiv:2006.06444, June 2020.

[26] V. N. Hartmann, O. S. Oguz, D. Driess, M. Toussaint, and A. Menges. Robust task and motion planning for long-horizon architectural construction planning. *arXiv preprint arXiv:2003.07754*, 2020.

[27] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask R-CNN. In *ICCV*, pages 2980–2988. IEEE Computer Society, 2017.

[28] R. B. Girshick. Fast R-CNN. In *ICCV*, pages 1440–1448. IEEE Computer Society, 2015.

[29] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In *NIPS*, pages 91–99, 2015.

[30] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. In *NIPS*, pages 2017–2025, 2015.

[31] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778. IEEE Computer Society, 2016.

[32] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, pages 8024–8035, 2019.

[33] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.*, 115(3):211–252, 2015.

[34] S. P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inf. Theory*, 28(2):129–136, 1982.

[35] M. Toussaint. KOMO: Newton methods for k-order Markov constrained motion problems. e-Print arXiv:1407.0414, 2014.

[36] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. D. Reid, and S. Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *CVPR*, pages 658–666. Computer Vision Foundation / IEEE, 2019.

[37] T. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. In *ECCV (5)*, volume 8693 of *Lecture Notes in Computer Science*, pages 740–755. Springer, 2014.

[38] M. Asai. Photo-Realistic Blocksworld Dataset. *arXiv preprint arXiv:1812.01818*, 2018.

[39] F. Baradel, N. Neverova, J. Mille, G. Mori, and C. Wolf. Cophy: Counterfactual learning of physical dynamics, 2019.

[40] C. P. Burgess, L. Matthey, N. Watters, R. Kabra, I. Higgins, M. Botvinick, and A. Lerchner. Monet: Unsupervised scene decomposition and representation. *CoRR*, abs/1901.11390, 2019.

[41] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

[42] S. W. Kim, Y. Zhou, J. Philion, A. Torralba, and S. Fidler. Learning to Simulate Dynamic Environments with GameGAN. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020.

[43] R. Fisher, S. Perkins, A. Walker, and E. Wolfart. Connected component labeling. In *IEE International Conference on Intelligent Systems*, pages 300–350, 2003.

[44] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 4(2):100–107, 1968.

A Appendix

A.1 Implementation

Our source code can be found in https://github.com/sontung/location-based-generative for the visual module, and in https://github.com/MarcToussaint/rai for the motion planning with nonlinear optimization.

A.2 $\mathcal{G}$-network architecture and training

To implement π, we replace the last layer of ResNet-34 [31] with one sigmoid-activated fully-connected layer $FC(512, 2)$. Suppose the output of π: $\phi = [a, b]$, our 2×3 transformation matrix α is $\alpha = \begin{pmatrix} 1 & 0 & -1.6a \\ 0 & 1 & 1.6b \end{pmatrix}$. Here 1.6 is our tuned hyper-parameter to make sure that ϕ covers a good area of the 128×128 image plane. We then pass α to the grid sampler ρ to transform the default matrix u^{def} to the desired location implied by ϕ. Tab. 2 details our training hyper-parameters for training $\mathcal{G}$.

Table 2: Training-related hyper-parameters

Parameter	**Value**
Optimizer	Adam [41]
Learning rate	1×10^{-3}
Weight decay	1×10^{-4}
Batch size	16

A.3 Ambiguity of Relations

Scene graph representation only takes into account the relative *spatial* relations, therefore does not express exact positional information. This problem creates ambiguity in which two different states may share the same representation (Fig. 8). Although this is not critical, we find a workaround to disambiguate the scene graph by adding imaginary bases. We then add one more relation for every base object, and also remove all the *left* relations as they become redundant. Note that the *left*/*right* relations can still be used, e.g., for such bases, in unstructured environments.

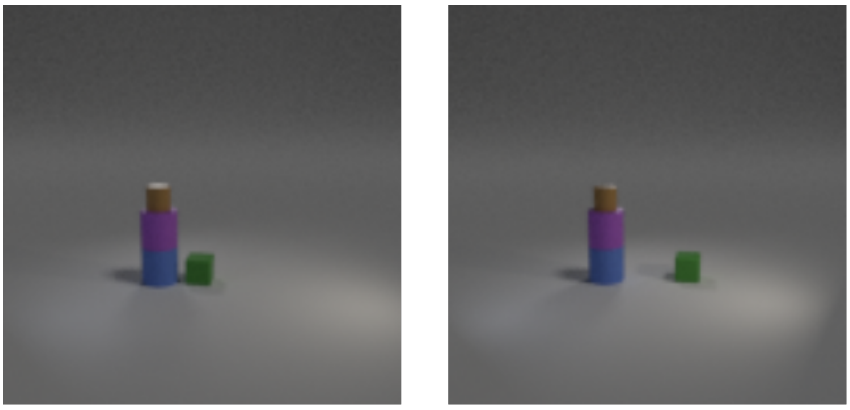

Figure 8: Two different states share the same scene graph representation ({brown, up, pink}, {pink, up, blue}, {blue, left, green}). New scene graph: *left*: ({brown, up, pink}, {pink, up, blue}, {blue, up, stack0}, {green, up, stack1}), *right*: ({brown, up, pink}, {pink, up, blue}, {blue, up, stack0}, {green, up, stack2}). Here *stack0*, *stack1*, and *stack2* are three additional imaginary bases.

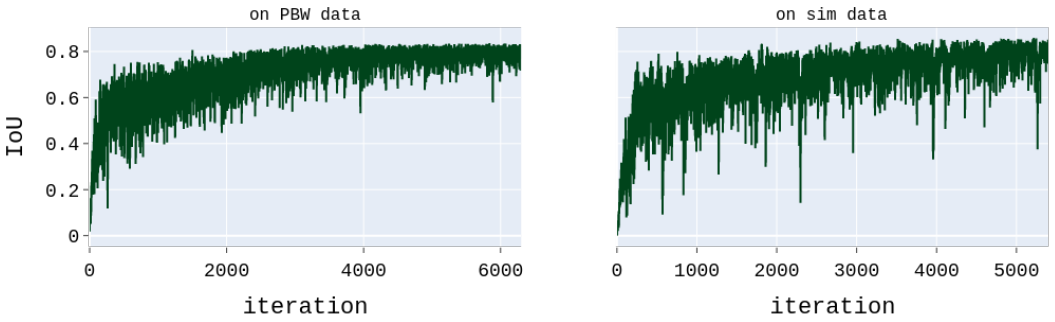

Figure 9: IoU on the (5 objects, 3 stacks) training sets of $\mathcal{D}_{\text{pbw}}$ (*left*), and $\mathcal{D}_{\text{sim}}$ (*right*).

A.4 Oracular $\mathcal{M}$

We explain here why our symbolic action model $\mathcal{M}$ is oracular. A general action model synthesizes the new state $\tilde{z}_1$ after applying an action a_0 to a state z_0, i.e., $\tilde{z}_1 = \mathcal{M}(z_0, a_0)$. Depending on the representation of z, finding this mapping can be challenging, especially when z are in the image space [42]. The synthesized $\tilde{z}_1$ can be visually correct, but will not be exactly the same as the true z_1. We avoid this problem by utilizing a symbolic scene graph to represent z. Given an action $a_0 = (p, q)$, our rule-based $\mathcal{M}$ works as follows: (1) find the top object o_1 of stack p, (2) find the top object o_2 of stack q, (3) remove any relations which o_1 holds from z_0, (4) add the relation $\{o_1, up, o_2\}$ to z_0, (5) output the newly modified z_0 as $\tilde{z}_1$. Hence, if the scene graph representations of the initial and goal scenes are correctly predicted, such an oracular action model always finds a feasible plan.

A.5 Additional Analysis on IoU Metric

We discussed in section 6 of the main text that the Intersection-over-Union (IoU) can be an indicator to measure the confidence of the model for a novel scene. The confidence score can be computed using $f(Z_{j_1})$, where f is either $\min$ or mean function, $Z_{j_1} = \{\text{IoU}(u_{j_2}, \tilde{u}_{j_2}) | 1 \leq j_2 \leq J\}$ with J is the number of scene objects and $1 \leq j_1 \leq |\mathcal{D}_{\text{sim-6obj}}|$. We further analyze to see which f is more suitable. Denote A as the set of **correctly** predicted scene graph $A = \{f(Z_{j_1}) | 1 \leq j_1 \leq |\mathcal{D}_{\text{sim-6obj}}|\}$, and B as the set of **incorrectly** predicted scene graph $B = \{f(Z_{j_1}) | 1 \leq j_1 \leq |\mathcal{D}_{\text{sim-6obj}}|\}$. The probability of indicator function f rejecting a correctly predicted scene graph is then $\frac{|A > \max(B)|}{|A|}$. This probability is 0.047 for $f_{\min}$ and 0.266 for f_{mean}, therefore suggests that $f_{\min}$ is a better indicator function due to the lower number of potential false positives. Fig. 10 also reflects this as we see a bigger overlapping area between the sets of correct and incorrect predictions (right column).

A.6 Image Masks and Planning with Goal Images

We present additional figures to further support the main text. Fig. 11 illustrates different mask images u_i and default matrices u_i^{def}. These samples are either rendered by [38] (top row) or our simulator (bottom row). Fig. 12 shows three additional plans which are different from those presented in the main text. In these samples, we specify the goals as scene *images* instead of relations.

B Extensions Towards Applicability in Real-World Settings

B.1 Dealing with random stack locations

The visual module $\mathcal{G}$ does not generalize well to novel locations of stacks. We randomly added a distance ϵ ($0 \leq \epsilon \leq \epsilon_{\text{max}}$) between stacks to augment the stack locations in the testing dataset $\mathcal{D}_{\text{pbw}}$ of *6-object, 3-stack*. We then chose randomly 500 test samples of this testing dataset and test the $\mathcal{G}$ which was trained on *5-object, 3-stack*. The IoU metric dropped from 0.74 to 0.651, 0.374, and 0.26 for various values of ϵ_{max} (Tab. 3). This issue can be fixed by first detecting the

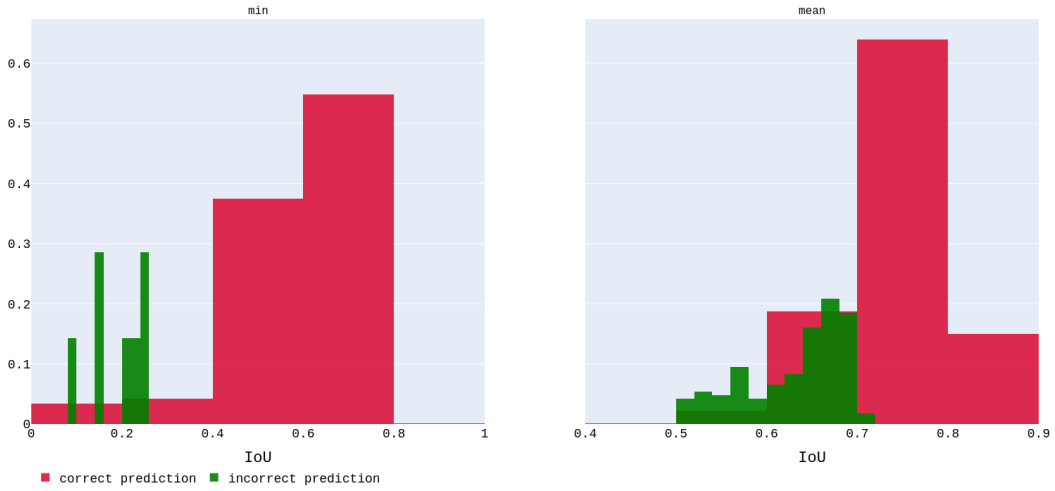

Figure 10: Histogram of occurrences for different IoU values in the set of incorrectly and correctly predicted scene graph.

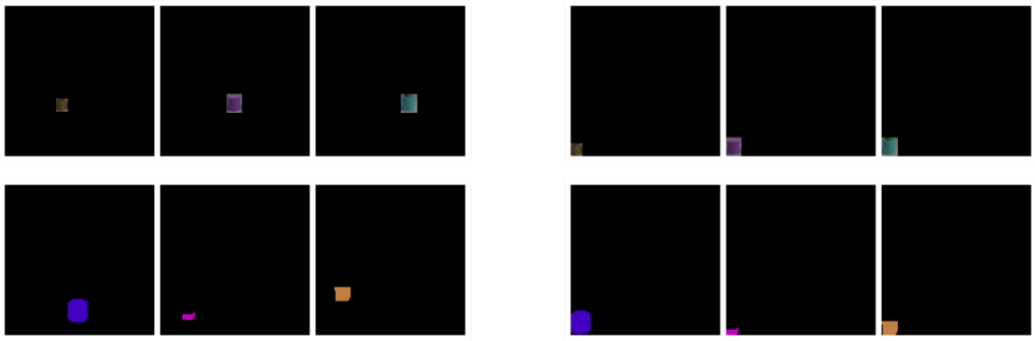

Figure 11: *Left column:* mask images u_i. *Right column:* default matrices u_i^{def}. *Top row:* samples from $\mathcal{D}_{\text{pbw}}$. *Bottom row:* samples from $\mathcal{D}_{\text{sim}}$.

Table 3: Performance comparison when randomly translating stacks horizontally.

ϵ_{max}	**Original**		**Improved**	
	IoU	**SG accuracy**	**IoU**	**SG accuracy**
0.5	0.651	**0.912**	**0.726**	0.892
1.0	0.374	0.892	**0.724**	**0.902**
2.0	0.26	0.784	**0.701**	**0.902**

stack regions by a Connected-component labeling algorithm[2] [43]. This algorithm starts at a non-zero pixel and start to expand to neighboring pixels (if they are also non-zero). The final sets of these non-zero pixels are called regions. These regions are then translated horizontally by d (where $d \in \{0, -5, 5, -10, 10, -15, 15\}$) to check if the new location (which is translated by d from the original) yields a better prediction (which is quantified by the IoU confidence). Tab. 3 shows that this technique improves the prediction significantly, hence successfully enhances the ability of $\mathcal{G}$ to novel locations (see Fig. 13 for a further illustration). Also note that the detected regions are exactly the stacks, therefore the number of stacks m is discovered as well.

[2]We use a modified version of a Python implementation, which is archived at https://github.com/jacklj/ccl.

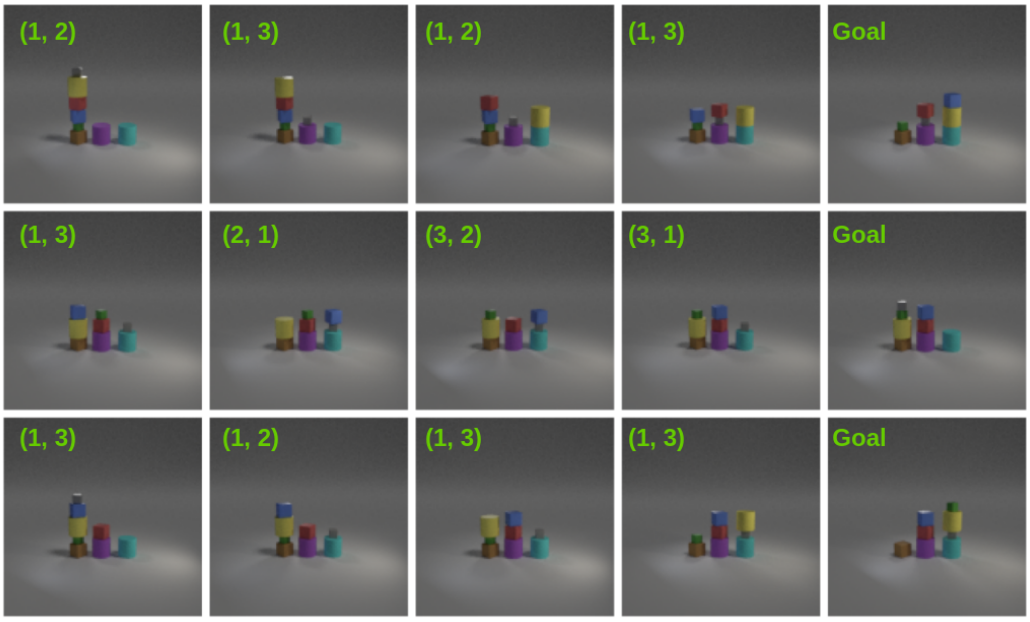

Figure 12: Three sample plans illustrated in each row. Initial images u_0 are the left-most, desired scenes u_{goal} are the right-most images of each row, respectively. Action (*green*), which transitions the current scene to the next, is located at the top-left corner of each image. Action notation follows sections 4.2 and 5.1.

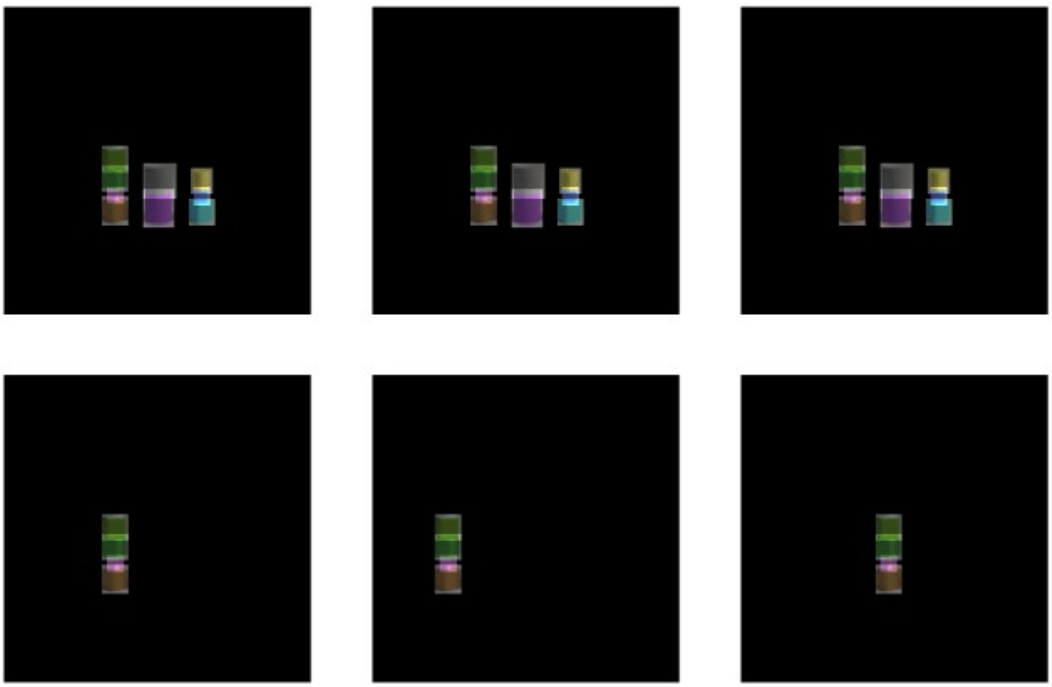

Figure 13: Illustrations of how to translate stacks in order to generalize better to novel locations. *Top row* illustrates the original locations while *bottom row* illustrates the translated locations of the left-most stack. From *left corner* to *right corner*, we translate the stack by 0, -15 and 15 pixels.

B.2 Prediction of object properties

Scene graph representation can be complemented if the properties of the objects are informed. For example, one of these properties is *shape*. Training a shape predictor, which infers the shape from the input bounding box, is possible. We generate an additional dataset $\mathcal{D}_{\text{pbw-shape}}$ of *4-object, 3-stack* to train such a predictor and another similar test dataset with different object colors. The shape predictor reaches an accuracy of 97% on the test set, which is no surprise since neural networks are known to excel at this task. Using this shape predictor, we can put heavy penalties on outputting actions involving fragile, unstable shapes using a heuristics search algorithm [44].

B.3 Possibility to include 3-D relations

Spatial 2-D relationships, such as *left, right, up*, and *down*, do not entirely capture the environment. For example, if there are 2 lines of stacks, it will be difficult to plan using only 2-D relationships. Therefore, 3-D relationships, such as: *in front* or *behind*, have a large impact on the usability of our method.

We propose an extension to capture the depth relationships (*in front, behind*) using only the depth information (e.g., from a depth sensor). We first predict the scene graph considering all the scene objects which are present in the current view. Due to possible occlusion, we implement a *pre-checking* subroutine. During this subroutine, the robot first moves all the visible objects into both the right-most and left-most stacks, thus is able to observe the occluded objects and predict the relationships for these objects. To evaluate this technique, we generated a dataset $\mathcal{D}_{\text{pbw-3D}}$ of 4 objects and 6 possible stack locations (three stacks are in front of the other three) and tested using the model which was trained on a dataset of 5 objects and 3 stacks (same as in Sec. 5.2). The test dataset $\mathcal{D}_{\text{pbw-3D}}$ has 2309 scene graphs which includes the depth relationships, 86.67% of which our model predicted correctly. Therefore, this experiment confirms the possible generalization of our model to 3-D relationships using an additional depth sensor.