Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

# Representation learning of scene images for task and motion planning

Son Tung Nguyen

| | |
|---|---|
| **Course of Study:** | Informatik |
| **Examiner:** | Prof. Dr. Jim Mainprice |
| **Supervisor:** | Dr. Özgür S. Ögüz<br>Prof. Dr. Marc Toussaint |
| **Commenced:** | March 2, 2020 |
| **Completed:** | October 29, 2020 |

## Abstract

This thesis investigates two different methods to learn a state representation from only image observations for task and motion planning (TAMP) problems. Our first method integrates a multimodal learning formulation to optimize an autoencoder not only on a regular image reconstruction but also jointly on a natural language processing (NLP) task. Therefore, a *discrete*, *spatially meaningful* latent representation is obtained that enables effective autonomous planning for sequential decision-making problems only using visual sensory data. We integrate our method into a full planning framework and verify its feasibility on the classic blocks world domain [26]. Our experiments show that using auxiliary linguistic data leads to better representations, thus improves planning capability. However, since the representation is not interpretable, learning an accurate action model is extremely challenging, rendering the method still inapplicable to TAMP problems. Therefore, to address the necessity of learning an *explainable* representation, we present a self-supervised learning method to learn *scene graphs* that represent objects ("red box") and their spatial relationships ("yellow cylinder on red box"). Such a scene graph representation provides spatial relations in the form of symbolic logical predicates, thus eliminates the need of pre-defining these symbolic rules. Finally, we unify the proposed representation with a non-linear 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.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# 1 Introduction

Sequential manipulation planning is one of the core challenges of robotics, which involves reasoning both on the *task* level and the *motion* level. The task level requires solving for high-level, discrete sequences of actions. Considering a task *grab a beer*, one action sequence can be *open the fridge*, *get the beer*, *close the fridge*, *and place the beer on the table*. On the motion level, continuous configurations has to be figured out to execute each of these high-level actions. Solving for such an action sequence requires joint optimization over all the constraints of each action. This is necessary since one motion plan of an action greatly affects the feasibility of the subsequent actions. However, due to the fact that each of the above actions imposes different constraints, it is too complex to solve this joint optimization problem analytically. To tackle the hybrid nature of such problems, task and motion planning (TAMP) methods requires pre-defined *symbolic* state representations and *rule-based* action model to be given while using either sampling-based [23, 29, 43, 49, 65] or optimization-based [67, 68] motion planners for solving the continuous motion trajectory. Realizing the predicates for the spatial relationships and the action transition rules that determine the logical and geometrical constraints are crucial to enhance the autonomy of the robotic agent that relies on existing TAMP methods. Therefore, acquiring *planning-compatible, relational* representations and transition rules from image observations is exactly the focus of this thesis work.

Representation learning is about "learning representations of the data that make it easier to extract useful information when building classifiers or other predictors" [7]. In other words, a representation would be of no use if it were not applied to any learning task. Moreover, it is not yet possible to learn a task-agnostic representation, which works well for all the subsequent learning tasks. Therefore, many of the works in the field have been focusing on learning continuous representations [9, 33, 42, 70] due to the straightforward gradient feedback of subsequent learning objectives. Meanwhile, learning discrete representations is not as common, despite the potential application to multiple interesting problems. One of these problems, in particular, is planning. Although it is still possible to plan on continuous state representation [34], the resulting plan is suboptimal and does not guarantee to reach the desired state due to the impossibility to perform *exact state comparison*.

Earlier studies [3, 4] show promising results of learning discrete, *planning-feasible* representation. Both of these works proposed to use a Gumbel-Softmax (GS) [37, 52] layer in a Variational Autoencoder [42] architecture to learn a discrete latent vector. However, the technique was evaluated on simple toy-like puzzles, in which low-dimensional image observations ($32 \times 32$) were sufficient. Additionally, van den Oord et al. [54] addessed that GS layer works only with low-dimensional images. We first propose to use Vector-Quantized Variational Autoencoder (VQ-VAE) [54] to encode higher-dimensional image observations, which is more realistic in robotic settings. However, both VQ-VAE and GS-activated encoder suffers from providing unstable encoding. This problem results from the fact that even though both produces discrete latents, they still rely on a continuous encoder. Hence, the latent alters when there is a small change in the observation (e.g., appearance of irrelevant objects or illumination). This is critical to planning performance since we would want an invariant representation for most, if not all, of the similar scenes. Inspired by Asai and Kajino [5], we investigate a method to enhance stability by jointly training the VQ-VAE with a visual-question answering framework in a pre-training step. This setting enables encoding image features which are important only for question-answering (QA). In particular, when the questions are about spatial relations, like those in the CLEVR dataset [38], the visual reasoning ability of the state encoder is supported. Since the spatial relations often characterize the scene semantics, understanding of these relationships results in a higher quality of representation. The statement is confirmed by investigating how efficiently a *stochastic* action model learns transition rules in the classic blocks world problem [2, 26]. Using the new technique, such an action model is able to gain approximately 8.2% in accuracy in the task of learning transition rules.

Despite this improvement, VQ-VAE still does not provide sufficiently invariant latent representation. The core issue perhaps lies within the explainability of such a representation. Even though the presented encoder [3, 4, 54] produces discrete vectors, these vectors contain just *unintelligible* scalars. We would not know which relationships between which scene objects each of these scalars denote. This unexplainability renders constructing a *rule-based, oracular* action model infeasible, forcing previous works [3, 4] to learn *stochastic* action models. These stochastic models are more complex to learn in TAMP problem settings. Even if such a model exists, there would still be uncertainties in inferring state transitions, hence misleading the search. A *symbolic, structured, hence human-readable* representation would be more appreciated. We then propose a visual module to first capture the representations of the input scenes in the form of *scene graphs* [39]. This is possible via an encoder network that transforms visual input into a local (relative to the view) coordinate system. Inside this coordinate system, $k$-means algorithm is used 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 scene graph, e.g., a triple $[\text{red}, \text{up}, \text{green}]$ denotes *red box is on top of green box*. Since these scene graph representations are symbolic, the action model is well-defined, enabling the use of any generic graph exploration algorithms to search for the optimal path to reach a goal state from an initial state.

To sum up, two methods have been developed to learn a state representation from image observations. The first method involves training a VQ-VAE as an image encoder via a QA framework. This allows us to learn a state representation that is more efficient, more stable under noisy settings, and with which it is easier to learn transition rules. However, since these representations are not interpretable, a *strong* stochastic action model is required, thus rendering planning impractical. We then propose a second method to learn a scene graph representation. Such a scene graph contains information about spatial relationships and provides a symbolic state representation. This characteristic enables the use of an *oracular* action model and eliminate the need of a *stochastic* one. Therefore, this technique integrates effectively into an existing motion planner [68] to solve for TAMP problems. The main contributions of this thesis are:

- We implement VQ-VAE to obtain discrete, planning-feasible representations of *large* images,
- We integrate an auxiliary QA loss during a pre-training step to restrict the latent space, and further improve visual reasoning capability of the framework.
- We implement a visual module that learns relative coordinates in a self-supervised way,
- We exploit this visual module to map a scene image into a scene graph,
- We present a unified framework comprising *learning* and *task and motion planning* directly from visual data to solve sequential robotic manipulation tasks effectively.

# 2 Related Work

**Learning manipulation skills** Reinforcement learning (RL) have been the focus of many works in learning motor skills for robotic manipulation. RL-based methods, in general, require a supervision signal, which comes from either a reward function [12, 19, 21] or a demonstration [18, 55, 57, 58]. These supervision signals, however, demand large human effort to design and annotate. Lynch et al. [51] proposed to learn from *unannotated, teleoperated, simulation-based* data in a self-supervised manner on triples [current state, logged teleoperated action, next state]. The resulting policy is trained to maximize the log-likelihood of the logged action. In another work with the same motivation, Nair et al. [53] present a technique to continuously sample and achieve a goal state by a simple distance reward function (in the latent space), thus facilitating the self-improvement of its general-purpose skills without a hand-designed reward. While RL-based methods require a flexible input modality (e.g., image pixels) and show impressive learned policy for motor skills, they do not maintain an action model which is critical to understand transition rules. This issue prevents these methods to perform offline planning for longer horizon without modifying the environment.

**Symbolic planning** This thesis work is heavily inspired by the use of symbolic planning [3, 17, 28, 34] for manipulation. Symbolic planning often utilizes an *efficient, optimal* graph search algorithm, hence allows us to look into further planning horizon without acting in the environment. However, symbolic planners do not work out of the box with continuous state representations. Huang et al. [34] overcame this challenge by relaxing the discrete requirement of symbolic planners. This method applies only the actions with the highest chance of satisfying preconditions. Since these actions are not necessarily the optimal actions, the resulting plan is suboptimal. This plan also does not guarantee to reach the goal state since we can only quantify the *reaching condition* by a distance measure. Therefore, the method is applicable to the tasks in which *exact* goal achievement is not critical, such as manipulation of deformable objects. On the other hand, different works [3, 4] attempted to obtain discrete state representations to be compatible with existing classical planning by adding a Gumbel-softmax [37, 52] layer to enforce discrete outputs. While being planning-compatible, these representations are not ideal to learn a stochastic action model to be effective in real-world robotic settings. This difficulty of learning an action model misguides the search, thus renders the plan suboptimal.

**Task and motion planning** Unlike RL-based counterparts which learn pose configurations from image pixels, task and motion planning (TAMP) methods solve for such motion trajectories of configurations by numerical optimization algorithms. This optimization is particularly challenging when we have to optimize jointly for a sequence of action, e.g., *pick a the red box and place it on top of the blue box*, since each of these actions impose different geometric constraints. Toussaint [67] first proposes a method known as *logic-geometric programming* (LGP) to formulate this optimization problem into a non-linear program (NLP) and solve locally using constrained optimization methods [66]. This approach optimizes directly on the geometric constraints, thus is different from earlier TAMP's methods [46, 49, 65], in which geometric constraints are introduced as symbolic abstractions. LGP, however suffers from the inefficiency due to the large search space of possible actions. Several subsequent studies successfully enhanced the scalability of LGP either by branch-and-bound heuristic search [69] or using neural networks for predicting action sequences, thus reducing the number of NLPs to be solved [14, 15].

**Inductive priors for representation learning** Annotated data is often expensive and not always available. One of the most commonly proposed solutions is unsupervised learning of a pre-trained representation. The main idea is trying to make a good use of largely available unannotated datasets (images [30], natural languages [1, 13], offline exploratory data [71]), obtaining inductive priors which empower pre-trained models. Inspired by Andreas et al. [1], we present a technique which complements discrete representation learning with linguistic data. The training phase is divided into two phases in [1]. In *pre-trained phase* both a sampler of missing language data in the downstream task and a shared parameter space are learned, and in *concept-learning phase* the parameter space is fine-tuned on the downstream task. During the concept-learning phase, the sampler is used to sample any absent parameter in the downstream dataset. We propose to merge pre-trained and concept-learning phases into a single step, which results in a unified framework and naturally solves the problem of absent parameters in the downstream task.

# 3 Background

## 3.1 Deep learning

Deep learning is a branch of machine learning which deals with the question: how to learn from experience. While machine learning field has developed a number of learning algorithms, deep learning concerns only neural network architecture. Because of their arbitrary complexity, neural networks have large representation capability, thus have been applied widely and successfully in many tasks: machine translation, speech processing, and computer vision. Throughout its history, deep learning architectures have evolved into many different forms to be suitable for a specific task, hence rendering impractical to describe the best architecture for all the tasks. Therefore, we describe only the most basic blocks which are used in this thesis work.

### 3.1.1 Convolutional neural network architecture

**Convolutional layer:** Convolutional layer is the core component of CNNs where most of the computation takes place. Convolutional layer has four hyper-parameters: **F** - the field size, **K** - the number of filters, **S** - the stride, **P** - the amount of zero padding. First, the input matrix is padded with zeros according to the parameter **P**. Next, a window kernel is convolved within the input matrix to compute the dot product between the kernel elements and the input matrix elements. This process results in **K** two-dimensional feature maps. Note that the number of trainable weights can be reduced using *parameter sharing* scheme. This scheme constrains all the kernels within the same depth to use a single set of weights and bias. Hence, the number of weights becomes $\mathbf{K} \cdot \mathbf{F} \cdot \mathbf{F} \cdot \mathbf{D}$ which is reduced by a factor of $\mathbf{W}_2 \cdot \mathbf{H}_2$ (where **D** - the depth of the input matrix, $\mathbf{W}_2$ and $\mathbf{H}_2$ - the width and height of the resulting feature map).

**Pooling layer:** Pooling layer is used to shrink down the dimension of the input matrix, thus controlling the size of the whole network and helping to combat overfitting. The most common operation of the pooling layer is the **MAX** operation. Similar to the convolutional layer, the pooling layer employs a kernel window of size **F**. This kernel is convolved across the input matrix with **S** - stride, but output the largest element instead of the dot product. Note that the **MAX** operation is not differentiable, therefore only directs the gradient of the largest element during backpropagation.

**Fully-connected layer:** The desired output of an forward pass through a neural network is often a single vector. Depending on the learn task, this vector denotes different meanings. However, the output of the convolutional and pooling layer is a matrix. Therefore, it is common to flatten this matrix into a one-dimensional vector first. This vector is then multiplied with a weight vector of the fully-connected layer to produce an output vector of desired size.

### 3.1.2 Spatial transformer network

Spatial transformer network [36] is a special neural network architecture which learns spatial transformation. Spatial transformer network consists of two main components: a localisation network $f_{\text{loc}}$ and a grid sampler. The localisation network outputs the affine vector $\theta = f_{\text{loc}}(U)$, where $U$ is the input feature map ($U \in \mathbb{R}^{H \times W \times C}$). $f_{\text{loc}}$ can be any neural network which includes a regression final layer. After $\theta$ is determined by $f_{\text{loc}}$, a grid $G$ is defined in the the input map $U$ so that each element lies in a corresponding grid point. Next, this grid $G$ is transformed using the affine parameter $\theta$ to obtain $\mathcal{T}_\theta(G)$. The grid sampler then uses $\mathcal{T}_\theta(G)$ and $U$ to produce a transformed output matrix $V$ using bi-linear sampling:

$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|) \tag{3.1}$$

where $U_{nm}^c$ is the value at location $[n, m]$ in channel $c$ of the input, and $V_i^c$ is the output value for pixel $i$ at location $[x_i^t, y_i^t]$ in channel $c$. Note that $[x_i^t, y_i^t]$ corresponds to $[x_i^s, y_i^s]$ by the affine transformation $\mathcal{T}_\theta$:

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \mathcal{T}_\theta \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} \tag{3.2}$$

Also note that Equation (3.1) allows us to define $\frac{\partial V_i^c}{\partial U_{nm}^c}$, $\frac{\partial V_i^c}{\partial x_i^s}$, and $\frac{\partial V_i^c}{\partial y_i^s}$, therefore enabling gradient backpropagation to the input feature map $U$ and $f_{\text{loc}}$ since $\frac{\partial x_i^s}{\partial \theta}$ and $\frac{\partial y_i^s}{\partial \theta}$ are known using Equation (3.1). The interested readers can refer to [36] for a complete derivation of these partial derivatives.

### 3.1.3 Variational auto encoder

Variational auto-encoder (VAE) [42] is a neural network architecture which focuses on learning representation using reconstruction loss. VAEs consist of three components: a posterior distribution $q(z|x)$ which is parameterized by an neural network called the encoder, a prior distribution $p(z)$ where $z$ is mapped to, and a decoder distribution $p(x|z)$ which is also parameterized by an neural network. While there are many works which focus on continuous representation [9, 33, 70], discrete representation [37, 52, 54] is fewer but useful in many tasks, such as planning or reasoning. Among these works of discrete representation learning, Vector Quantized - Variational AutoEncoder (VQ-VAE) [54] tends to work better when experimenting with large datasets, e.g., ImageNet [62] or CIFAR-10 [45]. VQ-VAE maps a latent representation $z$ which is an ordered set of $L$ differentiable, pre-defined, continuous embedding vectors $e_i$: $z = \{e_i \mid e_i \in \mathbb{R}^{K \times D}, 1 \leq i \leq K\}$, $|z| = L$, where $e_i$ are $D$-dimensional selected from the embedding space of size $K$ and $L < K$. Observe that $z$ can be uniquely represented by indices of embedding vectors $e_i$ in the embedding space, therefore facilitating discrete trait while keeping the stochastic nature as $e_i$ are continuous.

### 3.1.4 Optimization

**Objective functions** Objective functions are used to measure how good or bad the prediction currently is. Based on these feedbacks, the optimizer will update the networks' weights so that better predictions will be made next time. Depending on the learning task, the objective function should be chosen accordingly. Here, we describe three different objective functions, which are used in this thesis. Denoting $N$ - the number of training samples, $(x_i, y_i)$ - the $i$-th training sample, $x$ - the input, $y$ - the output, and $f$ - the learned function (e.g.,. a neural network):

- Mean-squared loss $\mathcal{L}_{\text{mse}} = \Sigma_{i=1}^{N}(f(x_i) - y_i)^2$, which is usually employed in reconstruction tasks, e.g.,. generative modelling.
- Cross-entropy loss $\mathcal{L}_{\text{cent}} = -\log(\frac{e^{f_{y_i}}}{\Sigma_j e^{f_j}})$, which is useful to produce probabilities. The cross-entropy loss minimizes the negative log-likelihood of the correct class. This is equivalent to minimizing the Kullback-Leibler divergence between the predicted distribution $f(x_i) = [f_1, f_2, ..., f_j, ..., f_M]$ and the correct distribution $p = [0, ..., 1, ..., 0]$ (the number 1 appears at the $y_i$ location).
- Hinge loss $\mathcal{L}_{\text{hinge}} = \max(0, \gamma - d(f(x_i), y_i))$, which is useful to perform contrastive learning. The hinge loss constrains the gradient backpropagation only when $d(f(x_i), y_i))$ is smaller than $\gamma$. Otherwise, no learning is performed. This is convenient when the distance $d$ between $f(x_i)$ and $y_i$ is desired to be at least $\gamma$.

**Backpropagation** Backpropagation [61] is the powerful method of distributing gradient from the output back to the inputs. During optimization, we know how the output $f$ needs to change (using one of these objective functions above), but not how the inputs need to change to obtain the necessary change in the output. The *chain rule* tells us exactly how to. For example: $f = (x + y)z = qz$, therefore $\frac{\partial f}{\partial q} = z$ and $\frac{\partial f}{\partial z} = q$. Because we also know $\frac{\partial q}{\partial x} = \frac{\partial q}{\partial y} = 1$, we apply the chain rule to obtain $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = z$ and $\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y} = z$. At this point, it is clear how the inputs $[x, y, z]$ has to change with respect to the output $f$. Abstracting $q = x + y$ is known as *staged backpropagation* in practice. This abstraction allows us to break down the complex $f$ into simpler functions whose known local gradient rules. These local gradients are then chained together using the chain rule to obtain the global gradient, without the explicit derivation of the partial gradients of function $f$.

**Optimizer algorithms** After backpropagation is performed, we know how the inputs should change to obtain the desired change in the output. One simple rule is just updating our parameters like: $w_{t+1} = w_t - \alpha\Sigma_{i=1}^{N}\frac{\partial\mathcal{L}(f(w_t, x_i), y_i)}{\partial w_t}$ where $\mathcal{L}$ - the objective function, $\alpha$ - the learning rate, $N$ - the number of training samples, $w_t$ - the current parameter at time $t$, $f$ - the function which maps $x_i$ to the prediction using parameter $w$, and $[x_i, y_i]$ - the $i$-th training sample. This rule is commonly known as *Vanilla Gradient Descent*, in which the gradient of the whole training set is computed to update the parameter $w_t$. However, $N$ can be very large, e.g., 14 million samples in the ImageNet dataset [62], therefore it is wasteful and memory intensive to compute the gradient for the whole dataset. The solution for large datasets is to compute gradient for mini-batches of the dataset, update the parameter using this gradient, and iterate until the whole dataset has been learned. Another common optimzation algorithm is Adam [41] which computes adaptive learning rates of each of the parameters by storing past gradients and squared gradients, hence is more memory intensive. However, Adam often works well in practice and is the choice for all the experiments in this thesis.

## 3.2 Scene graph

As Johnson et al. [39] wrote, “a scene graph is a data structure that describes the contents of a scene”. Each edge of this graph structure denotes the relationship between the two nodes, e.g., *a man is holding a beer* will have the edge of *holding* between the node *a man* and the node *a beer*. Johnson et al. [39] formally formulated a scene graph as $G = (O, E)$ where $O = \{o_1, \ldots, o_n\}$ is a set of objects, $E \subseteq O \times R \times O$ is a set of edges, and $R$ is a set of relationships. In the context of this thesis, we only focus on spatial relationships, thus $R = \{\textit{left, up}\}$ with a possible extension to include the *front* relation to be more effective in the 3-D world. This is a important highlight since it is much more challenging to form a self-supervised learning formulation of a more *general, semantic* relations like those in [39]. When used as state representations, scene graphs offer multiple appealing properties: symbolic, explainable, and invariant if done right. These properties enables scene graphs to be applicable to planning problems.

## 3.3 Classical planning

Planning is one of the core abilities of biological intelligence. Planning concerns with the realization of action sequences which must be taken to achieve a desired state of the environment. Classical planning algorithms have been developed since the early age of A.I. These algorithms often include three components: the input state $x_0$, the output state $x_{\text{goal}}$, and the action model $\mathcal{M}$. State representation $x_i$ is structured and symbolic, thus allows for goal state verification. Goal state verification serves as a stopping condition for the search, and is not trivial if $x_i$ is continuous or complex, high-dimensional (e.g., image observations). This requirement renders classical planning algorithms impractical in real-world settings. It is the core of this thesis that trying to learn structured, discrete representation of image observations, hence bridging classical planning to robotic manipulation. The action model $\mathcal{M}$ outputs a consequent state after applying an action $a_i$ to a state $x_i$: $\mathcal{M}(x_i, a_i) = x_{i+1}$. Depending on the state representation and the planning problem, $\mathcal{M}$ can be *oracular* (deterministic, rule-based) or *stochastic* (unknown dynamic environment). After these three components are defined, we can use a generic graph search algorithm [20] to search for the optimal action sequences. We describe two of such algorithms, namely breadth-first search and A-star algorithm [27], which are sufficient within the experimental settings of this thesis (see Algorithms A.1 to A.2).

## 3.4 Task and motion planning

Task and motion planning (TAMP) is a planning problem in which we try to solve for task (what should I do first, second, third, and so on?) and motion (how do I move my arms to perform the task?) jointly. A task, such as "get one bottle of wine", requires multiple sub-tasks to be done on the *task level*, such as: go to the fridge, open the fridge, grab the wine bottle, close the fridge, and put the bottle on the table. On the *motion level*, each of these sub-tasks requires to solve for continuous parameter of the robot's joints. For example, in order to open the fridge, how much and how far the robotic arm should rotate and translate so that the finger tips align with the door handler of the fridge? This is a challenging problem since each action also influences the feasibility of subsequent actions, e.g., how the robot grasps the bottle influences how the robot puts the bottle on the table. Lozano-Pérez and Kaelbling [50] emphasizes this problem as "significantly nonlinear and much too complex to solve exactly analytically in continuous form". However, Toussaint [67] proposed to formulate the problem into a mathematical program and solve locally using constrained optimization tools. This method is referred as *Logic - Geometric Programming (LGP)* [67, 68, 69], on top of which this thesis work is built. We now describe three main components which are crucial to solve an LGP formulation.

**Non-linear programming formulation** LGP, in the most general form, formulates a TAMP problem as a non-linear program (NLP) [68]. 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 configuration $x_0$. Given a goal description $g$, we aim to find a sequence of actions $a^r_{1:K}$ and a path $x : [0, KT] \rightarrow \mathcal{X}$ that minimizes:

$$\begin{aligned}
\min_{\substack{x,K,\\ a^r_{1:K}, s_{1:K}}} \quad & \int_0^{KT} c(\overline{x}(t), s_{k(t)})\, \mathrm{d}t \\
\text{s.t.} \quad & x(0) = x_0, \qquad s_0 = \tilde{s}_0, \qquad h_{\text{goal}}(x(KT), g) = 0 \\
& \forall t \in [0, KT] : \quad h_{\text{path}}\big(\overline{x}(t), s_{k(t)}\big) = 0, \quad g_{\text{path}}\big(\overline{x}(t), s_{k(t)}\big) \leq 0 \\
& \forall k \in 1, \ldots, K : h_{\text{switch}}(\hat{x}(t), a^r_k) = 0, \quad g_{\text{switch}}(\hat{x}(t), a^r_k) \leq 0 \\
& \qquad a^r_k \in \mathcal{A}^r(s_{k-1}), \qquad s_k \in q(s_{k-1}, a^r_k),
\end{aligned} \tag{3.3}$$

where:

- $\overline{x} = (x, \dot{x}, \ddot{x})$ which denotes poses, pose velocity, and pose acceleration
- $\hat{x} = (x, \dot{x})$
- $h$ describes equality constraint, e.g., velocity of the object must stay constant
- $g$ describes inequality constraint, e.g., acceleration of robot's joints should not excess a limit
- $g_{\text{path}}$ and $h_{\text{path}}$ describes the (in)equality constraints for the motion path, e.g., holding object must have zero velocity relative to the end-effector
- $g_{\text{switch}}$ and $h_{\text{switch}}$ describes the (in)equality constraints of the transition between kinematic modes, e.g., when touching the object, the arm must have zero velocity
- $s_k$ are symbolic states which define the constraints at each phase, and determine the available actions from the set $\mathcal{A}^r$
- $k \in 1, \ldots, K$ denotes the $k$-th switch, e.g., switching from *holding the object* to *putting the object down*
- $t \in [0, KT]$ denotes the discretized time step for each trajectory plan

**Multi-bound tree search** Multi-Bound Tree Search (MBTS) algorithm [69] is a combination of branch-and-bound [48], heuristic search [27], and Monte - Carlo (MC) tree search [11]. In MBTS algorithm, geometric feasibility is taken into account, hence avoids expanding the nodes which are geometrically infeasible. This is a key improvement over MC search, which was used originally in the work of Toussaint [67]. The efficiency was characterized by Toussaint and Lopes [69] as "there could be around 20 possible decisions in the start state, while geometrically only about 4 of them are feasible". MBTS maintains a tree-like structure, in which the root node contains the start state in *symbolic* form. Each child node then contains the following information:

– the list of child nodes

– symbolic state which is originated from the root

– three NLPs, each of which corresponds to different coarser levels of the full LGP: $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ (detailed below)

– information about whether $\mathcal{P}_i$ has been given to the optimizer and a feasible solution has been found.

Note that solving the NLP for the full LGP is computationally expensive, but the only way to detect geometrical infeasibilities. However, Toussaint and Lopes [69] proves that “by dropping terms from the cost function, or constraint function terms from constraint functions, a simpler, lower-bound $\hat{\mathcal{P}}$ is obtained”. This coarser $\hat{\mathcal{P}}$ is computationally cheaper, and computing feasibility of $\hat{\mathcal{P}}$ allows for pruning the tree early in case of detected infeasibility. However, this is only an approximated heuristic due to the locality of the optimization. The finest level $\mathcal{P}_3$ is the full LGP and only solved when a symbolic terminal node is reached. On the next level $\mathcal{P}_2$ the NLPs of the all the keyframes, i.e. when a switch is required, is considered. $\mathcal{P}_2$ and $\mathcal{P}_3$ are passed to the optimizer only when a symbolic state is the desired goal state. The coarsest level $\mathcal{P}_1$ requires solving only the inverse kinematics problem of the last pose in the action sequence. $\mathcal{P}_1$ of the current node is solved when the parent node’s $\mathcal{P}_1$ has been solved.

***k*-order motion optimizer** Toussaint [66] describes the $k$-order motion optimizer (KOMO), which formulates NLPs generally as:

$$\begin{aligned} &\min_{x_{0:T}} \sum_{t=0}^{T} f_t(x_{t-k:t}) \\ \text{s.t.} \quad &\forall_t : \ g_t(x_{t-k:t}) \leq 0 \ , \ h_t(x_{t-k:t}) = 0 \ . \end{aligned} \tag{3.4}$$

where

– $x_t \in \mathbb{R}^n$ is a joint configuration and $x_{0:T} = (x_0, \ldots, x_T)$ is a trajectory of length $T$

– $x_{t-k:t} = (x_{t-k}, .., x_{t-1}, x_t)$ are $k + 1$ tuples of consecutive states

– the functions $f_t(x_{t-k:t}) \in \mathbb{R}^{d_t}$, $g_t(x_{t-k:t}) \in \mathbb{R}^{m_t}$, and $h_t(x_{t-k:t}) \in \mathbb{R}^{l_t}$ are arbitrary first-order differentiable non-linear $k$-order vector-valued functions

Inspired by Conn et al. [10], Toussaint [66] expanded Equation (3.4) to the augmented Lagrangian form:

$$\hat{L}(x) = f(x) + \sum_j \kappa_j h_j(x) + \sum_i \lambda_i g_i(x) + \alpha \sum_j h_j(x)^2 + \beta \sum_i [g_i(x) > 0] g_i(x)^2 \tag{3.5}$$

The first iteration starts with $\kappa = \lambda = 0$, hence Equation (3.5) drops the second and third terms. The gradient of $x$ is computed by $\dot{x} = \min_x \hat{L}(x)$ using a Gauss-Newton method, then Lagrange parameters will be updated by

$$\kappa \leftarrow \kappa + 2\alpha h_j(\dot{x}) \ , \quad \lambda \leftarrow \max(0, \lambda + 2\beta g_i(\dot{x})) \tag{3.6}$$

# 4 From Images to Task Planning: How Natural Language Processing Can Help Physical Reasoning

This chapter[1] integrates a multi-modal learning formulation into classical AI planning. For sequential manipulation tasks, planning with visual reasoning is challenging for autonomous agents. We propose to optimize an autoencoder not only on a regular image reconstruction but also jointly on a natural language processing (NLP) task. In essence, a *discrete*, *spatially meaningful* latent representation is obtained that enables effective autonomous planning for sequential decision-making problems only using visual sensory data. We integrate our method into a full planning framework and verify its feasibility on the classic blocks world domain. Our experiments show that using auxiliary linguistic data leads to better representations, thus improves planning capability. Our main contributions of this chapter are:

- We implement Vector-Quantized Variational Autoencoder (VQ-VAE) to obtain discrete state representations of *large* scene images which are feasible for classical planning.
- We integrate an auxiliary question-answering loss during a pre-training step to restrict the latent space, and further improve visual reasoning capability of the framework.

## 4.1 Problem Statement

We focus on solving planning problems for sequential robotics manipulation tasks autonomously. Our objective is to achieve physical reasoning by only visual perception and without human-designed logical rules for high-level action selection. The goal is to find an action sequence $a_{1:N}$, given an image of the scene $x_0$, and a goal state description $v_{\text{goal}}$, e.g., *red box is above yellow box*,

$$\begin{aligned} &\operatorname{argmin}_{a_{1:N}} \sum_{i=1}^{N} \mathcal{H}(z_{i-1}, a_i) \\ &\text{s.t. } z_0 = \mathcal{G}(x_0), \quad z_N = \mathcal{G}(\mathcal{F}(v_{\text{goal}})), \\ &\qquad z_{i+1} = \mathcal{M}(z_i, a_i), \quad a_i \in \mathbf{A} \end{aligned} \tag{4.1}$$

where $\mathcal{H}$ defines the optimality criterion, $z_0$, $z_N$ is the initial and final latent states encoded by $\mathcal{G}$, $\mathcal{M}$ is an action model, $\mathcal{F}$ is a mapping from the text space to the image space, and $\mathbf{A}$ is a fixed set of actions (Figure 4.2). Note that this formulation is generalizable to whether the goal state is in the

[1]This chapter was discussed in the form of a presentation at the R:SS 2020 workshop on Self-Supervised Robot Learning

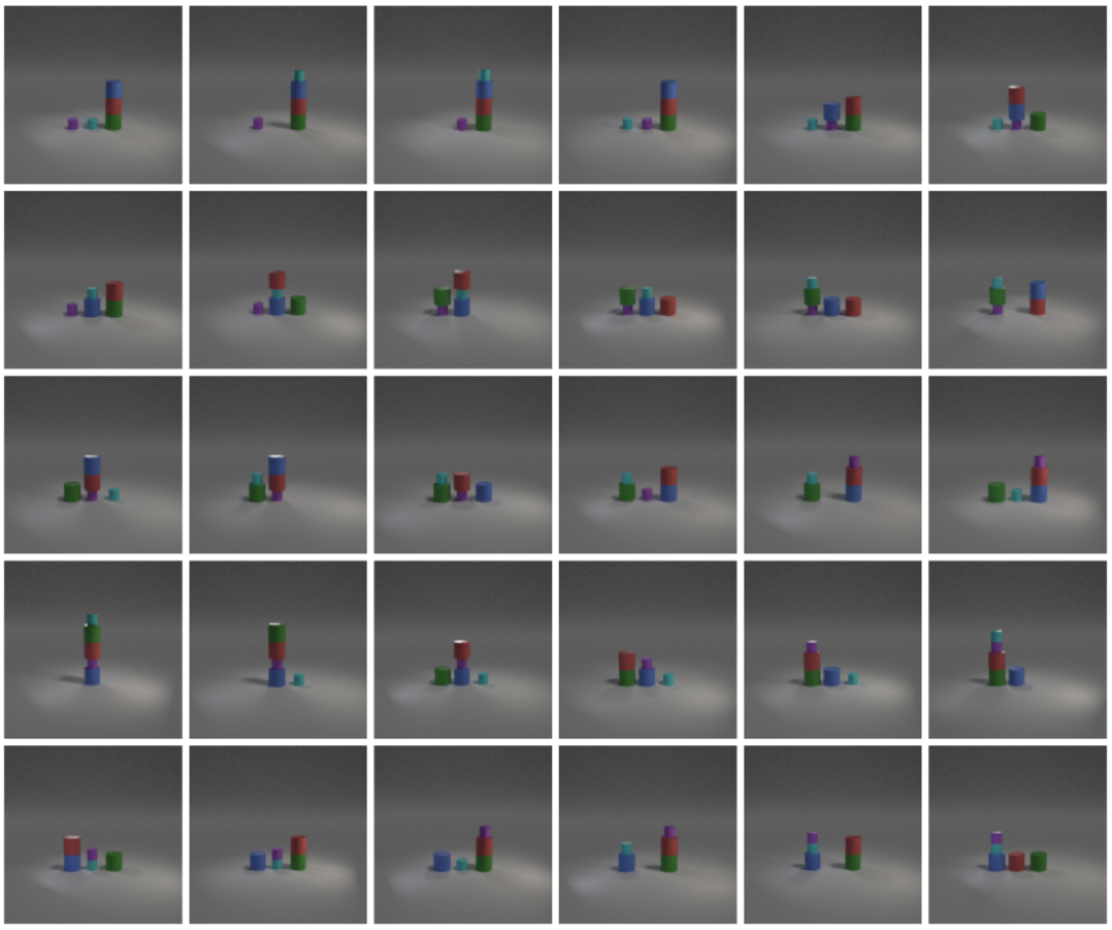

**Figure 4.1:** Five samples of planning sequences (from *top* row to *bottom* row) using $\mathcal{M}_1$ where the *left-most* and *right-most* images are specified as initial and goal states, respectively.

image or the text space. We aim for this generalization since textual (or verbal) goal state $v_{goal}$ is reasonable in real-world settings. We do not focus on deriving such $\mathcal{F}$ in this work, instead we relax this constraint and use $x_{\text{goal}}$, i.e., $\mathcal{G}(\mathcal{F}(v_{\text{goal}})) \simeq \mathcal{G}(x_{\text{goal}})$.

## 4.2 Methodology

First, we describe how to realize $\mathcal{G}$. Then, we describe two possible action models $\mathcal{M}_1$ and $\mathcal{M}_2$. These action models are used to integrate $\mathcal{G}$ into a planning framework to solve for Equation (4.1).

### 4.2.1 State encoder

To realize $\mathcal{G}$, we propose to use a Vector Quantized - Variational AutoEncoder (VQ-VAE) [54] to directly encode raw images into propositions under a multi-modal training scheme. The latent vector $z$ in VQ-VAE is an ordered set of $L$ differentiable, pre-defined, continuous embedding vectors

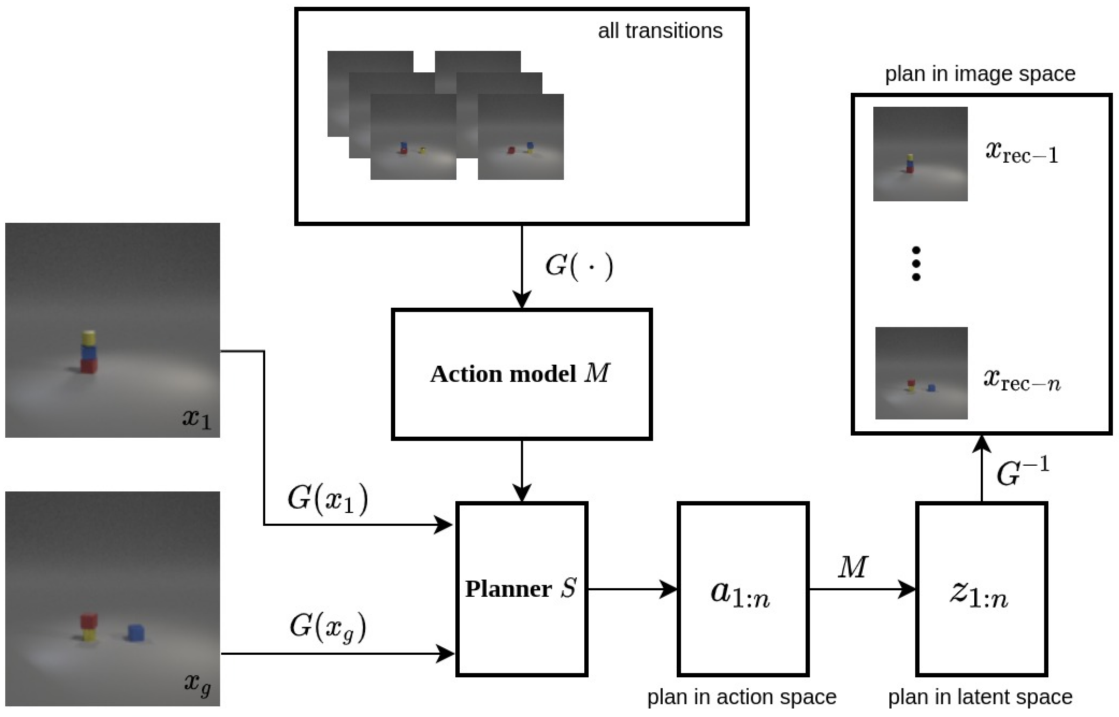

**Figure 4.2:** An overview of the high-level planning framework as formalized in Equation (4.1).

$e_i$, i.e., $z = \{e_i \mid e_i \in \mathbb{R}^{K \times D}, 1 \leq i \leq K\}$, $|z| = L$, where $e_i$ are $D$-dimensional selected from the embedding space of size $K$ and $L < K$. Observe that $z$ can be uniquely represented by indices of embedding vectors $e_i$ in the embedding space, therefore facilitating trivial goal state verification while keeping the stochastic nature as $e_i$ are continuous. This is crucial for our training settings where $z$ is jointly optimized with an auxiliary linguistic task.

In order to make use of linguistic data, we propose to jointly train $\mathcal{G}$ (which is a VQ-VAE) in a visual-question answering (VQA) framework (Figure 4.3). Given an image $x$, a question $w_{1:T}$ which is a sequence of $T$ words $w_i$, the VQA framework returns an answer by forming a distribution over a fixed set of answer $\mathbf{Y}$, i.e., $p(y_j|\cdot)$ where $1 \leq j \leq |\mathbf{Y}|$. Under this setting, our goal is to encode image features which are important not only for reconstruction but also for question-answering. Hence, when the questions are about spatial relations, like those in the CLEVR dataset [38], the autoencoder focuses more on the semantically essential scene elements.

To process the textual data, we use a relation network [63], $p(y_j|\cdot) = RN(x, w_{1:T}) = RN_x$ ($w_{1:T}$ omitted for brevity). Our full training objective becomes:

$$\mathcal{L}_{\mathcal{G}} = \alpha \mathcal{L}_{\text{rec}}(x_{\text{rec}}, x) + \beta RN_{x_{\text{rec}}} RN_x + \mathcal{L}_{\text{lh}} \tag{4.2}$$

where $\mathcal{L}_{\text{rec}}$ is the mean squared error loss, $D_{\text{KL}}$ is the Kullback–Leibler divergence loss, and $\mathcal{L}_{\text{lh}}$ is the log-likelihood loss originally presented in [54]. We keep the log-likelihood loss in all of our experiments.

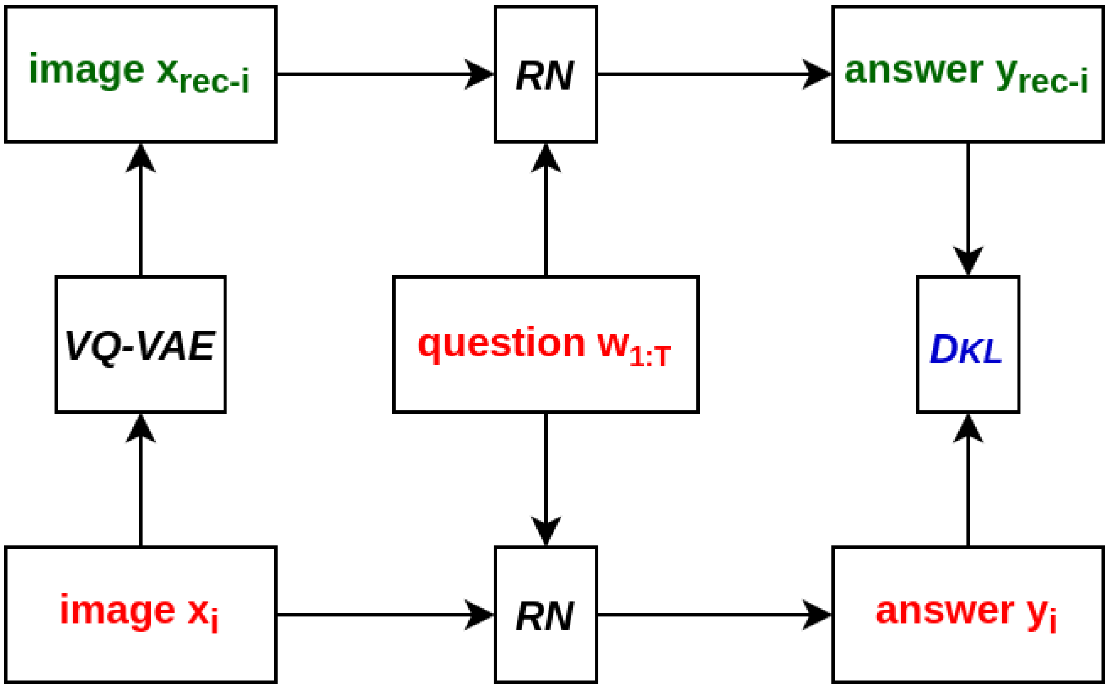

**Figure 4.3:** An overview of a forward pass through our VQA framework from inputs (depicted in red color) to outputs (depicted in green color).

### 4.2.2 Action model

In order to solve for Equation (4.1), an action model $\mathcal{M}$ is missing. We evaluate $\mathcal{G}$ with two action models.

#### Oracular action model

The oracular action model $\mathcal{M}_1$ observes all the possible state transitions in the state space, therefore $\tilde{z}_{i+1} = \mathcal{M}_1(z_i, a_i)$ is exactly equal to $z_{i+1}$ where $z_{i+1}$ and $\tilde{z}_{i+1}$ are the ground-truth and predicted resulting state of applying an action $a_i$ to a state $z_i$.

#### Stochastic action model

Our stochastic action model $\mathcal{M}_2$, which is inspired by Asai and Fukunaga [4], has two components: *action policy* $\pi$ and *action discriminator* $\mathcal{D}$.

- The $\pi$ network is trained on transition pairs $(\mathcal{G}(x_i), a_i, \mathcal{G}(x_{i+1}))$ where $a_i$ is the action which transitions the scene image $x_i$ to $x_{i+1}$. $\pi$ samples the result state after applying an action to one state, i.e., $\tilde{z}_{i+1} = \pi(z_i, a_i)$ and $z_i = \mathcal{G}(x_i)$. To train $\pi$, we use a contrastive learning objective function similar to [44]:

$$\mathcal{L}_\pi = d(\tilde{z}_{i+1}, z_{i+1}) + \max(0, \gamma - d(\tilde{z}_{i+1}, z_{i+1}^{\text{neg}})) \tag{4.3}$$

where $d$ is the squared difference, $\gamma$ is a hyper parameter ($\gamma = 1$ during our experiments [44]), and $z_{i+1}^{\text{neg}}$ are negative examples, sampled from the same training batch.

- The $\mathcal{D}$ network decides which action is applicable given a state, i.e., $\mathcal{D}(z_i) = \{u_j \mid 1 \leq j \leq |\mathbf{A}|\}, u_j \in \{0, 1\}\}$ where $u_j = 1$ if $a_j$ is applicable to state $z_i$. To train $\mathcal{D}$, we use binary cross-entropy (BCE) loss in a multi-label classification manner:

$$\mathcal{L}_\mathcal{D} = \text{BCE}(\mathcal{D}(z_i), \mathbf{U}) \tag{4.4}$$

where $\mathbf{U} = \{u_j\}$.

We found that the $\mathcal{D}$ component often performs much better than the $\pi$ component (apprx. 94% vs. 65% accuracy, respectively) on the test set. Since the accuracy of $\pi$ appears to be the bottleneck of $\mathcal{M}_2$, our analysis focuses on this component.

## 4.3 Evaluation

We evaluate our method based on three criterion: *reasoning*, *planning*, and *efficiency*. Our experiments are based on the following setting: (1) we train the state encoder $\mathcal{G}$ using the CLEVR dataset [38], (2) we use $\mathcal{G}$ to encode the scene images of the photorealistic Blocks world dataset [2] to evaluate our overall planning framework.

**Table 4.1:** Reasoning and reconstruction performance

| Loss function | QA accuracy | Reconstruction error |
|---|---|---|
| KL-Divergence only | **0.932923** | 0.118789 |
| Reconstruction only | 0.676227 | **0.000464** |
| Both | 0.931829 | 0.002303 |

**Reasoning:** We analyze our VQA framework with three different objective functions (Table 4.1). We trained the framework with CLEVR dataset [38] for 150 epochs over the full training set. The training set consists of 70 000 images and 699 989 questions. With the $D_{\text{KL}}$ objective, our state encoder reasoning shows better performance than using only $\mathcal{L}_{\text{rec}}$ on the held-out test set. Figure 4.5 shows reconstructed images in each case. We observe that when using only $D_{\text{KL}}$ loss, the framework tends to neglect unnecessary details (e.g. background, shadows) as reflected by its larger reconstruction error but with a high QA accuracy.

**Planning with $\mathcal{M}_1$:** Following Asai and Fukunaga [4], we show that our learned representation is compatible with planning algorithms, for example, with a breadth-first search graph exploration. Note that one can achieve a planning accuracy of 100% with $\mathcal{M}_1$ on one condition: each state is *uniquely* encoded. With VQ-VAE, we fulfill this condition, thus achieve the absolute planning accuracy using $\mathcal{M}_1$ (Figure 4.1 and Figure 4.8). Similar to [5], we challenged this *uniqueness*

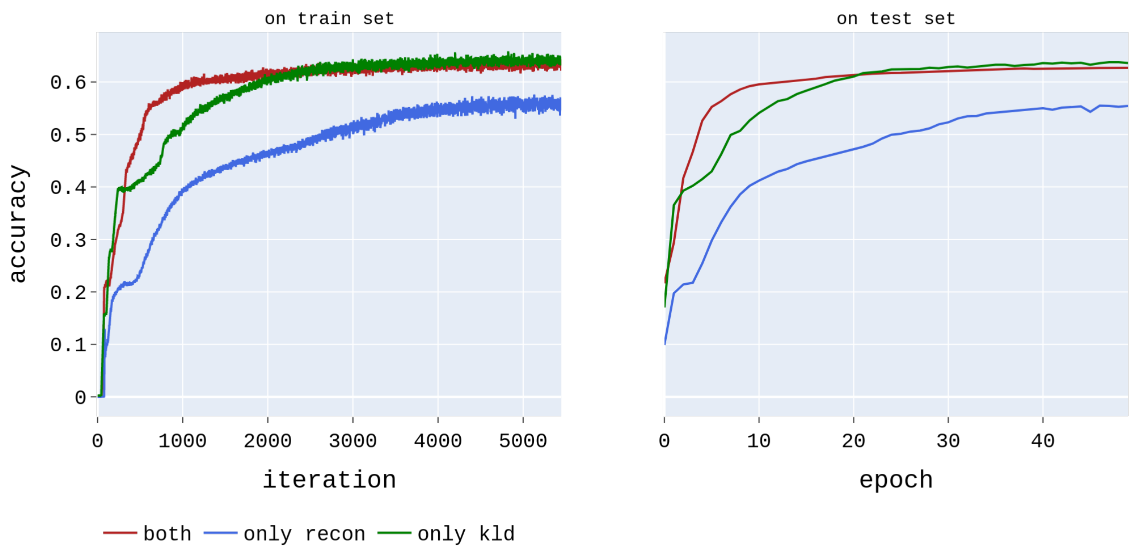

**Figure 4.4:** Accuracy of the stochastic action model overtime on the training (left) and test (right) sets using representations acquired by only $D_{\mathrm{KL}}$ loss (green line), $\mathcal{L}_{\mathrm{rec}}$ loss (blue line), and both (red line).

under noisy settings. Each scene image is perturbed with noise sampled from a standard normal distribution (Figure 4.7) $\mathcal{N}(0, 1)$, i.e., $\tilde{x}_i = x_i + \alpha u$ where $\alpha$ is a noise magnitude and $u \sim \mathcal{N}(0, 1)$. Our latent representations are less affected by noise, as shown by smaller deviation between $\mathcal{G}(x_i)$ and $\mathcal{G}(\tilde{x}_i)$ (Figure 4.6), when trained with $D_{\mathrm{KL}}$ loss. However, the effects are magnified under heavier noise ($\alpha > 0.01$).

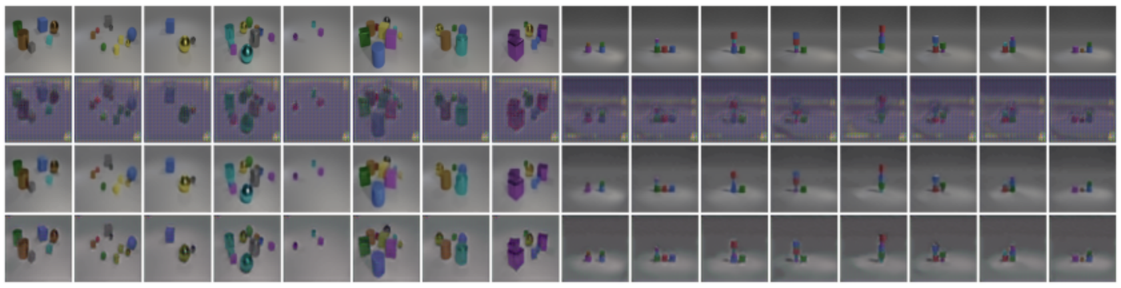

**Figure 4.5:** Qualitative results of our reconstructed images when trained under three schemes. From top to bottom: *first row:* real images, *second row:* reconstructed images using only $D_{\mathrm{KL}}$ loss, *third row:* reconstructed images using only $\mathcal{L}_{\mathrm{rec}}$ loss, *fourth row:* reconstructed images using both losses.

**Planning with $\mathcal{M}_2$:** We evaluate our learned representations in a stochastic manner with another action model, $\mathcal{M}_2$. We first encode all the transitions in the problem domain of 5 objects and 3 stacks and use these transitions to train $\mathcal{M}_2$. After training, we evaluate $\mathcal{M}_2$ using the transitions in the problem domain of 4 objects and 3 stacks. We report the accuracy by comparing the predicted latent $\tilde{z}_{\mathrm{to}} = \mathcal{M}_2(z_{\mathrm{from}}, a)$ to the ground-truth latent $z_{\mathrm{to}}$, i.e., accuracy $= \frac{1}{M}\frac{1}{N}\sum\sum_{i=1}^{N} u_i$ where $M$ is the size of the test set, $z_m^n$ is the $n$-th bit of the latent $z_m$ ($1 \leq n \leq N$), and $u_i = 1$ if $\tilde{z}_{\mathrm{to}}^i = z_{\mathrm{to}}^i$ and 0

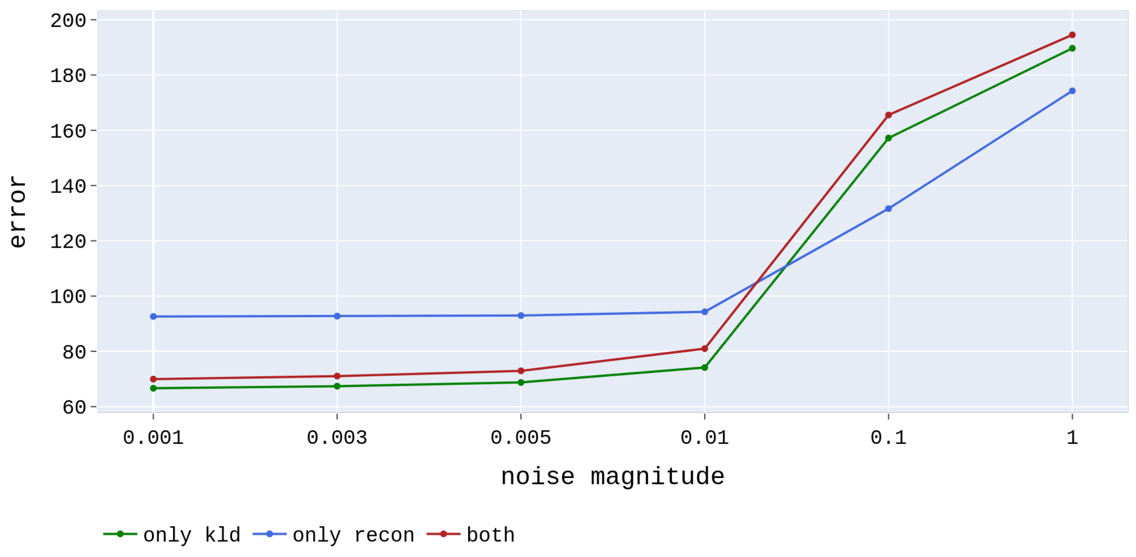

**Figure 4.6:** Absolute difference of latent representations when encoded under *noisy* versus *normal* setting.

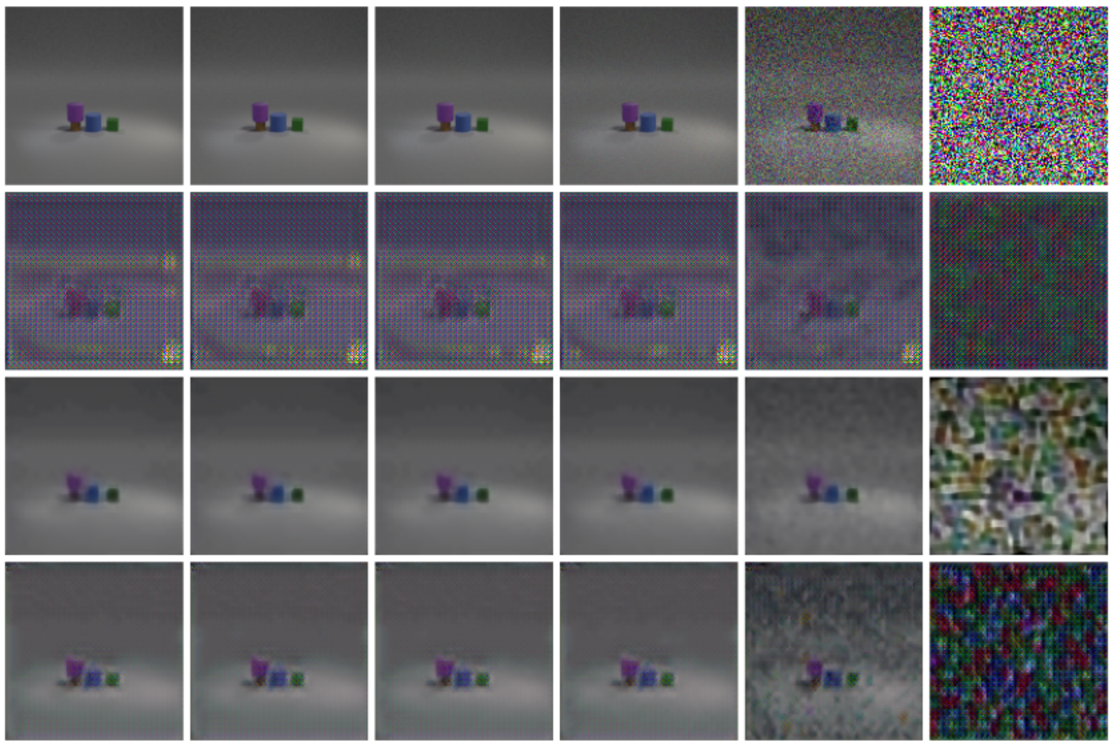

**Figure 4.7:** Reconstruction quality under different magnitudes of addition noise. From *left* to *right*: six noise levels: 0.001, 0.003, 0.005, 0.01, 0.1, and 1. From *top* to *bottom*: real images, reconstructed images using only $D_{\mathrm{KL}}$ loss, using only $\mathcal{L}_{\mathrm{rec}}$ loss, and both.

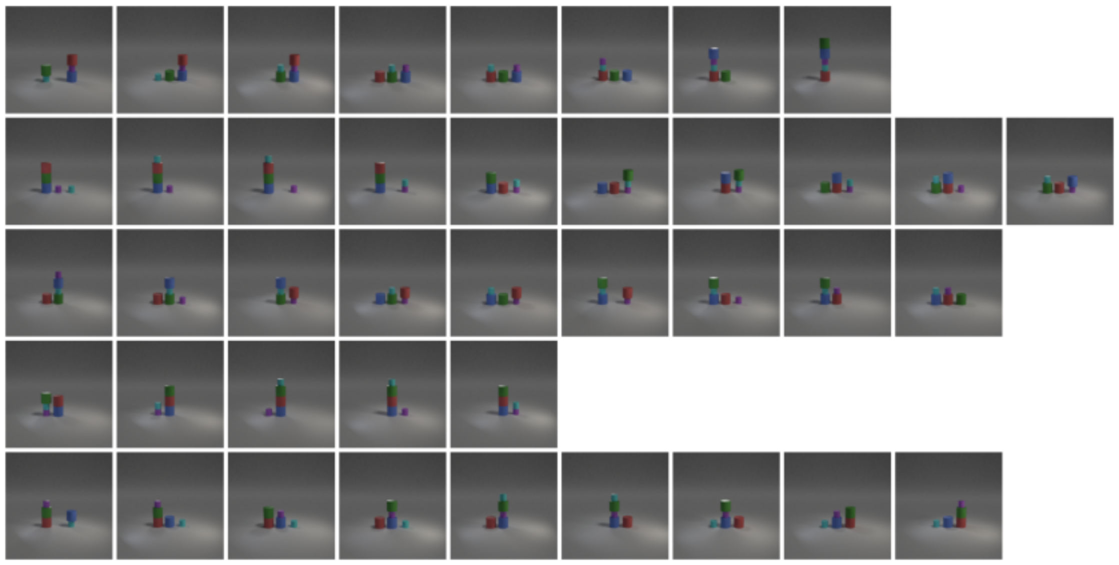

**Figure 4.8:** Some more plan sequences using $\mathcal{M}_1$.

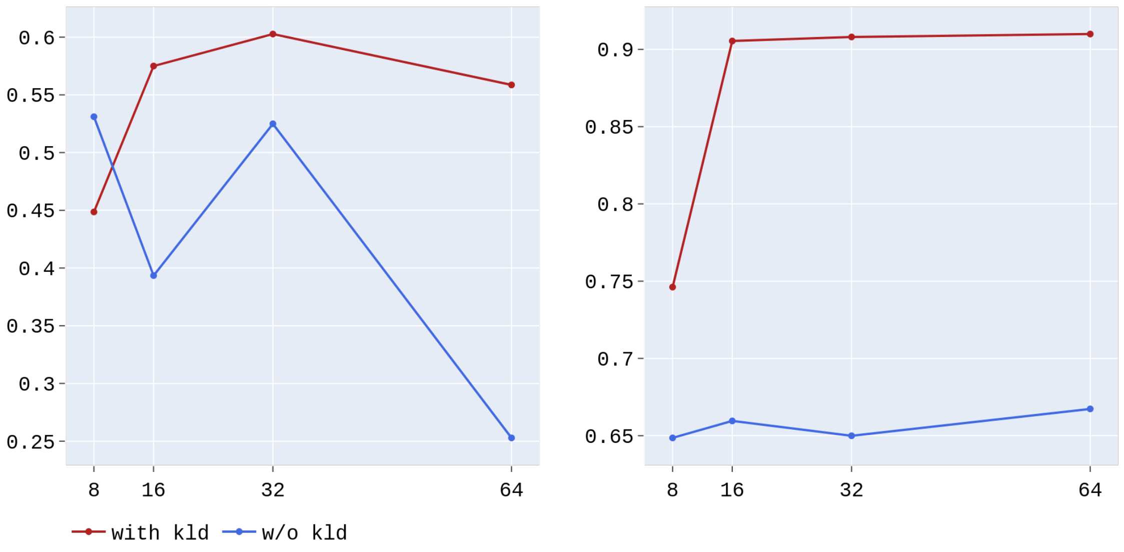

**Figure 4.9:** Planning (*left*) and QA (*right*) accuracies using different latent sizes.

otherwise. For this action model $\mathcal{M}_2$, our experiment shows that the latent representation trained with $D_{\text{KL}}$ objective enables easier learning of transition rule even when $\mathcal{G}$ does not observe the action data [2] (Figure 4.4).

**Efficiency:** We investigated the effect of the latent vector dimensionality, i.e., $\dim(\mathcal{G}(x)) = (o, 32, 32)$ where $o \in \{8, 16, 32, 64\}$. We kept $o = 64$ for all the previous experiments. Due to computational time constraints, we used a subset of the CLEVR dataset [38] consisting of only 50 000 questions. We train our VQA framework using this smaller version under two settings: optimizing *only* $\mathcal{L}_{rec}$ *loss* and *both* $D_{KL}$ *and* $\mathcal{L}_{rec}$ *loss*. With $D_{\text{KL}}$ loss, our state encoder achieves better QA and planning accuracy even for smaller latent sizes (Figure 4.9).

## 4.4 Conclusion

We introduced a novel method to train a state encoder and integrated it into a classical planner. We are able to obtain latent representations that are higher quality in terms of visual reasoning, and thus feasible for planning. We also presented a full *symbolic* planning framework to readily unify with the state encoder. Our experiments demonstrate better performance of the whole system when pre-trained with the proposed multi-modal scheme.

The limitation of our framework lies at the $\pi$ component of the action model. While doing sufficiently well in our toy experiment [2], $\pi$ has to perform better to be effective under more complex settings. Another aspect, which renders our framework semi-automatic, is that a set of actions $\mathbf{A}$ must be defined explicitly. A fully autonomous agent has to learn applicable actions and their effects itself. This work takes the first steps towards achieving this goal.

While classical AI planners are efficient and optimal, high-dimensional state representations, such as visual data, hinders their applicability on real-world robotic manipulation problems [16]. Our work hopefully sheds light on how to obtain such representations by taking advantage of auxiliary (linguistic) data. The presented planner is ready to realize high-level plans, and can be integrated into a motion planner (e.g., [67, 68, 69]) for robotic sequential manipulation tasks.

# 5 Self-supervised Learning of Scene-Graph Representations

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 from visual sensory data to alleviate the resulting knowledge acquisition bottleneck. We compute *scene graphs* that represent objects ("red box"), and their spatial relationships ("yellow cylinder on red box"). Leveraging visual perception allows us to learn a representation with which we can plan high-level actions using a graph search. 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. As the main contributions of this work[1]:

- 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 *task and motion planning* directly from visual data to solve sequential robotic manipulation tasks effectively.

## 5.1 Problem Statement

This works relies on the non-linear programming (NLP) formulation of the Logic-Geometric Programming (LGP) framework [28, 68] 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 geometrical 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 geometrical constraints are imposed by this high-level planning on the NLP. This modification on planning still allows us

[1]This chapter has been revised, slightly rewritten, and submitted to the Conference on Robot Learning (CoRL) 2020.

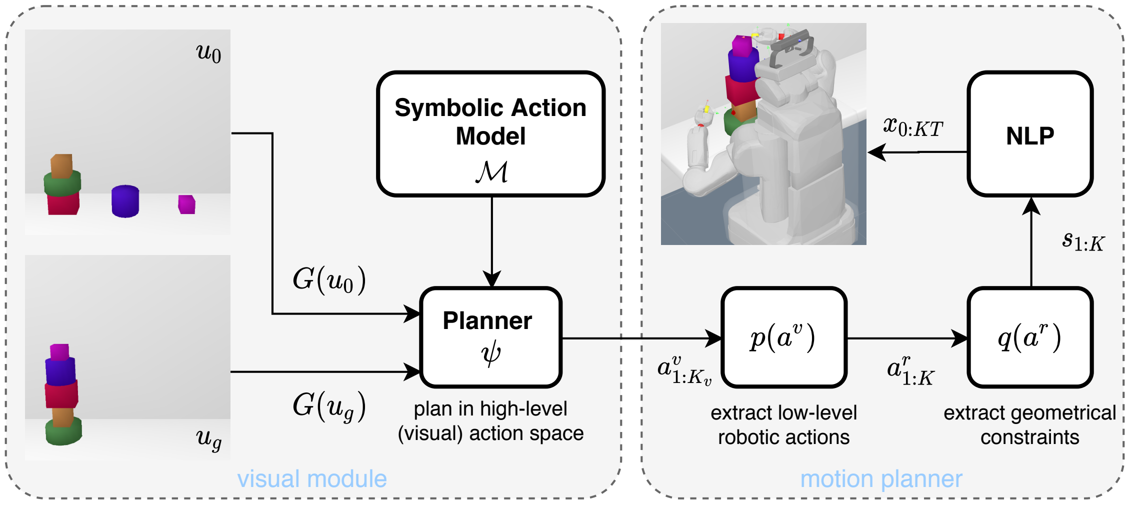

**Figure 5.1:** Flow of the proposed framework. *Left*: High-level action planning: a visual reasoning module consisting of a scene graph representation (Section 5.2.1) and an action model (Section 5.2.2), and *Right*: Motion optimization: a nonlinear programming (NLP) component (Section 5.1).

to use the original NLP formulation of the LGP only with slight changes (Figure 5.1). 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:

$$\begin{aligned}
\min_{\substack{x,K,\\ a^r_{1:K}, s_{1:K}}} \quad & \int_0^{KT} c(\overline{x}(t), s_{k(t)})\,\mathrm{d}t \\
\text{s.t.} \quad & x(0) = x_0, \qquad s_0 = \tilde{s}_0, \qquad h_{\text{goal}}(x(KT), g) = 0 \\
& \forall t \in [0, KT] : \quad h_{\text{path}}\big(\overline{x}(t), s_{k(t)}\big) = 0, \quad g_{\text{path}}\big(\overline{x}(t), s_{k(t)}\big) \le 0 \\
& \forall k \in 1, \ldots, K : h_{\text{switch}}(\hat{x}(t), a^r_k) = 0, \quad g_{\text{switch}}(\hat{x}(t), a^r_k) \le 0 \\
& \qquad a^r_k \in \mathcal{A}^r(s_{k-1}), \qquad s_k \in q(s_{k-1}, a^r_k),
\end{aligned} \tag{5.1}$$

where $\overline{x} = (x, \dot{x}, \ddot{x})$ and $\hat{x} = (x, \dot{x})$. $\tilde{s}_0$ describes the initial geometrical constraints, $g_{\text{path}}$ and $h_{\text{path}}$, the (in)equality constraints for the motion path, and $h_{\text{switch}}$ and $g_{\text{switch}}$, 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 scene graphs. In particular, the goal of the visual planner $\psi$ is to find an action sequence $a^v_{1:K^v}$, given an image of an initial scene $u_0$, and a desired scene $u_g$,

$$\begin{aligned}
\operatorname{argmin}_{a^v_{1:K^v}} \quad & \sum_{i=1}^{K^v} c^v(z_{i-1}, a^v_i) \\
\text{s.t.} \quad & z_0 = \mathcal{G}(u_0), \qquad z_N = \mathcal{G}(u_{\text{goal}}), \\
& z_{i+1} = \mathcal{M}(z_i, a^v_i), \quad a^v_i \in \mathcal{A}^v
\end{aligned} \tag{5.2}$$

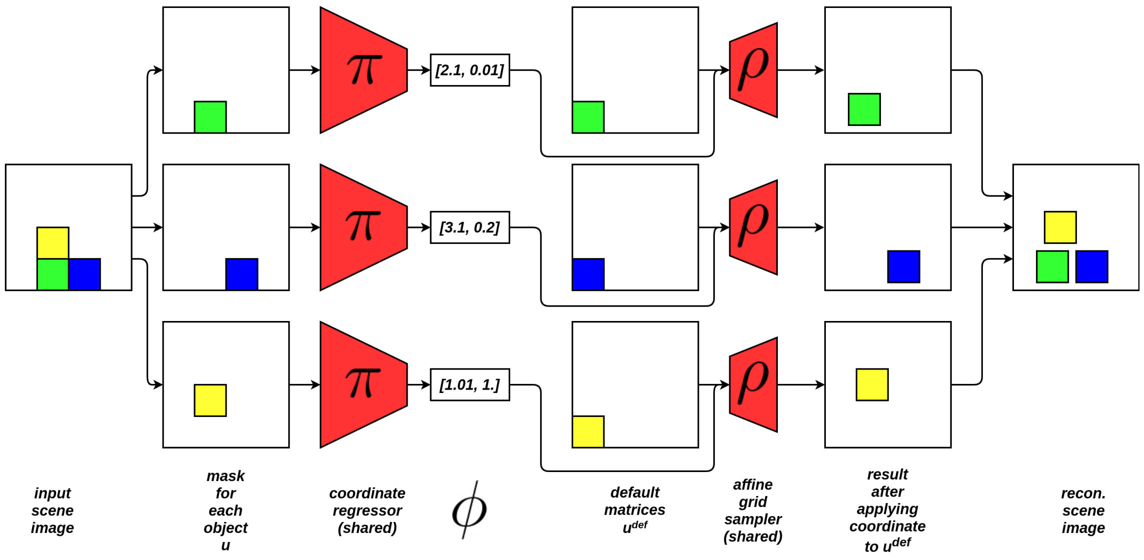
[2.1, 0.01]
[3.1, 0.2]
[1.01, 1.]
input scene image
mask for each object u
coordinate regressor (shared)
$\phi$
default matrices $u^{def}$
affine grid sampler (shared)
result after applying coordinate to $u^{def}$
recon. scene image

**Figure 5.2:** A forward pass through the visual module $\mathcal{G}$ with the coordinate regressor $\pi$ and the affine grid sampler $\rho$ (both depicted in red) from the input scene image to the reconstructed scene image.

where $c^v$ defines the optimality criterion, $z_0$, $z_N$ is the initial and final scene graph 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 geometrical constraints $s_{1:K} = q(a^r_{1:K})$ (Fig. Figure 5.1). 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 geometrical 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.

## 5.2 Methodology

We present a unified framework to solve sequential robotic manipulation tasks. The three main components comprise 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 from the image pair $(u_0, u_{\text{goal}})$ using the visual module $\mathcal{G}$ (Figure 5.2), (*ii*) we use the action model $\mathcal{M}$ to realize a high-level plan from $u_0$ to $u_{\text{goal}}$, and (*iii*) we 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.

### 5.2.1 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 [24, 31, 59]. 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 scene graph describing spatial relations between $N$ objects.

#### Architecture overview

The visual module $\mathcal{G}$ consists of two sub-modules: coordinate regressor $\pi$ and affine grid sampler $\rho$, which was presented in [36] (illustrated by the red boxes in Figure 5.2). Note that the visual module $\mathcal{G}$ is similar to the Spatial Transformer Network by Jaderberg et al. [36]. The only difference is that $\mathcal{G}$ only works with the translation vector $\phi$ instead of the full $3 \times 3$ transformation matrix. Next, we explain the details of those components and how they are combined.

$\pi$ **- Coordinate regressor:** The coordinate regressor infers a vector $\phi$ 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 $\phi$ denotes how far the object should be from the bottom-left corner (depicted in the fourth column of Figure 5.2). The backbone of the $\pi$ module is a Res-Net 34 network [32]. This module is pre-trained[2] on the ImageNet dataset [62]. We replace the last layer with a fully-connected layer to output $\phi$.

$\rho$ **- Affine grid sampler:** The affine grid sampler takes the vector $\phi$ and transforms a default matrix $u^{\text{def}}$ (depicted in the fifth column of Figure 5.2) into the desired location which is implied by $\phi$, i.e., $\tilde{u}_i = \rho(\phi, u_i^{\text{def}})$. The module $\rho$ is differentiable, hence capable of providing feedback on how to *spatially* transform the default matrices $u_i^{\text{def}}$, i.e., optimizing the vector $\phi$ from the $\pi$ according to a label signal. See Figure A.3 for examples of $u_i$ and $u_i^{\text{def}}$.

#### Training objective

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

$$l = l_{\text{mse}}\left(\sum_{i=1}^{N} \tilde{u}_i, \sum_{i=1}^{N} u_i\right) + \sum_{i=1}^{N} l_{\text{mse}}\left(\tilde{u}_i, u_i\right) \tag{5.3}$$

where $l_{\text{mse}}$ 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.

[2]The pre-trained weights are available in the TorchVision library [56].

We compute $\tilde{u}_i$ by $\tilde{u}_i = \rho(\pi(x_i), u_i^{\text{def}})$ where $u_i^{\text{def}}$ is the default matrix in which the bounding box of $o_i$ is located at the bottom left corner (Figure 5.2).

#### Coordinates to scene graph

Our scene graphs 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 $o_2$ (Figure 5.3). We first use the k-means clustering algorithm to group horizontal coordinates into $m$ clusters, where $m$ is the number of stacks. 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 scene graph representation of triples.

### 5.2.2 Symbolic action model

After capturing the scene graphs 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 scene graphs for the image pair $(u_0, u_{\text{goal}})$, the action model is guaranteed to find the optimal plan if the goal can be satisfied. This is possible as our planner relies on scene graph representation for intermediate states. To find the action sequence, we start from the input scene graph, use this action model to explore the neighboring graphs from the action set, and add these neighboring graphs into a queue-like data structure (Figure 5.3). 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 scene graph $z_{\text{goal}}$ which is either encoded by $u_{\text{goal}}$ or user-specified (see Figure 5.4 for sample plans).

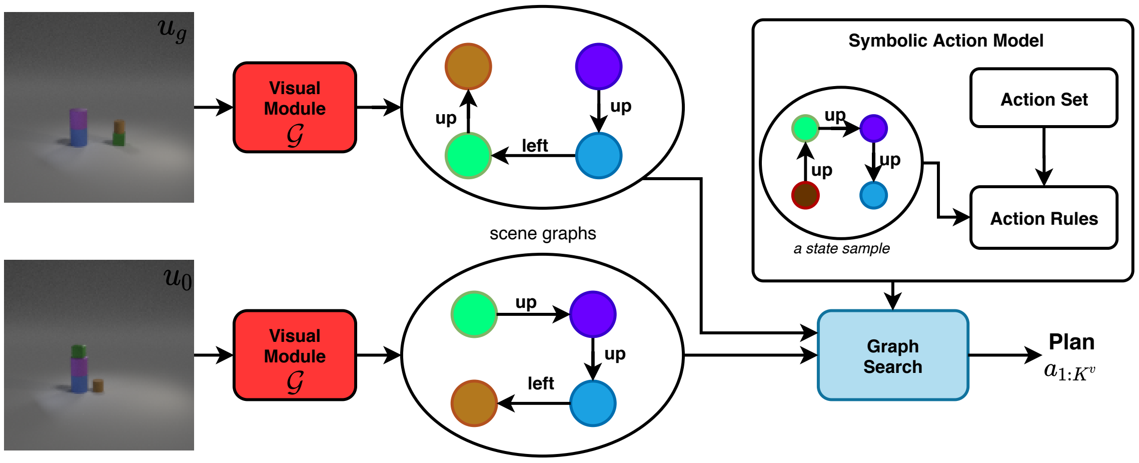

**Figure 5.3:** An overview of how we use the symbolic action model $\mathcal{M}$ and the scene graph representation to perform graph search.

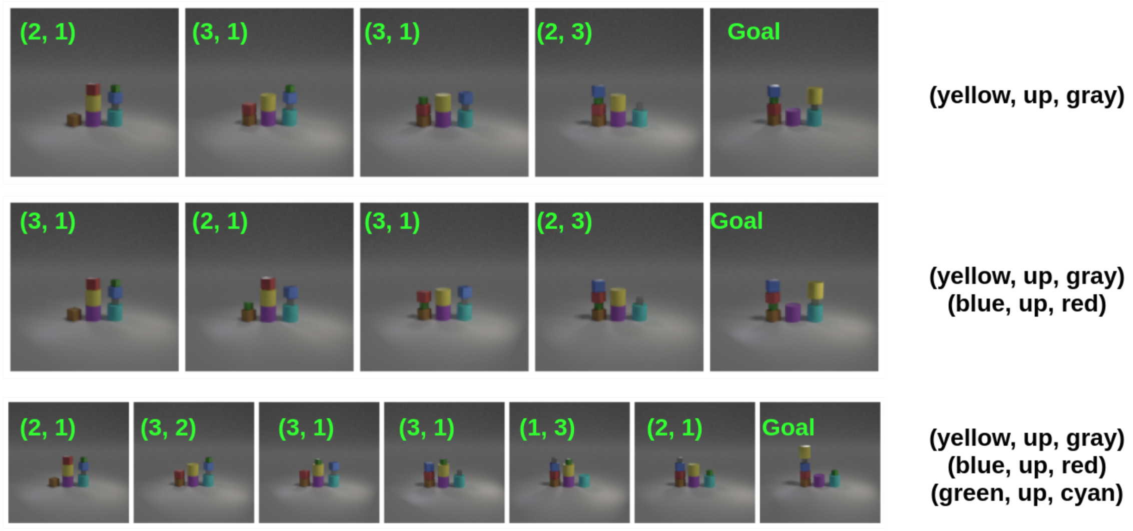

**Figure 5.4:** Three sample plans. Initial images $u_0$ are the left-most images of each row. Desired relations $z_{\text{goal}}$ (the right-most column) are specified as textual commands instead of desired images $u_{\text{goal}}$, which is also possible (Figure A.4). (*Green*) Action, which transitions the current state to the next, is located at the top-left corner of each image. Action notation follows Section 5.2.2 and Section 5.3.1.

### 5.2.3 Motion planner

Once an action sequence is found by the previous module, any optimization library can be used to solve the problem described in Equation (5.1). We use KOMO ($k$-th order markov optimizer) [66], and the changes introduced in [28] that makes the solver more robust, to formulate and solve Equation (5.1). KOMO represents the trajectory directly in configuration space, and computes velocities and accelerations as finite differences of configurations. This leads to a constrained sum of squares problem with a banded symmetric matrix, that can be efficiently solved using off-the-shelf Gauss-Newton methods. Note that, given the action sequence, any off-the-shelf motion planner that can deal with multi-modal problems could be used as well, such as the ones introduced in [29, 43].

## 5.3 Evaluation

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. Therefore, we structure and perform experiments to investigate in which novel settings our model generalizes and does *not* generalize. This analysis can be useful to understand when the model needs to re-calibrate, although this can be achieved automatically using the *Intersection-over-Union* indicator. We also evaluate our planner $\psi$ with a robotic arm in a simulation.

### 5.3.1 Setup

We evaluate our framework using the Blocks-world task [26] 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 $a^v_i = (p,q)$ 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, from the work of Asai [2], is used only to render photo-realistic images to collect the dataset $\mathcal{D}_{\text{pbw}}$. The second dataset $\mathcal{D}_{\text{sim}}$ 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 $\mathcal{D}_{\text{sim}}$.

### 5.3.2 Results

**Table 5.1:** Performance on different evaluation datasets.

| **Dataset** | **SG prediction 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 |

Since our action model $\mathcal{M}$ is oracular, the planning accuracy depends on whether $\mathcal{G}$ correctly captures the scene graphs of the desired scene images. Therefore, we focus our analysis on the performance of $\mathcal{G}$. We report the Intersection-over-Union (IoU) metric [60] (Figure 5.5) and the scene graph prediction accuracy for our experiments (Table 5.1). The IoU is computed between the ground-truth $u_i$ 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 scene graph prediction accuracy and IoU metric (first 3 rows of Table 5.1). During plan execution, the robot may 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 good enough to retain accuracy in predicting scene graphs (Table 5.1 last 3 rows).

**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.

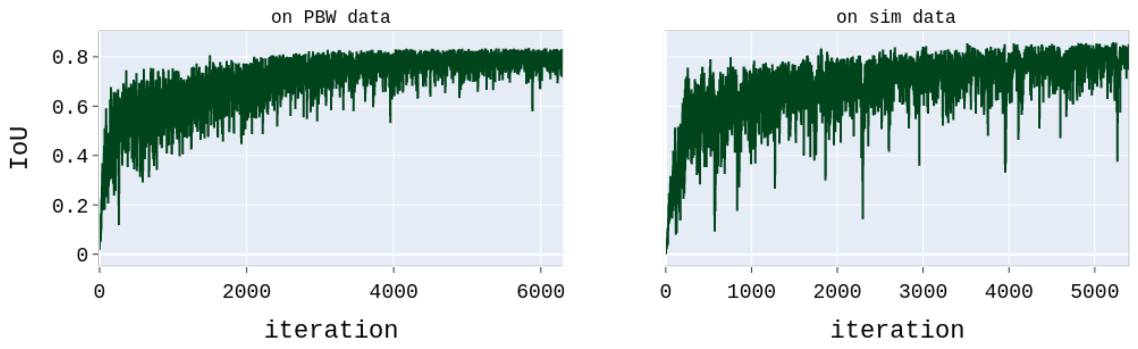

**Figure 5.5:** IoU on the (5 objects, 3 stacks) training sets of $\mathcal{D}_{\text{pbw}}$ (*left*), and $\mathcal{D}_{\text{sim}}$ (*right*).

**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 material). Even though the original LGP formulation struggles to solve problems involving many objects (e.g., the number of objects $N \geq 5$) and which requires long action sequences (e.g., action sequence length $K \geq 6$) [16], 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 [28]. We merely want to point out that the visual representations learned by our method proves to be a useful and feasible component for task and motion planning approaches.

## 5.4 Extension to generalize in real-world settings

### 5.4.1 Dealing with random stack locations

**Table 5.2:** Performance comparison when randomly translating stacks horizontally.

| $\epsilon_{\max}$ | **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** |

The visual module $\mathcal{G}$ does not generalize well to novel locations of stacks. We randomly added a distance $\epsilon$ ($0 \leq \epsilon \leq \epsilon_{\max}$) 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 $\epsilon_{\max}$ (Table 5.2). This issue can be fixed by first detecting the stack regions by a Connected-component labeling algorithm[3] [22]. 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). Table 5.2 shows that this technique improves the prediction significantly, hence successfully enhances the ability of $\mathcal{G}$ to novel locations.

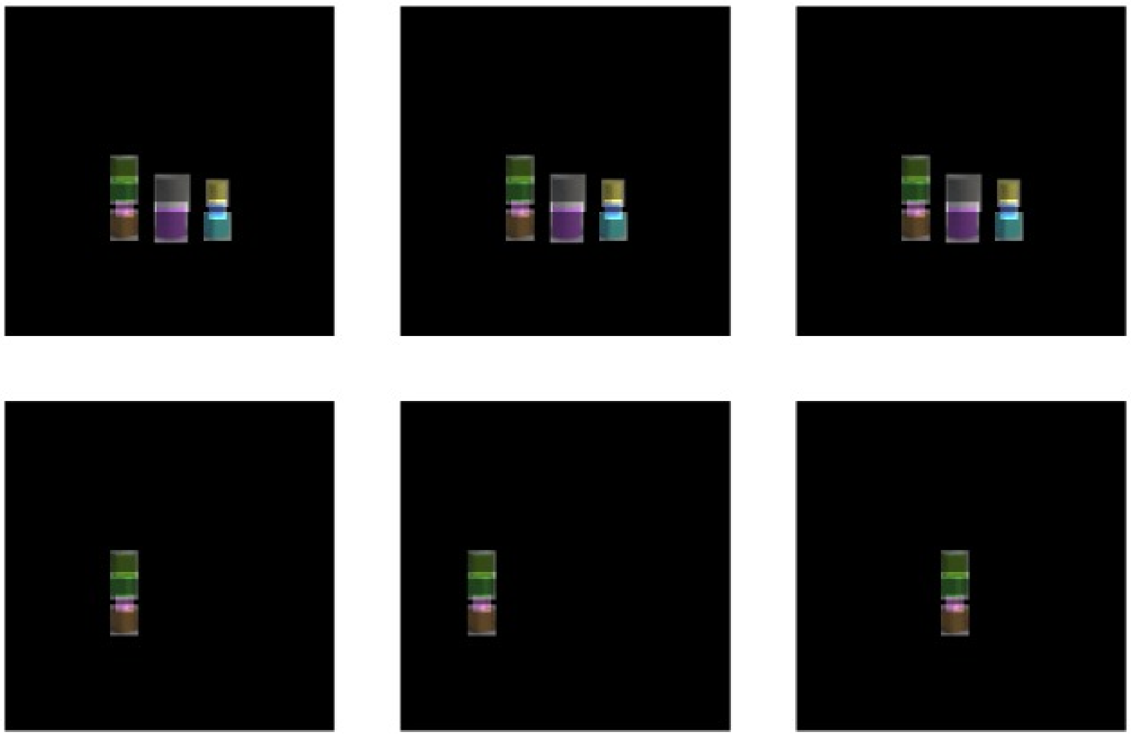

**Figure 5.6:** Illustrations of how to translate stacks in order to generalize better to novel locations. *Top row* illustrates the original locations while *bottom row* illustrates the translated locations of the left-most stack. From *left corner* to *right corner*, we translate the stack by 0, -15 and 15 pixels.

### 5.4.2 Incorporating stability into the plan search

Taking no consideration of physical constraints in the real world, the high-level plans from Figure 5.3 can include actions which may cause instability of the stack structure. For example, there may exist a spherical shape, on top of which the output plan should avoid stacking any shape. Hence, it is more realistic if the graph search can be informed of any heuristic that restrict the plan to obey the physics rules. Groth et al. [25] proposed to predict the stability of a stack of objects from just an image observation. This method is motivated by the *center of mass* rule, thus predicts if the center of mass of the top object lies within the surface of the bottom object. While showing good performance on

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

simulation images, there are two limitations of the method. First, the method of Groth et al. [25] did not generalize well to real images when having an accuracy drop of approximately 20%. Second, the method required the input image to infer the stability of that image. However, in the context of planning, the image, which we are interested in inferring, is usually from the future horizon. Therefore, the method is applicable when we can sample the images of possible neighboring states of a current state. Training such an image sampler in a semantically rich environment is a challenging and open problem, particularly when the effects of actions are large and highly complex. We thus present a more realistic technique which is based on capturing the properties of the scene objects.

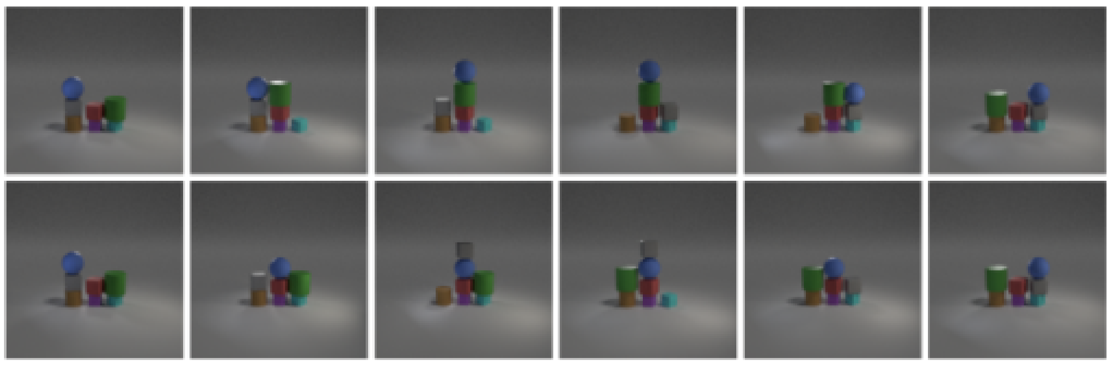

**Figure 5.7:** A sample plan when stability is enforced (*top*) and not enforced (*bottom*). Note that the former plan does not involve stacking any object on top of the sphere.

### 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* train to 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 [27] (see Algorithm A.2). Figure 5.7 shows an example of a planning sequence when stability is enforced.

## 5.4.3 Possibility to include 3-D relations

Spatial 2-D relationships, such as left, right, up and down, do not capture extensively 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 graphs of 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 using a dataset of 5 objects and 3 stacks (in Section 5.3). The testing 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. See Figure 5.8 for some sample plans which involve the depth relationships.

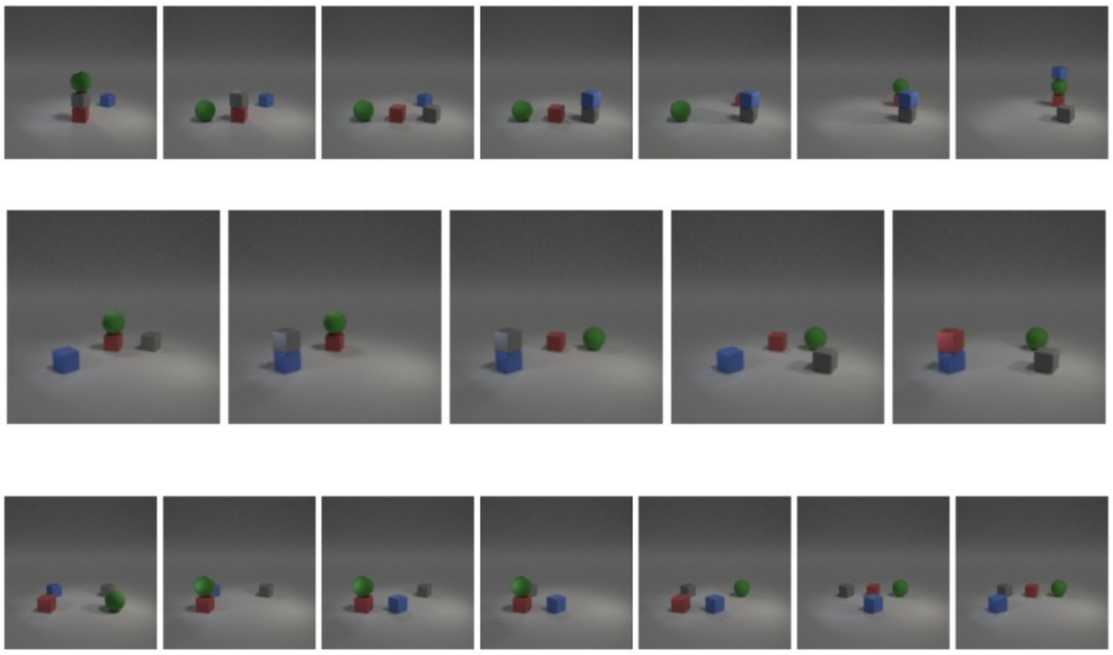

**Figure 5.8:** Planning samples with 3-D relationships.

## 5.5 Conclusion

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 then extract individual objects to infer individual coordinates. Even though we think that it is a reasonable assumption as there are many state-of-the-art object detection methods [24, 31, 59], this can be relaxed by adapting an unsupervised scene decomposition technique [8].

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 Section 5.4.1 of the sup. mat.) For a more realistic scene structure, our method can be further developed to reason about the 3-D world using the depth sensor. Exploiting the depth information, we implemented a *pre-checking* subroutine to handle occlusion. We detail this subroutine in Section 5.4.3 (Figure 5.8).

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. *(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, visual planner can verify if the plan is executed properly and adjust the plan whenever necessary.

For generic real-world settings, it is advantageous to incorporate physical constraints in addition to spatial relations among objects. These constraints help to reason about stacking stability, thus allow for safer manipulation. Since each object has different geometrical 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 [6]. We suggest one possible extension followed this idea in Section 5.4.2 (Figure 5.7).

This chapter 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. Relational structures learned autonomously by robots have the potential to improve the long-term autonomy of intelligent agents.

# 6 Discussion

**Results** Chapter 4 presents a multi-modal learning scheme to train a state encoder which produces higher quality representations in terms of visual reasoning, efficiency, and learning transition rules. First, under this learning scheme, our encoder produces higher reconstruction error, but high reasoning accuracy, indicating the neglect of unnecessary details. Second, the resulting latent representation enables easier learning of transition rule, when action data is not observed. Lastly, for the smaller latent size settings, our encoder under the multi-modal learning scheme outperforms its counterparts which are trained using normal scheme.

Chapter 5 addresses the difficulty of learning an accurate action model in Chapter 4 by proposing to use an *explainable* representation, thus enabling an *oracular* action model. Our proposed framework performed well on two synthetic datasets and showed good generalization to novel environmental settings (different amounts of objects). Further analysis also confirm sufficient performance under novel camera views, however, with a confidence drop. We successfully integrate our framework into an existing motion planner and observe correct plan execution in the simulation.

**Alternatives & Insights** We think that the evaluation method in Chapter 4 is not well-grounded due to the missing of a good *stochastic* action model. Formulating such an action model is very challenging and an open research question. To the best of our knowledge, [40, 47] are two of the early works tackling this problem. The action model by Kim et al. [40] samples the future image in settings where the action displacement is small, e.g., "move left by two pixels". Meanwhile, the action displacement in our problem setting is rather large and often intractable, e.g., "move this block to the third stack", thus rendering this method ineffective in our settings. The recent work of Lin et al. [47] achieves impressive results in the same direction, however, we did not have time to implement and evaluate this method. Nevertheless, formulating such an effective action model is a fruitful and interesting research direction, which highlights and complements well our contribution in Chapter 4.

During this thesis work, we have tried multiple different methods to solve the representation learning problem. Within Chapter 4, the missing piece is a stochastic action model to evaluate and highlight our proposed learning scheme as discussed above. Despite trying different formulations for such models (e.g., generative adversarial networks [40] with a dynamic engine and two discriminators, or conditional pix-to-pix network [35]), we failed to implement a working solution. Beside the difficulty of large action displacement, an additional reason for this failure is that objective functions, such as mean-squared error or discriminator loss, are not effective for learning the transition rules. Mean-squared error puts too much constraint on learning the transition rule, often results in too identical input-output image pairs. On the other hand, the discriminator loss fails to capture the real/fake space, since the fake space is often far larger than the real space.

While the method presented in Chapter 5 integrates well into an existing motion planner to solve TAMP problems effectively, the weakest link in our method is the assumption of accessible bounding boxes. Since those bounding boxes contain location information, there exists a trivial engineering technique to infer relative coordinates of the boxes. Therefore, the contribution in Chapter 5 would be much stronger if we incorporate a scene decomposition technique (e.g., [8]) into our framework. Note that such a decomposition technique may propose an irrelevant object (e.g., a shadow) and lead to an irrelevant relation (e.g., a shadow is behind a green cube). This effect, however, does not harm the framework, since an irrelevant relation will never appear among our desired relationships.

Another interesting and potential alternative direction to the method in Chapter 5 is *differential rendering* methods [64, 72] . Notice that if rendering is a process which goes from *parameters* (such as: colors, shapes, or locations) to *pixel values*, perception is just the inverse process of rendering process. Hence, if a rendering process is *differential*, we can construct a learning program which receives gradient feedbacks from such renderers and tries to reconstruct a scene observation. If the quality of the reconstruction is good enough, then the rendering parameters are sufficient to infer many object relationships. An example of this idea is the work of Sitzmann et al. [64], which generates novel views from the camera parameters (both extrinsic and intrinsic) and pixel coordinates. This method first predicts the distance from the camera / eye to the object, then compute the world coordinates using the pixel coordinates and the camera parameters. Finally, the learning loop is closed by a module which predicts the pixel value from the world coordinates. We emphasize that the world coordinates of objects are a valuable source of information to infer object relationships, and the technique of Sitzmann et al. [64] suggests how to learn this information in a self-supervised manner. Much inspired by this work, we attempted to formulate a similar method (see Equation (A.1)). This formulation, however, is inconsistent since due to a constraint of the mapping from a fixed world coordinate vector $xyz$ to a fixed color vector $\tilde{rgb}$. This conflicts with the fact that $xyz$ may have different color values depending on which scene image $I$. A more appropriate formulation would be conditioning on the current $I$, i.e., $\tilde{rgb} = \Xi(\Gamma_1(I), \Gamma_2(xyz))$. We believe that this formulation is one of the first steps towards applying interdisciplinary ideas from computer graphics to robotics, which is a potential and interesting research direction.

**Conclusion** In this thesis work, we have developed two methods to learn a state representation from image observations. The first method is to train a VQ-VAE [54] as an image encoder with a relation network [63] using a question-answering dataset [38]. This allows us to encode state representations which are more efficient (concentrating only on relevant objects), more stable under noisy settings, and easier to learn transition rules (with a *naive, stochastic* action model). The second method is about learning to predict scene graphs from images without ground-truth labels. Such a scene graph contains information about spatial relationships (e.g., up, down, left, and right), thus providing an invariant, interpretable, and symbolic state representation. This characteristic enables the use of an *oracular* action model and eliminate the need of a *stochastic* one. Therefore, our proposed technique integrates effectively into an existing motion planner [68] to solve for task and motion planning problems, such as: a classical Blocksworld domain [2]. Multiple generalizations of this technique are also discussed to aim for a more realistic robotic use case.

# A Appendix

## A.1 Algorithm pseudocode

**Algorithm A.1** Breadth-first search algorithm

```
procedure BFS($G$, $root$)
    $Q \leftarrow$ queue
    $root$.visited $\leftarrow$ true
    $Q$.put($root$)
    while $Q$ is not empty do
        $v \leftarrow Q$.pop()
        if $v = G$ then
            return $v$
        for all $w \in v$.neighbours()
            if $w$.visited = false then
                $w$.visited $\leftarrow$ true
                $w$.parent $\leftarrow v$
                $Q$.put($w$)
end procedure
```

**Algorithm A.2** A-star algorithm by Hart et al. [27]

```
procedure SEARCH($goal$, $root$, $h$)
    $Q \leftarrow$ array
    $Q$.put($root$)
    $P \leftarrow$ map
    $G \leftarrow$ map
    $F \leftarrow$ map
    $G[root] \leftarrow 0$
    $F[root] \leftarrow h(root)$
    while $Q$ is not empty do
        $v \leftarrow \min_{x \in Q}(F[x])$
        if $v = goal$ then
            $p \leftarrow$ array // solution path
            $p$.put($v$)
            while $v \in P$ do
                $v \leftarrow P[v]$
                $p$.put($v$)
            return $p$
        $Q$.remove($v$)
        for all $w \in v$.neighbours()
            $g \leftarrow G[v] + 1$
            if $w$ not in $G$ or $g < G[w]$ then
                $P[w] \leftarrow v$
                $G[w] \leftarrow g$
                $F[w] \leftarrow g + h(w)$
                if $w$ not in $Q$ then
                    $Q$.put($w$)
    return null
end procedure
```

## A.2 Supplementary materials for Chapter 5

### A.2.1 $\mathcal{G}$-network architecture and training

To implement $\pi$, we replace the last layer of ResNet-34 [32] with one sigmoid-activated fully-connected layer $FC(512, 2)$. Suppose the output of $\pi$: $\phi = [a, b]$, our $2 \times 3$ transformation matrix $\alpha$ 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 $\phi$ covers a good area of the $128 \times 128$ image plane. We then pass $\alpha$ to the grid sampler $\rho$ to transform the default matrix $u^{\text{def}}$ to the desired location implied by $\phi$. Table A.1 details our training hyper-parameters for training $\mathcal{G}$.

**Table A.1:** Training-related hyper-parameters

| Parameter | Value |
|---|---|
| Optimizer | Adam [41] |
| Learning rate | $1 \times 10^{-3}$ |
| Weight decay | $1 \times 10^{-4}$ |
| Batch size | 16 |

### A.2.2 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 (Figure A.1). 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.

### A.2.3 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 [40]. 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.

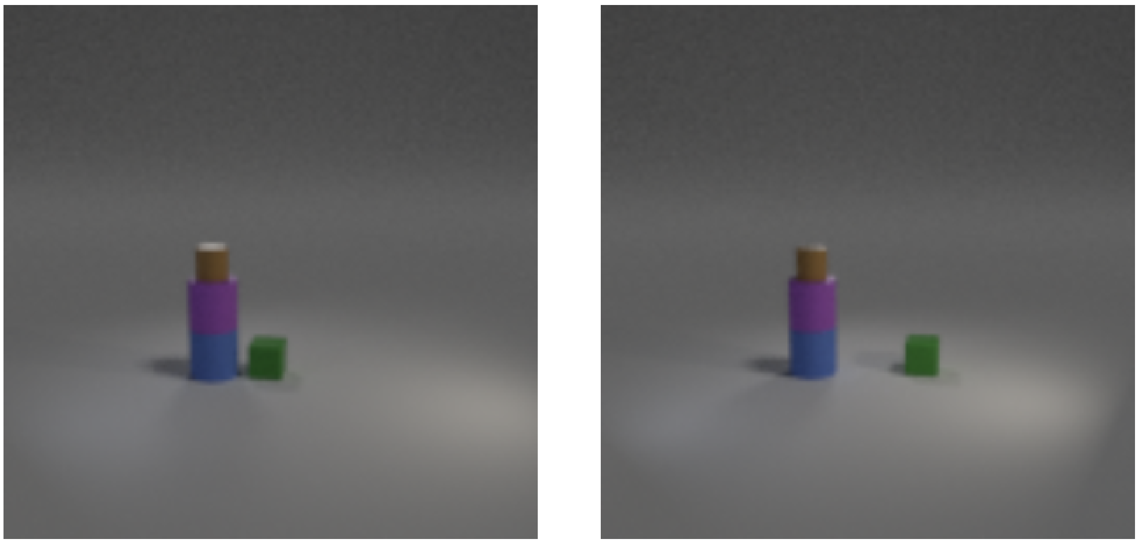

**Figure A.1:** Two different states share the same scene graph representation ({brown, up, pink}, {pink, up, blue}, {blue, left, green}). New scene graph: *left*: ({brown, up, pink}, {pink, up, blue}, {blue, up, stack0}, {green, up, stack1}), *right*: ({brown, up, pink}, {pink, up, blue}, {blue, up, stack0}, {green, up, stack2}). Here *stack0*, *stack1*, and *stack2* are three additional imaginary bases.

### A.2.4 Knowing $k$ in advance

Our limitation of requiring the number of stacks $k$ to be known in advance can be addressed by a simple algorithm. This algorithm takes in the mask image $u_i$ as an input and outputs $k$. Starting at the bottom of $u_i$, the algorithm draws a horizontal line from the left-most pixel to the right-most pixel, then sees how many times this line intersects with the stacks. Finally, the algorithm outputs the number of stacks $k$ as $k = \frac{1}{2}\max(K)$, where $K = \{k_i | 1 \leq i \leq 128\}$, $k_i$ is the number of intersections between the horizontal line at row $i$ and the mask area, and 128 is the number of rows in $u_i$.

### A.2.5 Additional Analysis on IoU Metric

We discussed in Section 5.5 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 $f_{\text{min}}$ and 0.266 for $f_{\text{mean}}$, therefore suggests that $f_{\text{min}}$ is a better indicator function due to the lower number of potential false positives. Figure A.2 also reflects this as we see a bigger overlapping area between the sets of correct and incorrect predictions (right column).

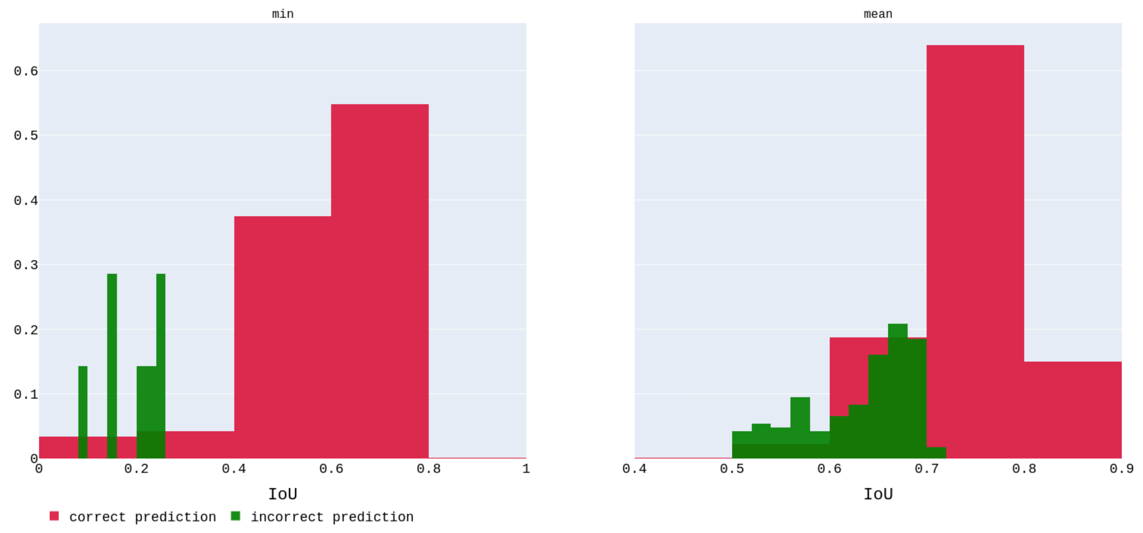

**Figure A.2:** Histogram of occurrences for different IoU values in the set of incorrectly and correctly predicted scene graph.

### A.2.6 Image Masks and Planning with Goal Images

We present additional figures to further support Chapter 5 of the main text. Figure A.3 illustrates different mask images $u_i$ and default matrices $u_i^{\text{def}}$. These samples are either rendered by [2] (top row) or our simulator (bottom row). Figure A.4 shows three additional plans which are different from those presented in the main text. In these samples, we specify our desired scene *images* instead of relations.

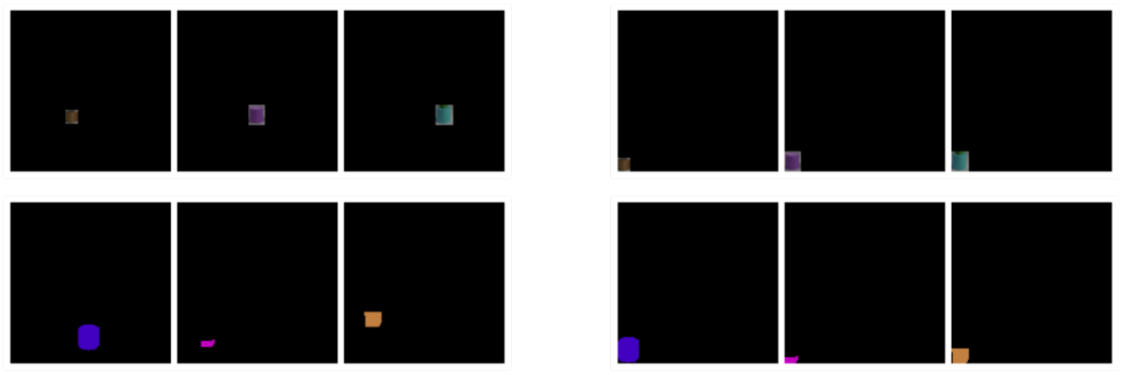

**Figure A.3:** *Left column:* mask images $u_i$. *Right column:* default matrices $u_i^{\text{def}}$. *Top row:* samples from $\mathcal{D}_{\text{pbw}}$. *Bottom row:* samples from $\mathcal{D}_{\text{sim}}$.

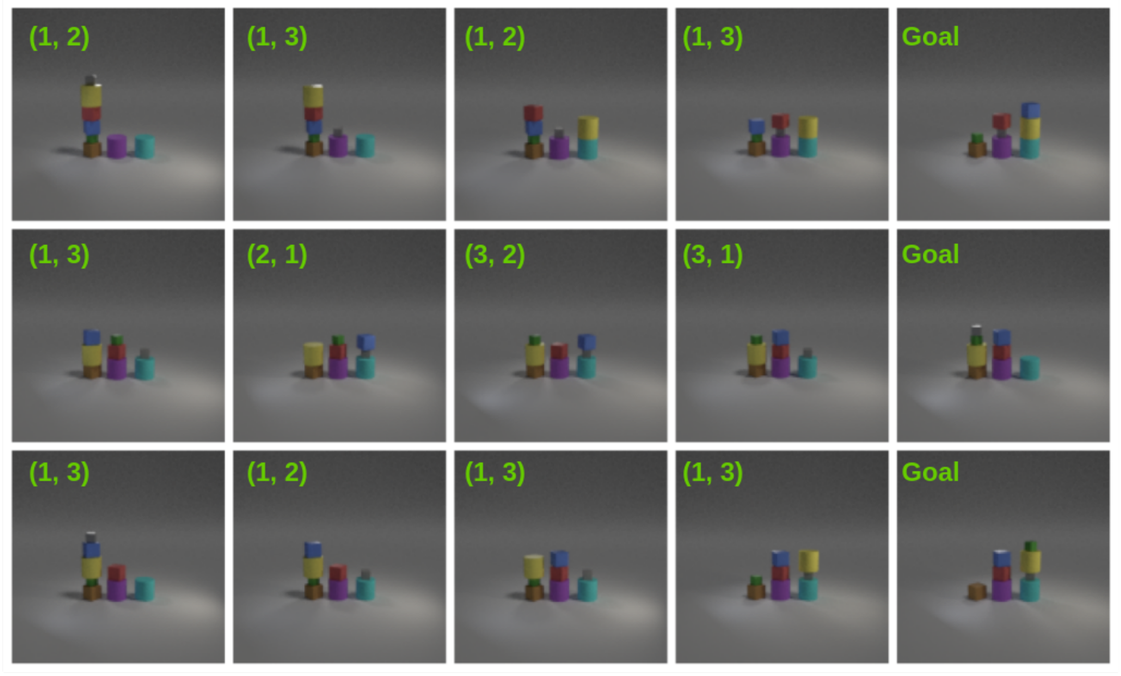

**Figure A.4:** Three sample plans illustrated in each row. Initial images $u_0$ are the left-most, desired scenes $u_{\text{goal}}$ are the right-most images of each row, respectively. Action (*green*), which transitions the current scene to the next, is located at the top-left corner of each image.

## A.3 Additional equations for Chapter 6

$$\begin{aligned} r\tilde{g}b &= \Xi(\Gamma_2(xyz)) \\ xyz &= P \cdot uvd \\ d &= \Upsilon(\Gamma_1(I), uv) \end{aligned} \tag{A.1}$$

where:

- $\Xi$ is the pixel generator, which is the same as in [64]
- $\Gamma_1$, $\Gamma_2$, $\Upsilon$ are represented by neural networks
- $P$ is the full projection matrix
- $I$ is the input image
- $d$ is the predicted depth value
- $uv$ is the 2-D vector, which denotes the pixel coordinate at $(u, v)$
- $uvd$ is the 3-D vector, which is $(u, v, d)$
- $rgb$ is the 3-D vector, which denotes the true pixel color at $(u, v)$
- $r\tilde{g}b$ is the 3-D vector, which denotes the predicted pixel color at $(u, v)$
- $xyz$ is the 3-D vector, which denotes the world coordinate of the pixel at $(u, v)$

# Bibliography

[1] J. Andreas, D. Klein, S. Levine. "Learning with Latent Language". In: *NAACL-HLT*. Association for Computational Linguistics, 2018, pp. 2166–2179 (cit. on p. 16).

[2] M. Asai. "Photo-Realistic Blocksworld Dataset". In: *arXiv preprint arXiv:1812.01818* (2018) (cit. on pp. 14, 29, 32, 33, 41, 48, 53).

[3] M. Asai. "Unsupervised Grounding of Plannable First-Order Logic Representation from Images". In: *ICAPS*. AAAI Press, 2019, pp. 583–591 (cit. on pp. 13–15).

[4] M. Asai, A. Fukunaga. "Classical Planning in Deep Latent Space: Bridging the Subsymbolic-Symbolic Boundary". In: *AAAI*. AAAI Press, 2018, pp. 6094–6101 (cit. on pp. 13–15, 28, 29).

[5] M. Asai, H. Kajino. "Towards Stable Symbol Grounding with Zero-Suppressed State AutoEncoder". In: *ICAPS*. AAAI Press, 2019, pp. 592–600 (cit. on pp. 13, 29).

[6] F. Baradel, N. Neverova, J. Mille, G. Mori, C. Wolf. *CoPhy: Counterfactual Learning of Physical Dynamics*. 2019. arXiv: 1909.12000 [cs.CV] (cit. on p. 46).

[7] Y. Bengio, A. C. Courville, P. Vincent. "Representation Learning: A Review and New Perspectives". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 35.8 (2013), pp. 1798–1828 (cit. on p. 13).

[8] C. P. Burgess, L. Matthey, N. Watters, R. Kabra, I. Higgins, M. Botvinick, A. Lerchner. "MONet: Unsupervised Scene Decomposition and Representation". In: *CoRR* abs/1901.11390 (2019) (cit. on pp. 45, 48).

[9] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, P. Abbeel. "InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets". In: *NIPS*. 2016, pp. 2172–2180 (cit. on pp. 13, 18).

[10] A. R. Conn, N. I. M. Gould, P. L. Toint. "A Globally Convergent Augmented Lagrangian Algorithm for Optimization with General Constraints and Simple Bounds". In: *SIAM J. Numer. Anal.* 28.2 (Feb. 1991), pp. 545–572. ISSN: 0036-1429. DOI: 10.1137/0728030. URL: https://doi.org/10.1137/0728030 (cit. on p. 23).

[11] R. Coulom. "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search". In: *Computers and Games*. Vol. 4630. Lecture Notes in Computer Science. Springer, 2006, pp. 72–83 (cit. on p. 22).

[12] M. P. Deisenroth, G. Neumann, J. Peters. "A Survey on Policy Search for Robotics". In: *Found. Trends Robot* 2.1–2 (Aug. 2013), pp. 1–142. ISSN: 1935-8253. DOI: 10.1561/2300000021. URL: https://doi.org/10.1561/2300000021 (cit. on p. 15).

[13] J. Devlin, M. Chang, K. Lee, K. Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *NAACL-HLT (1)*. Association for Computational Linguistics, 2019, pp. 4171–4186 (cit. on p. 16).

[14] D. Driess, O. Oguz, J. .-.-S. Ha, M. Toussaint. “Deep Visual Heuristics: Learning Feasibility of Mixed-Integer Programs for Manipulation Planning”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 9563–9569 (cit. on p. 16).

[15] D. Drieß, J. Ha, M. Toussaint. “Deep Visual Reasoning: Learning to Predict Action Sequences for Task and Motion Planning from an Initial Scene Image”. In: *CoRR* abs/2006.05398 (2020) (cit. on p. 16).

[16] D. Driess, O. S. Oguz, J.-S. Ha, 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 (cit. on pp. 33, 42).

[17] D. Driess, O. S. Oguz, M. Toussaint. “Hierarchical Task and Motion Planning using Logic-Geometric Programming (HLGP)”. In: *RSS Workshop on Robust TAMP* (2019) (cit. on p. 15).

[18] Y. Duan, M. Andrychowicz, B. C. Stadie, J. Ho, J. Schneider, I. Sutskever, P. Abbeel, W. Zaremba. “One-Shot Imitation Learning”. In: *NIPS*. 2017, pp. 1087–1098 (cit. on p. 15).

[19] F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, S. Levine. *Visual Foresight: Model-Based Deep Reinforcement Learning for Vision-Based Robotic Control*. 2018. arXiv: 1812.00568 [cs.RO] (cit. on p. 15).

[20] S. Even. *Graph Algorithms*. 2nd. USA: Cambridge University Press, 2011. ISBN: 0521736536 (cit. on p. 20).

[21] C. Finn, S. Levine. “Deep visual foresight for planning robot motion”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 2786–2793 (cit. on p. 15).

[22] R. Fisher, S. Perkins, A. Walker, E. Wolfart. “Connected component labeling”. In: *IEE International Conference on Intelligent Systems*. 2003, pp. 300–350 (cit. on p. 43).

[23] C. R. Garrett, T. Lozano-Pérez, L. P. Kaelbling. “Ffrob: An efficient heuristic for task and motion planning”. In: *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 179–195 (cit. on p. 13).

[24] R. B. Girshick. “Fast R-CNN”. In: *ICCV*. IEEE Computer Society, 2015, pp. 1440–1448 (cit. on pp. 38, 45).

[25] O. Groth, F. B. Fuchs, I. Posner, A. Vedaldi. “ShapeStacks: Learning Vision-Based Physical Intuition for Generalised Object Stacking”. In: *ECCV (1)*. Vol. 11205. Lecture Notes in Computer Science. Springer, 2018, pp. 724–739 (cit. on pp. 43, 44).

[26] N. Gupta, D. S. Nau. “On the Complexity of Blocks-World Planning”. In: *Artif. Intell.* 56.2-3 (1992), pp. 223–254 (cit. on pp. 3, 14, 41).

[27] P. E. Hart, N. J. Nilsson, B. Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *IEEE Trans. Syst. Sci. Cybern.* 4.2 (1968), pp. 100–107 (cit. on pp. 20, 22, 44, 50).

[28] V. N. Hartmann, O. S. Oguz, D. Driess, M. Toussaint, A. Menges. “Robust Task and Motion Planning for Long-Horizon Architectural Construction Planning”. In: *arXiv:2003.07754 [cs]* (Mar. 17, 2020). arXiv: 2003.07754. URL: http://arxiv.org/abs/2003.07754 (visited on 05/25/2020) (cit. on pp. 15, 35, 40, 42).

[29] K. Hauser, J.-C. Latombe. “Multi-modal motion planning in non-expansive spaces”. In: *The International Journal of Robotics Research* 29.7 (2010), pp. 897–915 (cit. on pp. 13, 40).

[30] K. He, H. Fan, Y. Wu, S. Xie, R. B. Girshick. “Momentum Contrast for Unsupervised Visual Representation Learning”. In: *CoRR* abs/1911.05722 (2019) (cit. on p. 16).

[31] K. He, G. Gkioxari, P. Dollár, R. B. Girshick. “Mask R-CNN”. In: *ICCV*. IEEE Computer Society, 2017, pp. 2980–2988 (cit. on pp. 38, 45).

[32] K. He, X. Zhang, S. Ren, J. Sun. “Deep Residual Learning for Image Recognition”. In: *CVPR*. IEEE Computer Society, 2016, pp. 770–778 (cit. on pp. 38, 51).

[33] G. E. Hinton, R. R. Salakhutdinov. “Reducing the Dimensionality of Data with Neural Networks”. In: *Science* 313.5786 (2006), pp. 504–507. ISSN: 0036-8075. DOI: `10.1126/science.1127647`. eprint: `https://science.sciencemag.org/content/313/5786/504.full.pdf`. URL: `https://science.sciencemag.org/content/313/5786/504` (cit. on pp. 13, 18).

[34] D. Huang, D. Xu, Y. Zhu, A. Garg, S. Savarese, L. Fei-Fei, J. C. Niebles. “Continuous Relaxation of Symbolic Planner for One-Shot Imitation Learning”. In: *IROS*. IEEE, 2019, pp. 2635–2642 (cit. on pp. 13, 15).

[35] P. Isola, J.-Y. Zhu, T. Zhou, A. A. Efros. “Image-to-Image Translation with Conditional Adversarial Networks”. In: *CVPR* (2017) (cit. on p. 47).

[36] M. Jaderberg, K. Simonyan, A. Zisserman, K. Kavukcuoglu. “Spatial Transformer Networks”. In: *NIPS*. 2015, pp. 2017–2025 (cit. on pp. 18, 38).

[37] E. Jang, S. Gu, B. Poole. “Categorical Reparameterization with Gumbel-Softmax”. In: *ICLR (Poster)*. OpenReview.net, 2017 (cit. on pp. 13, 15, 18).

[38] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, R. B. Girshick. “CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning”. In: *CVPR*. IEEE Computer Society, 2017, pp. 1988–1997 (cit. on pp. 14, 27, 29, 32, 48).

[39] J. Johnson, R. Krishna, M. Stark, L. Li, D. A. Shamma, M. S. Bernstein, F. Li. “Image retrieval using scene graphs”. In: *CVPR*. IEEE Computer Society, 2015, pp. 3668–3678 (cit. on pp. 14, 20).

[40] S. W. Kim, Y. Zhou, J. Philion, A. Torralba, S. Fidler. “Learning to Simulate Dynamic Environments with GameGAN”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020 (cit. on pp. 47, 51).

[41] D. P. Kingma, J. Ba. “Adam: A Method for Stochastic Optimization”. In: *ICLR (Poster)*. 2015 (cit. on pp. 20, 51).

[42] D. P. Kingma, M. Welling. “Auto-Encoding Variational Bayes”. In: *ICLR*. 2014 (cit. on pp. 13, 18).

[43] Z. Kingston, M. Moll, L. E. Kavraki. “Sampling-Based Methods for Motion Planning with Constraints”. In: *Annual Review of Control, Robotics, and Autonomous Systems* (2018) (cit. on pp. 13, 40).

[44] T. N. Kipf, E. van der Pol, M. Welling. “Contrastive Learning of Structured World Models”. In: *ICLR*. OpenReview.net, 2020 (cit. on pp. 28, 29).

[45] A. Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009 (cit. on p. 18).

[46] F. Lagriffoul, D. Dimitrov, A. Saffiotti, L. Karlsson. "Constraint propagation on interval bounds for dealing with geometric backtracking". In: *IROS*. IEEE, 2012, pp. 957–964 (cit. on p. 16).

[47] Z. Lin, Y.-F. Wu, S. Peri, B. Fu, J. Jiang, S. Ahn. *Improving Generative Imagination in Object-Centric World Models*. 2020. arXiv: `2010.02054 [cs.LG]` (cit. on p. 47).

[48] J. D. C. Little, K. G. Murty, D. W. Sweeney, C. Karel. "An Algorithm for the Traveling Salesman Problem". In: *Oper. Res.* 11.6 (Dec. 1963), pp. 972–989. ISSN: 0030-364X. DOI: `10.1287/opre.11.6.972`. URL: `https://doi.org/10.1287/opre.11.6.972` (cit. on p. 22).

[49] T. Lozano-Pérez, 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*. IEEE, 2014, pp. 3684–3691 (cit. on pp. 13, 16).

[50] T. Lozano-Pérez, L. P. Kaelbling. "A constraint-based method for solving sequential manipulation planning problems". In: *IROS*. IEEE, 2014, pp. 3684–3691 (cit. on p. 21).

[51] C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, P. Sermanet. "Learning Latent Plans from Play". In: *CoRL*. Vol. 100. Proceedings of Machine Learning Research. PMLR, 2019, pp. 1113–1132 (cit. on p. 15).

[52] C. J. Maddison, A. Mnih, Y. W. Teh. "The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables". In: *ICLR (Poster)*. OpenReview.net, 2017 (cit. on pp. 13, 15, 18).

[53] A. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, S. Levine. "Visual Reinforcement Learning with Imagined Goals". In: *NeurIPS*. 2018, pp. 9209–9220 (cit. on p. 15).

[54] A. van den Oord, O. Vinyals, K. Kavukcuoglu. "Neural Discrete Representation Learning". In: *NIPS*. 2017, pp. 6306–6315 (cit. on pp. 13, 14, 18, 26, 27, 48).

[55] P. Pastor, H. Hoffmann, T. Asfour, S. Schaal. "Learning and generalization of motor skills by learning from demonstration". In: *ICRA*. IEEE, 2009, pp. 763–768 (cit. on p. 15).

[56] 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, S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *NeurIPS*. 2019, pp. 8024–8035 (cit. on p. 38).

[57] R. Rahmatizadeh, P. Abolghasemi, L. Bölöni, S. Levine. "Vision-Based Multi-Task Manipulation for Inexpensive Robots Using End-to-End Learning from Demonstration". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 3758–3765 (cit. on p. 15).

[58] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, S. Levine. "Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations". In: *Robotics: Science and Systems*. 2018 (cit. on p. 15).

[59] S. Ren, K. He, R. B. Girshick, J. Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *NIPS*. 2015, pp. 91–99 (cit. on pp. 38, 45).

[60] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. D. Reid, S. Savarese. "Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Regression". In: *CVPR*. Computer Vision Foundation / IEEE, 2019, pp. 658–666 (cit. on p. 41).

[61] D. E. Rumelhart, G. E. Hinton, R. J. Williams. "Learning Representations by Back-Propagating Errors". In: *Neurocomputing: Foundations of Research*. Cambridge, MA, USA: MIT Press, 1988, pp. 696–699. ISBN: 0262010976 (cit. on p. 19).

[62] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, L. Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y (cit. on pp. 18, 19, 38).

[63] A. Santoro, D. Raposo, D. G. T. Barrett, M. Malinowski, R. Pascanu, P. W. Battaglia, T. Lillicrap. "A simple neural network module for relational reasoning". In: *NIPS*. 2017, pp. 4967–4976 (cit. on pp. 27, 48).

[64] V. Sitzmann, M. Zollhöfer, G. Wetzstein. "Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations". In: *NeurIPS*. 2019, pp. 1119–1130 (cit. on pp. 48, 54).

[65] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, P. Abbeel. "Combined task and motion planning through an extensible planner-independent interface layer". In: *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2014, pp. 639–646 (cit. on pp. 13, 16).

[66] M. Toussaint. "A Tutorial on Newton Methods for Constrained Trajectory Optimization and Relations to SLAM, Gaussian Process Smoothing, Optimal Control, and Probabilistic Inference". In: *Geometric and Numerical Foundations of Movements*. Ed. by J.-P. Laumond, N. Mansard, J.-B. Lasserre. Cham: Springer International Publishing, 2017, pp. 361–392. ISBN: 978-3-319-51547-2. DOI: 10.1007/978-3-319-51547-2_15. URL: https://doi.org/10.1007/978-3-319-51547-2_15 (cit. on pp. 16, 22, 23, 40).

[67] M. Toussaint. "Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning". In: *IJCAI*. AAAI Press, 2015, pp. 1930–1936 (cit. on pp. 13, 15, 21, 22, 33).

[68] M. Toussaint, K. R. Allen, K. A. Smith, J. B. Tenenbaum. "Differentiable Physics and Stable Modes for Tool-Use and Manipulation Planning". In: *Robotics: Science and Systems*. 2018 (cit. on pp. 13, 14, 21, 33, 35, 48).

[69] M. Toussaint, M. Lopes. "Multi-bound tree search for logic-geometric programming in cooperative manipulation domains". In: *ICRA*. IEEE, 2017, pp. 4044–4051 (cit. on pp. 16, 21, 22, 33).

[70] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P. Manzagol. "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion". In: *J. Mach. Learn. Res.* 11 (2010), pp. 3371–3408 (cit. on pp. 13, 18).

[71] G. Yang, A. Zhang, A. S. Morcos, J. Pineau, P. Abbeel, R. Calandra. "Plan2Vec: Unsupervised Representation Learning by Latent Plans". In: *CoRR* abs/2005.03648 (2020) (cit. on p. 16).

[72] S. Zhao, W. Jakob, T.-M. Li. "Physics-Based Differentiable Rendering: From Theory to Implementation". In: *ACM SIGGRAPH 2020 Courses*. SIGGRAPH 2020. Virtual Event, USA: Association for Computing Machinery, 2020. ISBN: 9781450379724. DOI: 10.1145/3388769.3407454. URL: https://doi.org/10.1145/3388769.3407454 (cit. on p. 48).

**Declaration**

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature