

Infrastructure for Modeling and Inference Engineering with 3D Generative Scene Graphs

by

Austin J. Garrett

Department of Electrical Engineering and Computer Science
Proposal for Thesis Research in partial fulfillment of the requirements
for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2021

© Massachusetts Institute of Technology 2021. All rights reserved.

Author
Department of Electrical Engineering and Computer Science
January 25, 2021

Certified by
Vikash K. Mansinghka
Principal Research Scientist
Thesis Supervisor

Accepted by
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Infrastructure for Modeling and Inference Engineering with 3D Generative Scene Graphs

by

Austin J. Garrett

Submitted to the Department of Electrical Engineering and Computer Science
on January 25, 2021, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Recent advances in probabilistic programming have enabled the development of probabilistic generative models for visual perception using a rich abstract representation of 3D scene geometry called a scene graph. However, there remain several challenges in the practical implementation of scene graph models, including human-editable specification, visualization, priors, structure inference, hyperparameters tuning, and benchmarking. In this thesis, I describe the development of infrastructure to enable the development and research of scene graph models by researchers and practitioners. A description of a preliminary scene graph model and inference program for 3D scene structure is provided, along with an implementation in the probabilistic programming language Gen. Utilities for visualizing and understanding distributions over scene graphs are developed. Synthetic enumerative tests of the posterior and inference algorithm are conducted, and conclusions drawn for the improvement of the proposed modeling components. Finally, I collect and analyze real-world scene graph data, and use it to optimize model hyperparameters; the preliminary structure inference program is then tested in a structure prediction task with both the unoptimized and optimized models.

Thesis Supervisor: Vikash K. Mansinghka
Title: Principal Research Scientist

Acknowledgments

This work was conducted with the support of an amazing team of collaborators and mentors, and would not be possible without them. I am grateful to Marco Cusumano-Towner for encouraging and sharpening my exploration of deep technical questions. His genuine curiosity and passion for his work is an inspiration to always keep digging at the hard technical problems, and to always strive for a deeper understanding. I thank my adviser Vikash Mansinghka for his guidance in my journey through this program. I greatly appreciate his dedication to finding and encouraging the good in his students and collaborators, and the positive leadership example he sets by doing so. He has patiently helped me to see the value in my own work, and as such led me to gain a better appreciation for what it actually takes to do research.

My Master's would not have been the same Ben Zinberg; he has been a consistent bright light through the dark spots. I appreciate him for his collaboration on the work in this thesis, his humor, and above all, his friendship. Rachel Paiste has been a consistently grounding presence in the ever changing sea that has been the COVID pandemic. Her persistent support in the more challenging of times has been an invaluable source of stability, for which I am incredibly grateful. I'd like to thank Nishad Gothiskar, Falk Pollok, Aldo Pareja, and Javier Felipe Leon for their collaboration on and work on the project leading up to this thesis. Many others members and affiliates of MIT's aided in the development of this thesis through discussions and interactions; I'd like to thank Alex Lew and Tan Zhi-Xuan for growing my understanding of the deep field of probabilistic programming, Amanda Brower for her support, and Nick Charchut for being a bud.

Finally, none of this would've been possible without my parents David and Laura Garrett, who taught me genuinity of heart and sacrificed so much to help me succeed. Because of them, I am free to do what I love.

This research was funded by the DARPA Machine Common Sense program, under (sub)contract CW3013540.

Contents

1	Introduction	15
1.1	Modeling with generative scene graphs	16
1.2	Summary of this thesis	16
2	Scene Graphs	17
2.1	A scene graph model	17
2.1.1	Computing the 6DoF poses of all objects in the scene	18
2.1.2	Modeling face-to-face contact between two objects	18
2.1.3	A prior distribution on scene graphs	19
2.1.4	Probabilistic dynamics on scene graphs	20
2.1.5	Robust pose likelihood for noisy object detections	22
2.1.6	Representation as a generative program	23
2.2	Inference over scene structure	23
2.3	Reversible jump moves	24
2.3.1	Representation as an inference program	25
2.4	Example application domains of scene graphs	28
2.4.1	YCB objects on a synthetic tabletop	29
2.4.2	Real YCB objects on a physical tabletop	30
3	Visualizing Scene Graphs	33
3.1	Desiderata	33
3.2	Examples	34
3.2.1	Visualizing a single scene graph	35

3.2.2	Distributions over structure beliefs	36
3.2.3	Distributions over scene graphs	38
3.2.4	Visualizing inference in a particle filter	39
4	Analysis of Posteriors and Inference by Enumeration	41
4.1	Enumerative inference	41
4.2	Structure posterior	42
4.2.1	Static model	43
4.2.2	Dynamic model	44
4.2.3	RJMCMC structure inference	46
4.3	Contact slack model	46
4.3.1	Varying the gap between sliding objects	46
4.3.2	Hyperprior over slack standard deviation	47
4.4	Neural outlier model	49
4.5	Lessons for improving scene graph priors	52
5	Learning and Testing from Real-World Data	53
5.1	Data specification	53
5.2	Metrics	56
5.2.1	Modeling: marginal likelihood	56
5.2.2	Structure inference: graph edit distance	56
5.3	Learning hyperparameters with gradient ascent	57
5.4	Inlier detection observational model	59
5.5	Testing structure inference	61
6	Conclusion	65
6.1	Future directions	65

List of Figures

2-1 Our random walk transition model on the structure of scene graphs modifies the tree that defines the scene graph structure. Left: A transition from structure G' to G that makes o_5 a child of the root r instead of object o_3 . In G , parametrization of the pose of o_5 was previously parametrized relative to the pose of object o_3 ; in G' it is parametrized independently of the pose of the other objects. Right: The transition on structures is reversible, and the reverse transition is unique. This allows us to use the transition model on structure as the basis of reversible jump MCMC moves on structure, as well within the dynamics model on structure.	21
2-2 Abstract representation of our scene graph model; image data is parsed into noisy unstructured neural detections, which are modeled as observations of an underlying structured scene graph.	22
2-3 Probabilistic pseudocode for an example dynamic scene graph model	26
2-4 Custom implementation of RJMCMC algorithm for structure inference	27
2-5 Top-level inference procedure for particle filtering	28
2-6 Synthetic scenes generated using the GenSceneGraphs library in simulated dim light. The colored wireframe bounding boxes represent objects in the scene graph, and the white boxes represent contact edges (see Chapter 3 for more detail on how this visualization was produced).	29

2-7 Example data from real-world scenes containing objects from the YCB dataset. (a) and (c) show the underlying latent scene graph. (b) and (d) show observed inaccurate pose estimates generated from the nVidia DOPE detector. (b) in particular demonstrates the types of catastrophic detection and localization errors that deep neural networks can exhibit.	30
3-1 Visualization of a synthetic scene with its abstract scene graph overlaid. The Soft Scrub bottle is above the table, so doesn't have an edge.	34
3-2 Visualization of a distribution over abstract scene graph structure using Graphviz.	36
3-3 Various methods for visualizing multiple beliefs. Node labels are omitted in the rendered structure distribution, and instead the wireframe rendering and GraphViz node for an object o share the same color.	37
3-4 Visualization of sampling and rejuvenation moves in a particle filter. A complete video of this scene can be seen at: https://www.youtube.com/watch?v=0_0TvrGC65Q . (i) shows the first time step with the overlaid beliefs from the initial time step prior (initialized to the observed neural pose estimates), before structure inference. (ii) shows the beliefs over the first time step, after running 1 step of the rejuvenation kernel. (iii) shows the beliefs over the second time step, after sampling from the dynamics model. (iv) shows the beliefs over the second time step, after running 20 steps of the rejuvenation kernel.	39
4-1 Probability that an object is sliding in a static scene with different settings of σ_{slack} , and observed/latent vertical offset y	42
4-2 Probability that the scene structure G_t is in a “sliding” configuration at time t , in a dynamic scene where the object’s observed position and slack offset is given by $y(t) = 10 - 0.67 \cdot t$ cm, with hyperparameters $(\sigma_{\text{floating}}, \sigma_{\text{sliding}})$	44

4-3	RJMCMC structure inference results. For each value of d , the approximate probability of the top object sliding is inferred as the average structure of 100 particles run with 3 iterations of the reversible jump moves applied to the top object.	45
4-4	Posterior distribution on slack offset d . The dashed lines represent the set of observed y values, while the curve of the same color represents the slack offset posterior, given the observed y and sliding structure.	47
4-5	Joint posterior $p(d, \sigma_{\text{slack}} y = 5.0)$. The dashed red line indicates the observed position $y = 5.0$ of the top object. The blue star indicates the maximum a posteriori $(\sigma_{\text{slack}}^*, d^*)$	48
4-6	Slack offset posterior for d , conditioned on $\sigma_{\text{slack}} = \sigma_{\text{slack}}^*$. The dashed lines represent the set of observed y values, while the curve of the same color represents the posterior over slack given the observed y and sliding structure.	48
4-7	Posteriors over the latent pose of an object $p(x_2 y_2) = p(x_2 y_2, x_1 = 0)$, under two different settings of the dynamics parameter. The noisy observational model's inlier distribution is given by $\sigma_{\text{inlier}} = 1$ cm, while the prior probability of an outlier is $p_{\text{outlier}} = 0.1$. The dashed line represents the observed pose y_2 , with color representing the probability that y_2 is an outlier (totally green means $p_{\text{inlier}} = 1$, while totally red means $p_{\text{outlier}} = 1$).	51
5-1	A selection of frames from 5 of the 8 captured videos. (Left) We use a dynamic occluder to generate noise in the observed neural network detections, in which objects have their poses estimated incorrectly, or even flicker out of existence. (Right) Each scene is represented by a single static scene graph, with manually annotated structure, and object poses estimated as the average of the inlier neural detections.	55

5-2 Results of gradient ascent on σ . (Top) shows the change in log likelihood $\Delta\ell(\mathcal{T}; \sigma) = \ell(\mathcal{T}; \sigma_t) - \ell(\mathcal{T}; \sigma_0)$. (Bottom) shows the values of the parameters σ_t .	58
5-3 Scatter plot of the projection of nVidia DOPE’s observed pose estimates to the x and z position axes, along with corresponding marginal histograms for each axis respectively. The plotted contour is the overlaid noisy observation model for inlier detections (in the projection, a 2D multivariate normal distribution), with the standard deviation hyperparameter set to the maximum-likelihood value for σ_{slack} that we optimized for above.	59
5-4 Example of the performance on RJMCMC on inferring the structure posterior in a frame from scene #5. (Top) The uniform prior over scene graph structure contains a random assortment of inaccurate classifications. (Bottom) The RJMCMC structure inference procedure predicts the correct ground-truth structure with high probability.	61
5-5 Graph edit distance between the ground truth structure G and the mode \hat{G}^* of the inferred structure posterior, for the unoptimized model (top) and optimized model (bottom).	63

List of Tables

5.1 Description of data collected in our real-world experiments with YCB objects.	54
---	----

Chapter 1

Introduction

Scene graphs have recently emerged as a proposed unifying representation for 3D scenes, that allows for structured constraints on the relative geometric poses of objects [19, 14, 7]. This approach builds on top of the probabilistic programming paradigm in Bayesian statistics, which has been increasingly common in computer vision applications [1, 5, 6, 8, 11, 17, 18, 15]. However, with the introduction of this rich new representation, comes the introduction of several challenges to the practical implementation of scene graph models. Among these challenges are: human-editable specification, visualization, priors, structure inference, hyperparameters tuning, and benchmarking. This thesis describes the development of engineering infrastructure aimed at addressing these challenges, to enable robust exploration of scene graph modeling and inference in a principled way.

1.1 Modeling with generative scene graphs

Scene graphs are a scalable representation framework for 3D scenes, with a long history in computer graphics. They have been increasingly used as a unifying representation for a naturalistic combined structure for communicating, rendering, execution, and modeling in 3D graphical programs [16]. Several approaches to visual perception have proposed that vision operates as an inverse graphics pipeline, inferring the latent state of a scene from an observed generated rendered image [15, 9, 13]. Scene graph models are a natural extension of this line of research, extending modeling to more richly structured relative relations between objects.

1.2 Summary of this thesis

The methods employed in this thesis are implemented in the probabilistic programming system **Gen** [2]. Gen differs from other probabilistic programming systems, by emphasizing user freedom and flexibility of implementation over automation of inference, and providing users with an interface for specifying custom inference procedures defined on those models. This flexibility enables the development of powerful new tools for working with complex generative programs, like those based on scene graphs.

Chapter 2 specifies the scene graph data type and a reference model, as well as an inference procedure for estimating the underlying geometry of a scene. It additionally provides an implementation of this model and inference procedure in Gen. Chapter 3 describes methods for visualizing and understanding distributions over the scene graph representation, which is part of the vital infrastructure for working with probabilistic models that leverage it. Chapter 4 describes a set of constructed synthetic test cases that allow us to analyze, test, and ultimately improve the behavior of scene graph models and inference procedure. Chapter 5 leverages real-world scenes containing YCB objects on a physical tabletop to benchmark modeling and structure inference, and to tune scene graph model hyperparameters on real data.

Chapter 2

Scene Graphs

2.1 A scene graph model

We model the geometric state of a scene at a single time point as a *scene graph*, which is a tuple (G, Θ, Z) , where $G = (V, E)$ is an undirected graph that encodes the scene graph *structure* and Θ encodes the scene graph *continuous parameters* and Z encodes the scene graph *discrete parameters*. Although the framework we present can represent articulated objects, in this document we only consider scenes involving a set of rigid objects O . The scene graph structure includes a vertex $v_o \in V$ for each object $o \in O$ that represents the 6DoF pose of object o , as well as a vertex $r \in V$ that represents the coordinate frame of the observer. An edge $e = (v_i, v_j) \in E$ between objects $i, j \in O$ encodes that the pose of object j is parametrized *relative to* the pose of object i , by parameters θ_e and z_e . The pose of object j relative to object i is denoted $\Delta x(z_e, \theta_e) \in SE(3)$. An edge $e = (r, v_j) \in E$ from the root to an object $j \in O$ encodes that the pose of object j is not parametrized relative to that of any object, but is instead a full 6DoF pose $\theta_e \in SE(3)$ where $SE(3)$ is the special Euclidean group consisting of all 6DoF poses. The continuous and discrete parameters of the scene graph consist of the parameters for each edge in the structure ($\Theta := \{\theta_e\}_{e \in E}$ and $Z := \{z_e\}_{e \in E}$). The set of edges E must *span* the set of vertices $V := \{r\} \cup \{v_o\}_{o \in O}$, and be rooted at r ; this condition makes the undirected tree isomorphic to the *directed forest* constructed by removing the world node r .

2.1.1 Computing the 6DoF poses of all objects in the scene

Given a scene graph (G, Θ, Z) the pose $x_i \in SE(3)$ of an object $i \in O$ relative to the observer coordinate frame can be computed by walking path in the tree G from the root vertex $r \in V$ to the vertex $v_i \in V$, and successively computing the pose of each object along the path from the pose of its parent (the set of all such poses is $X := \{x_i\}_{i \in O}$). That is, given pose $x_u \in SE(3)$ and edge $(u, v) \in E$, the pose $x_v \in SE(3)$ is computed as $x_v := x_u \cdot \Delta x(z_e, \theta_e)$ where $\Delta x(z_e, \theta_e) \in SE(3)$ is the relative pose between vertex u and vertex v . For an edge from the root vertex to an object vertex ($e = (r, v_i)$), $x_u := \mathbf{1}$ (the identity element of $SE(3)$) and $\Delta x(z_e, \theta_e) = \theta_e \in SE(3)$ (note that there are no discrete parameters in this case, so $z_e := ()$). The functional form of $\Delta x(z_e, \theta_e)$ for an edge e between two objects is discussed below. One requirement is that $\Delta x(z_e, \theta_e)$ is a differentiable function of θ_e ; this enables inference algorithms that exploit gradient information.

2.1.2 Modeling face-to-face contact between two objects

An edge $(v_i, v_j) \in E$ from object i to object j indicates that the pose of object j is represented relative to the pose of object i . Various types of relative pose parametrizations for two objects are possible; for simplicity we model objects as polyhedra, and only model face-to-face contact between objects. That is, an edge $e = (v_i, v_j)$ indicates that a face of object i is in contact with a face of object j . Since each object has multiple faces, the choice of which pair of faces is in contact is encoded in the discrete parameters z_e for the edge. Concretely, let F_i and F_j denote the faces of object i and object j respectively. Then, $z_e \in F_i \times F_j$. The continuous parameters θ_e for an edge e between two objects is an element of $\mathbb{R}^2 \times [0, 2\pi)$ that contains two translational degrees of freedom ($x, y \in \mathbb{R}$, for the relative offset of the two faces) and one rotational degree of freedom ($\phi \in [0, 2\pi)$). For example, a cuboid object $i \in O$, the set of faces is $F_i = \{\text{Top}, \text{Bottom}, \text{Left}, \text{Right}, \text{Front}, \text{Back}\}$. For an edge $e = (i, j)$ where objects $i, j \in O$ where both objects i and j are cuboids, there are 36 possible values for z_e .

Slack variables for face-to-face contact We extend the parametrization of face-to-face contact between objects with three additional degrees of freedom of *slack variables*: (i) one degree of freedom that encodes the perpendicular distance ($d \in [0, \infty)$) between the two contact faces, and (ii) two degrees of freedom for relative orientation of the two faces, encoded the surface normal unit vector \mathbf{n} of the child object’s face, relative to the parent object’s face, which takes values on the sphere S^2 . Therefore, $\theta_e \in \mathbb{R}^2 \times [0, 2\pi) \times [0, \infty) \times S^2$ for an edge $e = (v_i, v_j)$ between two objects i and j . Note that although this edge parametrization uses six degrees of freedom for object-to-object edges like the edge parametrization for edges from the root ($e = (r, v_j)$), the prior distribution on these parameters (described below) encourages object j to be *almost* in face-to-face contact with object i ; whereas the prior distribution on the pose of object j for an edge of the form (r, v_j) is very different, and is typically a uniform distribution over positions within the scene bounding volume, and a uniform distribution on orientations.

2.1.3 A prior distribution on scene graphs

Various prior distributions on scene graphs are possible. In our experiments, we use a generic prior distribution on scene graphs over a collection of objects O that factors into two components: (i) a prior distribution on scene graph structures G , denoted $p(G)$, and (ii) a prior distribution on scene graph parameters (Z, Θ) given structure, denoted $p(Z, \Theta|G)$. While uncertainty about the number of objects is possible to represent in our framework and implementation, for simplicity assume that the set of objects is known a-priori, and that there is one vertex for each object, and one vertex representing the observer coordinate frame, so V is fixed a-priori to $\{r\} \cup \{v_o\}_{o \in O}$. Therefore, $p(G)$ reduces to a prior distribution on edges in the scene graph. For the prior distribution on structure, we use a uniform distribution on undirected trees that are rooted at vertex r and span all $|O| + 1$ vertices in the graph. Therefore, the prior probability of a graph $G = (V, E)$ is obtained by using Cayley’s

formula to count the number of undirected trees on $|O| + 1$ vertices:

$$p(G) := p_{\text{unif}(|O|)}(G) := \begin{cases} (|O| + 1)^{1-|O|} & \text{if } G \text{ is a undirected tree over vertices } V \text{ rooted at } r \\ 0 & \text{otherwise} \end{cases} \quad (2.1)$$

The prior distribution on scene graph parameters factors over the edges in the scene graph:

$$p(Z, \Theta | G) := \prod_{e \in E} p_e(z_e, \theta_e) \quad (2.2)$$

where $p_e(z_e, \theta_e)$ is a probability distribution on z_e and θ_e that depends on the two vertices in the edge $e = (u, v)$. For edges $e = (r, v_j)$ where j is an object, $z_e = ()$ and $p_e(z_e, \theta_e)$ is the uniform distribution on elements of $B \times SO(3)$ where B is the bounding volume of the scene and $SO(3)$ is 3D rotation group (the uniform distribution on $SO(3)$ is given by the Haar measure). For edges $e = (v_i, v_j)$ where i and j are objects, recall that $z_e \in F_i \times F_j$ and (using the edge parametrization including slack variables) $\theta_e = (x, y, \phi, d, \alpha_1, \alpha_2) \in \mathbb{R}^2 \times [0, 2\pi) \times [0, \infty) \times [0, 2\pi]^2$. The prior distribution on edge parameters is:

$$p_e(z_e, \theta_e) := \frac{1}{|F_i||F_j|} \cdot p_{\text{norm}(0, \sigma_{\text{slidingXY}})}(x) \cdot p_{\text{norm}(0, \sigma)}(y) \cdot \frac{1}{2\pi} \cdot p_{\text{norm}(0, \sigma_{\text{slack}})(\beta)}(d) \quad (2.3)$$

where p_{norm} is the normal distribution density function.

2.1.4 Probabilistic dynamics on scene graphs

We build temporal models of scene geometry based on Markov chains of scene graphs $\{(G_t, \Theta_t, Z_t)\}_{t=1}^T$, where t indexes time, with transitions $p(G_t, \Theta_t, Z_t | G_{t-1}, \Theta_{t-1}, Z_{t-1})$. We factor the dynamics model on scene graphs decomposes into a dynamics model on scene graph structure, and a dynamics on parameters:

$$p(G_t, \Theta_t, Z_t | G_{t-1}, \Theta_{t-1}, Z_{t-1}) := p(G_t | G_{t-1}) \cdot p(Z_t, \Theta_t, | Z_{t-1}, \Theta_{t-1}, G_{t-1}, G_t) \quad (2.4)$$

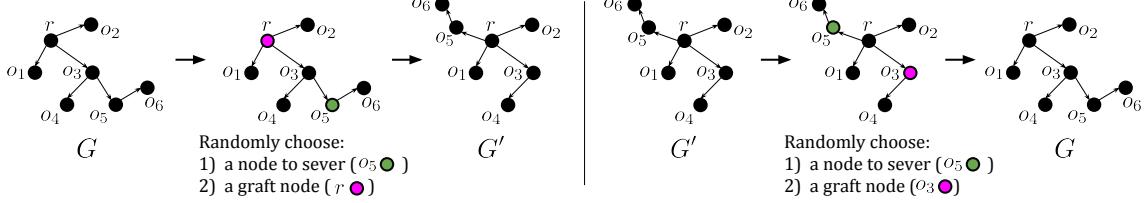


Figure 2-1: Our random walk transition model on the structure of scene graphs modifies the tree that defines the scene graph structure. Left: A transition from structure G' to G that makes o_5 a child of the root r instead of object o_3 . In G , parametrization of the pose of o_5 was previously parametrized relative to the pose of object o_3 ; in G' it is parametrized independently of the pose of the other objects. Right: The transition on structures is reversible, and the reverse transition is unique. This allows us to use the transition model on structure as the basis of reversible jump MCMC moves on structure, as well within the dynamics model on structure.

The dynamics on graph structure is based on a mixture of a random walk on structures (to capture incremental changes to the structure of a scene that often occur) and a uniform distribution on structures (to model sudden unexpected changes in structure):

$$p(G_t|G_{t-1}) := 0.9 \cdot p_{\text{walk}}(G_t|G_{t-1}) + 0.1 \cdot p_{\text{unif}(|O|)}(G_t) \quad (2.5)$$

where $p_{\text{walk}}(G'|G)$ is the distribution on graph structures induced by a sampling process in which one vertex in the undirected tree is selected at random, severed from the tree, and grafted back onto the graph at a uniformly chosen vertex, subject to the condition that the resulting graph is a tree; Figure 2-1 shows an example of this process. Recall that since the tree contains the root node r , this process can change the edge type (u, v_j) for an object i from (r, v_j) (representing independent 6DoF pose) to (v_i, v_j) (representing flush contact pose with another object v_i). The dynamics on parameters factor according to edges in the graph:

$$p(Z_t, \Theta_t | Z_{t-1}, \Theta_{t-1}, G_{t-1}, G_t) := \prod_{e \in E} p_e(z_e, \theta_e | G_t, G_{t-1}, z_{t-1e}, \theta_{t-1e}) \quad (2.6)$$

If an edge e is present in both G_t and G_{t-1} , then this distribution is a mixture of a random walk and a uniform distribution. If an edge e in G_t was not present in G_{t-1} , then this distribution is a uniform distribution.

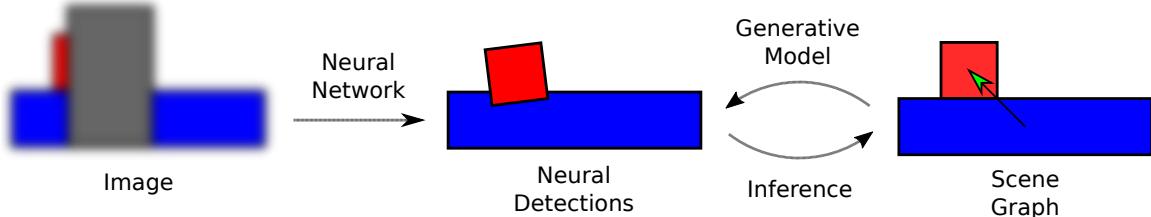


Figure 2-2: Abstract representation of our scene graph model; image data is parsed into noisy unstructured neural detections, which are modeled as observations of an underlying structured scene graph.

2.1.5 Robust pose likelihood for noisy object detections

Our model differs from previous inverse graphics work in an important way; instead of creating a full generative model all the way to pixel-level data, we instead model the output of a bottom-up feature detector that estimates the poses of objects from visual data. This was specifically motivated by the desire to leverage the power of existing deep learning approaches to object detection, and integrate them with a Bayesian approach. We do not justify the relative advantages of this approach in this thesis, but it is an important feature of the model that has implications for the forms of tests that are conducted later in the document. Figure 2-2 shows the abstract pipeline for our model. Our generative model is thus grounded in attempting to explain the output of a noisy detector, which requires us to accurately predict the presence and distribution of “outlier” detections. In practice, the failure modes of deep neural networks are complex and varied, and can depend on a huge range of factors, including occlusion, lighting condition, the presence of novel objects, etc. We provide a simple first-pass likelihood model for representing noisy bottom-up detectors, and note that developing more accurate likelihoods in scene graph models is a rich avenue for future work in this area.

We introduce a likelihood over observed object pose detections $Y := \{y_o\}_{o \in O} = \{(y_o^{\text{pos}}, y_o^{\text{rot}})\}_{o \in O}$. The likelihood consists of two components in a mixture model: (i) a uniform distribution over a finite cuboid region in 3D space, and (ii) a product of a diagonal multivariate Gaussian over object positions, and a von Mises–Fisher distribution over rotations, represented as unit vectors in S^3 (quaternions). The

first component represents the possibility of outlier neural “flicker”, where objects may be properly recognized as being in the scene, but are very poorly localized. The second component represents where the inlier cases where the neural detector roughly captures the correct location of the object in the scene, with small local inaccuracies.

$$p(Y|X, \sigma, \kappa) = \prod_{o \in O} \left(\frac{p_{\text{outlier}}}{LWH} + (1 - p_{\text{outlier}}) \cdot p_{\text{norm}(x_o^{\text{pos}}, \sigma_{\text{inlier}})}(y_o^{\text{pos}}) \cdot p_{\text{vmf}(x_o^{\text{rot}}, \kappa_{\text{inlier}})}(y_o^{\text{rot}}) \right) \quad (2.7)$$

2.1.6 Representation as a generative program

We implement our scene graph model in the Gen probabilistic programming system [2]. We leverage the dynamic DSL provided within Gen to implement the different modeling components as modular generative functions. Figure 2-3 shows example code of a probabilistic program that implements our scene graph model.

2.2 Inference over scene structure

When considering how to infer the posterior distribution over scene graph structure G , we might take inspiration from our graph random walk distribution $p_{\text{walk}}(G_t|G_{t_1})$ to build an incremental graph modification kernel that proposes small changes to the structure. However, when changing an edge type from $e \rightarrow e'$, we also induce a new set of parameters $\theta_{e'}$ that must be simultaneously proposed or resimulated. Ultimately these two parameterizations are just different ways to express identical absolute poses $x_j = x'_j$, as either relative to another object, or to the observer coordinate frame.

Reversible jump MCMC (RJMCMC) is a generalization of the MH algorithm that allows moves between models with different state spaces, while reusing information in the different model latents to accelerate inference. We provide the formulation of RJMCMC here without proof of asymptotic correctness; for a full derivation see Green et. al [4]. We define moves over a projection of the scene graph model to the static case; the dynamic case can easily be extended with an additional parameter for time.

2.3 Reversible jump moves

Define a set of models $k \in \mathcal{K}$, each with latent variables $\theta_k \in \mathbb{R}^{N_k}$ and joint density $p_k(k, \theta_k, \mathcal{D})$, as well as a set of reversible jump moves $m \in \mathcal{M}$, each of which switches between two models $k, k' \in \mathcal{K}$. For each model, we have a distribution over possible moves $q_k(m)$ with support over a subset of \mathcal{M} . Each reversible jump move from $k \rightarrow k'$ proposes a value for $\theta_{k'}$ given θ_k , using a differentiable bijection f . However, in general $N_k \neq N_{k'}$, which prevents us from making a bijection between these two spaces. To ameliorate this, we pad the latent spaces of both models with additional auxiliary randomness. In the forward direction ($k \rightarrow k'$), we sample $u \sim q(\cdot)$ where $u \in \mathbb{R}^{D_k}$, and in the backward direction ($k' \rightarrow k$), we sample $u \sim q'(\cdot)$ where $u \in \mathbb{R}^{D_{k'}}$; we additionally require $N_k + D_k = N_{k'} + D_{k'}$ such that we can define a bijection between our two extended spaces. We then let the bijection be between these two extended state spaces; the forward move then proposes new values $(\theta_{k'}, u') = f(\theta_k, u)$, which we accept with probability $\min\{1, \alpha\}$, where

$$\alpha = \frac{p_{k'}(k', \theta_{k'}, \mathcal{D})q(u)q_{k'}(m)}{p_k(k, \theta_k, \mathcal{D})q'(u')q_k(m)} \cdot |\det J_f| \quad (2.8)$$

Because our different parameterizations ultimately represent the same absolute poses, RJMCMC offers a way to reuse our inferred continuous parameters θ_e to efficiently propose values for $\theta_{e'}$. Explicitly, we define a class of reversible jump moves parameterized by $j \in V$ that determines which node will have a new parent proposed. If j is floating in $G = (V, E)$, then the models reachable have structure $(V, E \setminus \{(r, j)\} \cup \{(i, j)\})$, and discrete parameter $z_{(i,j)} = (f_i, f_j)$. The corresponding moves are sampled from the distribution $m_{\text{sliding}} = (i, f_i, f_j) \sim q_{\text{floating}}(\cdot, \cdot, \cdot)$. Inversely, if j is sliding, then the only model reachable has structure $(V, E \setminus \{(i, j)\} \cup \{(r, j)\})$. We call the move corresponding to this m_{floating} , and $q_{\text{sliding}}(m_{\text{floating}}) = 1$. The slack terms mean both floating and sliding continuous parameters have 6DoF, thereby precluding the need for sampling auxiliary continuous random variables; this is the main motivation for explicitly modeling a slack term.

The final component are the bijections. All sliding to floating moves have the

bijection $f_{m_{\text{floating}}}(G, \Theta, Z) = (G', \Theta', Z')$, where $E' = E \setminus \{(i, j)\} \cup \{(r, j)\}$, $z'_{(i,j)} = ()$, and $\theta_{(i,j)}$ is calculated using the method described in 2.1.1. All floating to sliding moves inversely have the bijection $f_{(i,f_i,f_j)}(G', \Theta', Z') = (G, \Theta, Z)$, where $E = E' \setminus \{(r, j)\} \cup \{(i, j)\}$, $z'_{(i,j)} = (f_i, f_j)$, and $\theta_{(i,j)}$ is calculated as the sliding continuous parameters given contacting faces f_i, f_j and parent pose x_i , such that the absolute pose $x'_j = x_j$.

The position parameterizations in the floating and sliding cases are both \mathbb{R}^3 , and thus the Jacobian correction is simply 1 for these subspaces. Orientations are slightly more complex; in the floating case, the orientation is represented as a unit vector in S^3 (which has a double-covering surjective homomorphism to the rotation group $\text{SO}(3)$). In the sliding case, the orientation is represented as the Hopf fibration, which has local product structure $S^2 \times S^1$. This local topology means our Jacobian correction for the transformation between these spaces is constant, and can be simply calculated as

$$\left| \det J_{f_{m_{\text{sliding}}}} \right| = \left| \det J_{f_{m_{\text{floating}}}} \right|^{-1} = \frac{\mu(S^2) \cdot \mu(S^1)}{\mu(\text{SO}(3))} = \frac{\mu(S^2) \cdot \mu(S^1)}{\mu(S^3)/2} = \frac{(4\pi) \cdot (2\pi)}{\pi \cdot \pi} = 8$$

Gen automatically calculates the acceptance ratio for RJMCMC and makes appropriate MH moves, given the involutions and associated Jacobian corrections.

2.3.1 Representation as an inference program

Gen provides an abstract interface to automatically run user-specified programmable inference on a generative program using its built-in inference library. Among the algorithms provided, Gen provides utilities for running RJMCMC, by specifying a generative proposal function to generate auxiliary randomness, and an associated involution function that provides the associated Jacobian correction. Figure 2-4 shows an user-space implementation of these algorithmic components ¹, while Figure 2-5 shows how they may be used in a top-level inference procedure (ie. a particle filter).

¹At the time of writing this document, Gen has added a DSL for involutive MCMC which and provides an interface for the development of custom RJMCMC algorithms. See documentation at <https://www.gen.dev/dev/ref/mcmc/#Involutive-MCMC-1>

```

1 @gen function model(T::Int)
2     latent_gs = []
3     for t = 1:T
4         # structure and parameter dynamics
5         if t == 1
6             structure ~ UniformUndirectedTree(N+1)
7             params ~ params_init(structure)
8         else
9             (prev_structure, prev_params) = decompose(gs[t-1])
10            structure ~ StructureTransition(prev_structure)
11            params ~ params_dynamics(structure, prev_params)
12        end
13
14        # observation
15        latent_g = SceneGraph(structure, params)
16        obs ~ noise_model(latent_g)
17        push!(latent_gs, latent_g)
18    end
19    return latent_gs
20 end

```

(a) Top-level scene graph model

```

1 @gen function init_params(structure::SimpleDiGraph)
2     params = []
3     for i in vertices(structure)
4         if isFloating(structure, i)
5             xs = {i} ~ init_floating_pose()
6         else
7             xs = {i} ~ init_sliding_pose(parent(structure, i))
8         end
9         push!(params, xs)
10    end
11    return params
12 end

```

(b) Parameter initialization

```

1 @gen function params_dynamics(structure, prev_params, hypers)
2     new_params = []
3     for i in vertices(structure)
4         if isFloating(structure, i)
5             new_xs = {i} ~ floating_pose_dynamics(prev_params[i])
6         else
7             new_xs = {i} ~ sliding_pose_dynamics(prev_params[i], parent(structure, i))
8         end
9         push!(new_params, new_xs)
10    end
11    return new_params
12 end

```

(c) Parameter dynamics

```

1 RobustNoisyPoseLikelihood = Mixture([GaussianVMF, UniformPose])
2
3 @gen function noise_model(g)
4     # use Mixture to construct a mixture of uniform and gaussian vmf with prob_outlier of being uniform
5     observed_poses = []
6     for i in vertices(g)
7         latent_pose = get_floating_pose(g, i) # get 6DoF pose for object i
8         observed_pose = {i} ~ RobustNoisyPoseLikelihood([1 - p_outlier, p_outlier],
9                 [(latent_pose, inlier_pos_stdev, inlier_rot_conc),
10                  (outlier_bounds,)])
11         push!(observed_poses, observed_pose)
12     end
13     return observed_poses
14 end

```

(d) Noisy observational model

Figure 2-3: Probabilistic pseudocode for an example dynamic scene graph model

```

1 @gen function structure_move_randomness(prev_trace, i::Int)
2     prev_structure = get_structure(prev_trace)
3     floating_to_sliding_move = isFloating(prev_structure, i)
4
5     if floating_to_sliding_move
6         prev_parent_node = ROOT_NODE_ID
7
8         # sample a new parent object from a categorical
9         probs = ones(N)
10        probs[i] = 0.0 # don't propose to make the object its own parent
11        new_parent_node_object ~ categorical(1:N, probs ./ sum(probs))
12
13        # sample new faces from a categorical
14        box_faces = [:bottom, :top, :left, :right, :front, :back]
15        (parent_face_probs, child_face_probs) =
16            structure_move_face_distributions(prev_trace, i, parent_object)
17        parent_face ~ categorical(box_faces, parent_face_probs)
18        child_face ~ categorical(box_faces, child_face_probs)
19    else
20        prev_parent_node = parent(prev_structure, i)
21        new_parent_node = ROOT_NODE_ID
22    end
23    new_structure = replaceEdge(prev_structure, i, prev_parent_node, new_parent_node)
24    return (new_structure, floating_to_sliding_move)
25 end

```

(a) Generative function for sampling auxiliary randomness. In Gen, sampling the discrete move is combined with sampling continuous auxiliary randomness in a single function.

```

1 function structure_move_involution(
2     prev_trace,           # original sampled scene graph model
3     fwd_randomness_choices, # choices (i, f_i, f_j) sampled from structure_move_randomness
4     fwd_randomness_ret,    # values returned from structure_move_randomness
5     fwd_randomness_args    # arguments passed to structure_move_randomness
6 )
7     i, = fwd_randomness_args
8     (new_structure, floating_to_sliding_move) = fwd_randomness_ret
9     prev_scene_graph = get_latent_scene_graph(prev_trace)
10
11    proposed_choices = choicemap() # to set proposed reversible jump moves in the trace
12    bwd_choices = choicemap() # to tell Gen which involution we're using in the backward direction
13    proposed_choices[:structure] = new_structure
14
15    if floating_to_sliding_move
16        parent_object = fwd_randomness_choices[:parent_object]
17        parent_face = fwd_randomness_choices[:parent_face]
18        child_face = fwd_randomness_choices[:child_face]
19        sliding_choices =
20            get_equivalent_sliding_param(prev_scene_graph, parent_object, i, parent_face, child_face)
21            set_submap!(proposed_choices, :params => (:sliding, i), sliding_choices)
22            log_jacobian_correction = log(4)
23    else
24        prev_parent_object = parent(get_structure(prev_scene_graph), i)
25        (choices, prev_parent_face, prev_child_face) =
26            get_equivalent_floating_param(prev_scene_graph, prev_parent_object, i)
27        bwd_choices[:parent_object] = prev_parent_object
28        bwd_choices[:parent_face] = prev_parent_face
29        bwd_choices[:child_face] = prev_child_face
30        set_submap!(proposed_choices, :params => (:floating, i), cm)
31        log_jacobian_correction = -log(4)
32    end
33
34    # update trace with proposed values
35    args = get_args(prev_trace)
36    argdiffs = map(_) -> NoChange(), args)
37    new_trace, weight, = update(prev_trace, args, argdiffs, proposed_choices)
38    weight += log_jacobian_correction
39    return (new_trace, bwd_choices, weight)
40 end

```

(b) Involution for reversible jump moves

Figure 2-4: Custom implementation of RJMCMC algorithm for structure inference

```

1 function rejuvenation_kernel(traces)
2     for (i, tr) in enumerate(traces)
3         for _ in 1:iters
4             # gaussian drift, gradient search, data-driven proposals, etc.
5             tr = continuous_parameters_kernel(tr)
6
7             # our custom rjmcmove
8             tr = rjmcmove_structure_kernel(tr)
9         end
10        traces[i] = tr
11    end
12 end

```

(a) Example rejuvenation kernel for continuous and discrete parameters

```

1 function scenegraph_particle_filter(observed_poses::Vector{SceneGraph})
2     # generate initial trace
3     init_obs = scenegraph_to_choicemap(observed_poses[1], all_objs)
4     state = initialize_particle_filter(model, model_args, init_obs, num_particles)
5
6     # run forward model and inference
7     for t = 1:length(observed_poses)+1
8         new_obs = scenegraph_to_choicemap(observed_poses[t+1], all_objs)
9         state = rejuvenation_kernel(get_traces(state))
10        particle_filter_step!(state, (t+1,), (UnknownChange(),), new_obs)
11    end
12    return state
13 end

```

(b) Top-level inference code for a particle filter scene graph model.

Figure 2-5: Top-level inference procedure for particle filtering

2.4 Example application domains of scene graphs

We present some example visual perception domains where scene graphs provide a useful abstraction for representing scene information. Our applications have thus far been applied with a specific orientation toward two main problems: inferring structured object relationships from unstructured poses of those objects, and robustly filtering object poses under failures in bottom-up visual perception. Our modeling choices were made with these goals in mind, so we focus on describing how these features are captured by our representation. In theory, we can extend these models to include even richer information about a scene, including dynamic physical properties of objects, or their functional (as opposed to geometric) relationships; we leave these and other applications for later work.

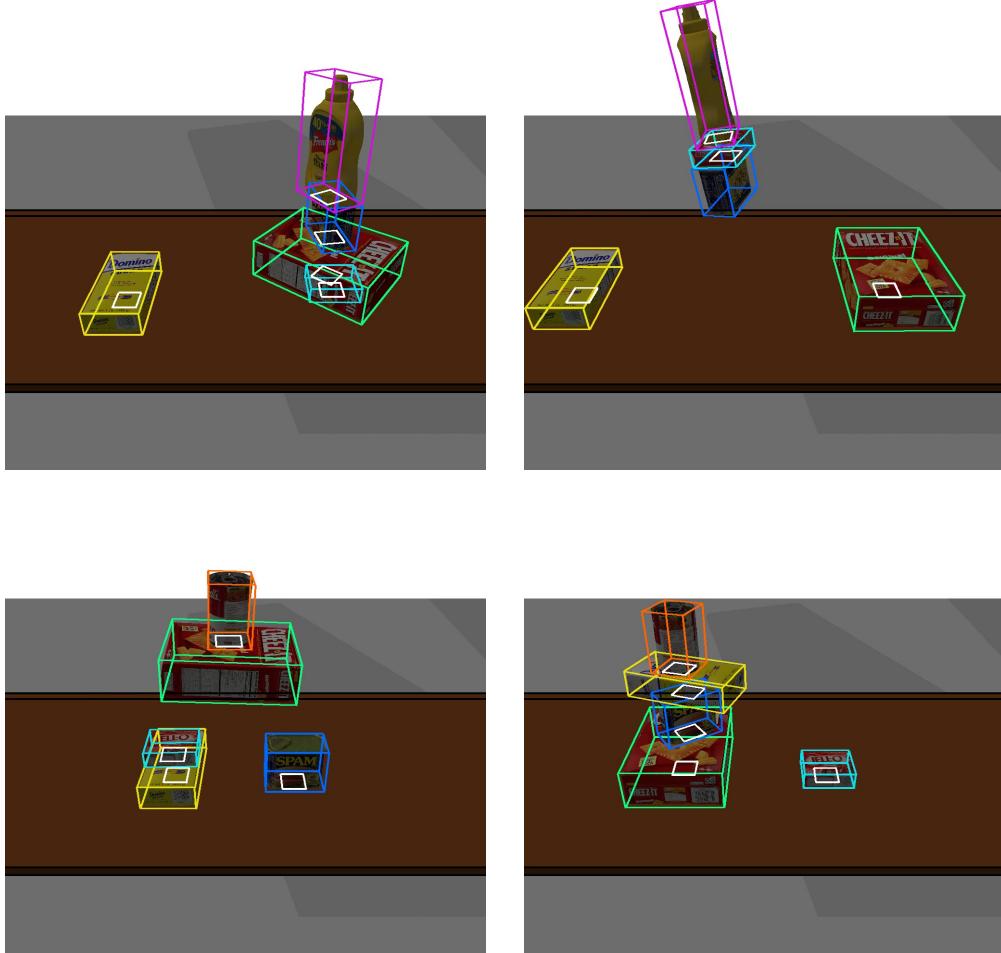


Figure 2-6: Synthetic scenes generated using the GenSceneGraphs library in simulated dim light. The colored wireframe bounding boxes represent objects in the scene graph, and the white boxes represent contact edges (see Chapter 3 for more detail on how this visualization was produced).

2.4.1 YCB objects on a synthetic tabletop

Our scene graph abstraction is concretely implemented as a part of the GenSceneGraphs library. This library provides facilities for constructing and manipulating scene graphs composed of objects that can be rendered in a synthetic view. Figure 2-6 shows a variety of example scenes generated using this library; each visualized scene has an underlying scene graph representation that captures all relevant information. As such, these rendered scenes show an intuitive view of what our information our representation captures.

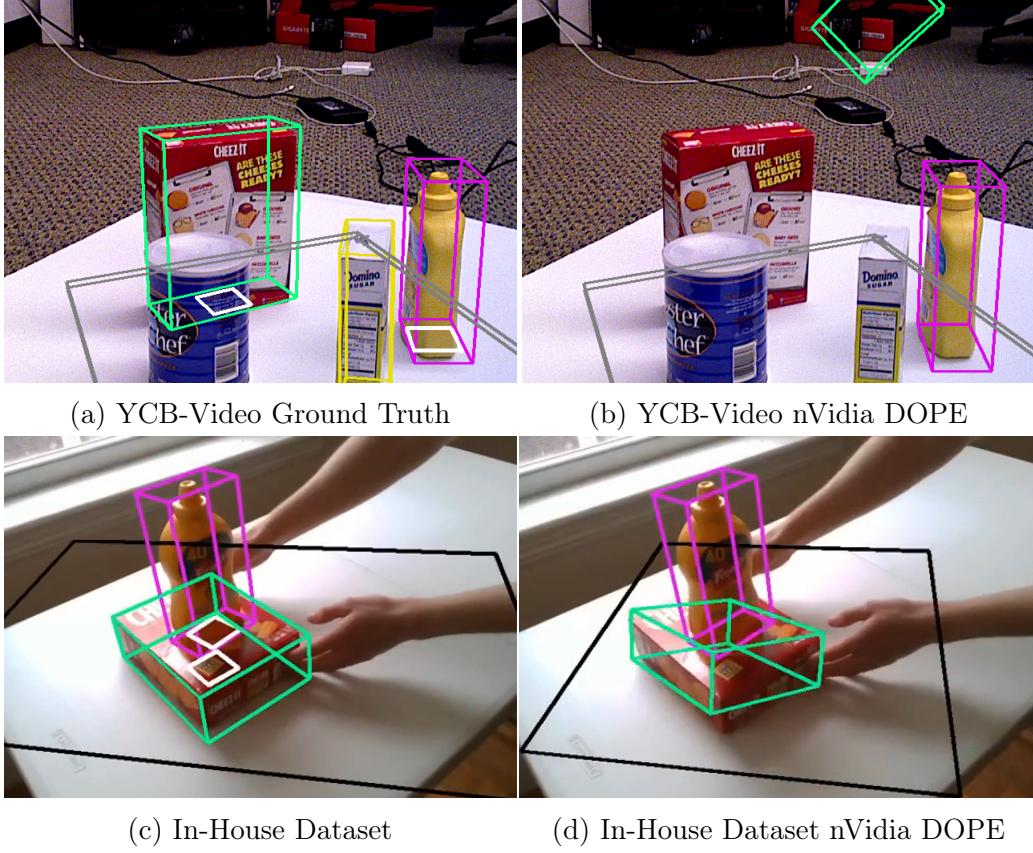


Figure 2-7: Example data from real-world scenes containing objects from the YCB dataset. (a) and (c) show the underlying latent scene graph. (b) and (d) show observed inaccurate pose estimates generated from the nVidia DOPE detector. (b) in particular demonstrates the types of catastrophic detection and localization errors that deep neural networks can exhibit.

2.4.2 Real YCB objects on a physical tabletop

Our scene graph representation can be used to represent information about real-world scenes as well. Figure 2-7 shows how scene graphs can represent the underlying poses and geometric relationships between objects in the real world. In contrast to the synthetic scenes, the real world contains a much larger variety and range of extraneous variables, that inject noise into visual perception systems and make inference more difficult. Deep neural networks have shown incredibly promise in real-world object perception. However, neural networks struggle with robustness and generalization,

and the presence of occluders, or even unfamiliar objects, can cause them to fail, as Figure 2-7 also shows. Our scene graph model was designed with the goal of filtering noisy visual perception systems, and enhancing them with structured information, by inferring the underlying scene graph representation from a set of noisy bottom-up detections.

Chapter 3

Visualizing Scene Graphs

Visualizing the richly structured geometric information contained in a generative scene graph model is vital for proper debugging and analysis of modeling and inference programs. This chapter presents a series of visualization utilities for rendering distributions over scene graphs. Thus this method serves as one of the main utilities used in data-exploration (neural detections are special case scene graphs with no edges), model development, and testing.

3.1 Desiderata

It is important to be able to clearly distinguish each part of a scene graph (G, Θ, Z) ; as will be seen later, incorrect behavior in any part of the model or inference introduces large qualitative variation in the posterior and inferred approximation. Rendering a single state estimate is a special subcase that serves as a primitive for more advanced visualizations; in particular visualizations that include useful information about distributions over scene graphs $p(G, \Theta, Z)$. Providing an clearly interpretable view of all possible distributions is a huge task, so to simplify, the utilities are focused on sample-based approximations of unimodal distributions over continuous parameters, with relatively small numbers of structures. Common characterizing features are the mean and uncertainty, so the visualizations are developed with the goal of providing a clear view of these features.

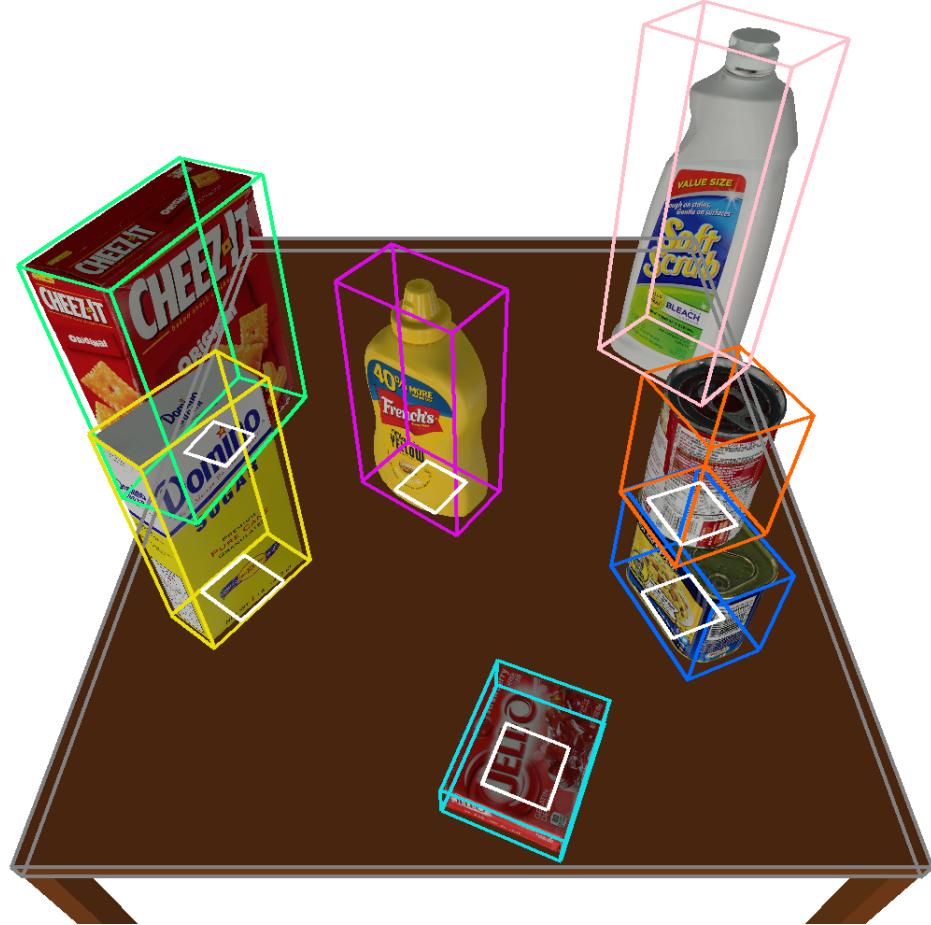


Figure 3-1: Visualization of a synthetic scene with its abstract scene graph overlaid. The Soft Scrub bottle is above the table, so doesn't have an edge.

3.2 Examples

The following section proceeds with an explanation of the visualization utilities using a series of illustrative examples. The first example lays out the primitive visualization of a single scene graph state. This is followed by a method to visualize distributions over discrete structure. These are then combined into a series of visualizations over distributions of complete scene graphs. The final example concludes with a practical development use-case in visualizing a scene graph particle filter.

3.2.1 Visualizing a single scene graph

The main utility is a function that accepts an image, a scene graph (G, Θ, Z) where $G = (V, E)$, and a camera configuration, and renders that scene graph as a wireframe representation overlaid on top of the image, as viewed from the provided camera specification. Figure 3-1 demonstrates using this function to see a synthetic rendered scene’s underlying scene graph. For each object $o \in O$ in the scene graph, the function renders a colored wireframe bounding box with the dimensions of o , and 6DoF pose given by x_o . Importantly, this is the *absolute* pose of the object, including any slack in relative contacts. Thus, the continuous parameters $\Theta := \{\theta_e\}_{e \in E}$ are not explicitly used in rendering, but can be distinguished in contacting objects $i, j \in O$ by the distance in their corresponding wireframe renderings. Contact edges $e \in E$ are visualized as white squares, located at the center of face f_i . Face f_j is not directly visualized, but can often be inferred from which face is closest to f_i . This does not preclude potential ambiguity for certain pathological slack terms that rotate i close to a multiple of $\pi/2$. However, in practical modeling applications such occurrences are rare, as sensible priors weight heavily against large slack terms.

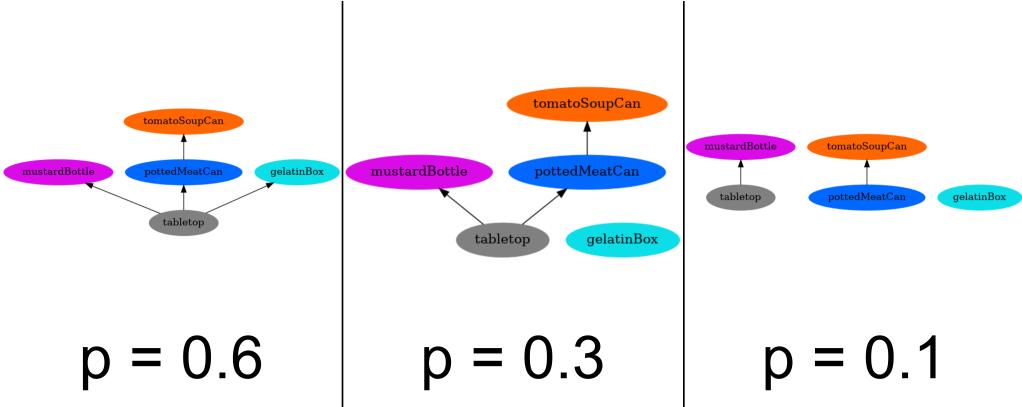
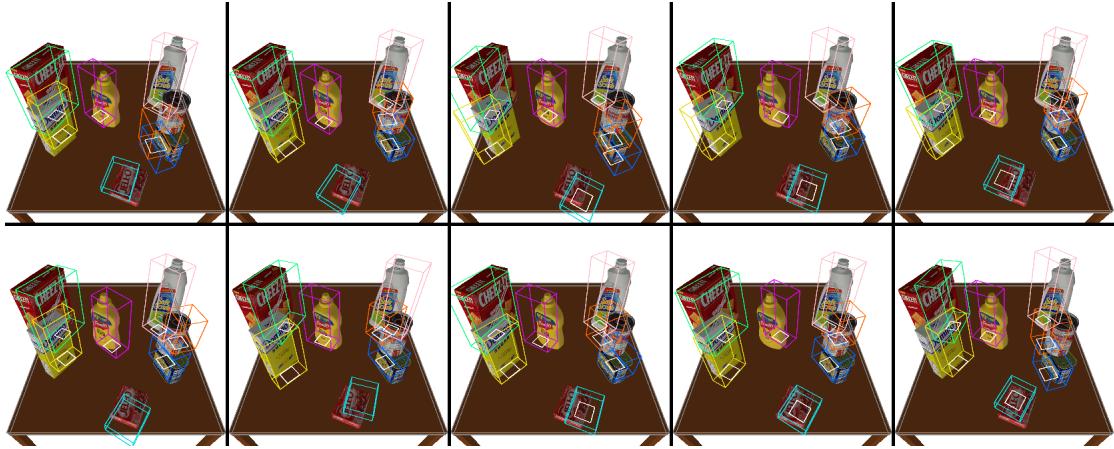


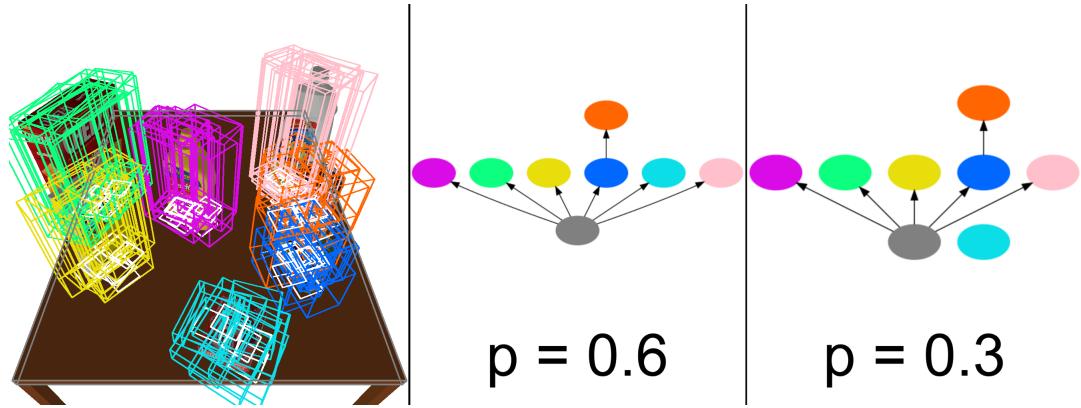
Figure 3-2: Visualization of a distribution over abstract scene graph structure using Graphviz.

3.2.2 Distributions over structure beliefs

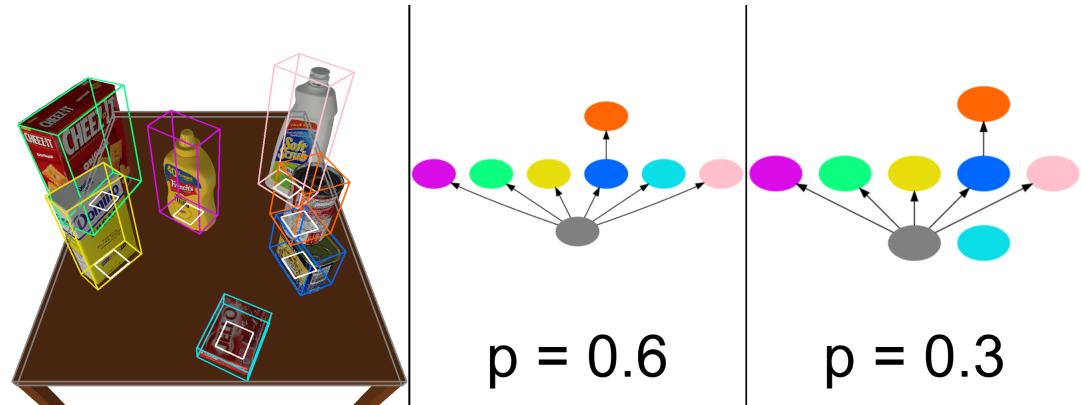
Using the simple scene graph overlay provides a representation of a single scene graph state, but it's not immediately clear how to use this utility to represent distributions over scene graphs. We developed a utility based off the Graphviz [3] graph visualization library to more clearly view distributions over discrete structure. It accepts a distribution over scene graphs, and renders a specified number of the most probable present in a scene. Figure 3-2 an example distribution, which can clearly represent several of the most probable structures in a clear fashion. This tool can be combined with the wireframe overlay to give rich visualizations of distributions over scene graphs.



(a) Visualization of each sample's state using the scene graph visualization utility.



(b) All 10 samples, rendered over the same image. Only the two most frequent structures are shown.



(c) Aggregated visualization of all the samples. Only the two most frequent structures are shown.

Figure 3-3: Various methods for visualizing multiple beliefs. Node labels are omitted in the rendered structure distribution, and instead the wireframe rendering and GraphViz node for an object o share the same color.

3.2.3 Distributions over scene graphs

Finally, Using the methods to render rich views of multiple samples. To generate the example visualization distribution, 10 scene graphs $\{\mathcal{G}_i\}_{i=1}^{10}$ were generated, with object parameters sampled from a uniform distribution centered at the graph as seen in Figure 3-1. Figure 3-3a shows a simple form of visualizing the collection of samples, as a collage of separate renderings of each sample. This provides the most information, but at a glance it's difficult to get a sense of the mean and uncertainty of the distribution it's representing.

A more information dense representation is the visualization of an aggregate of the separate samples into a single representative scene graph “belief” $\mathcal{G}^* = \text{agg}(\{\mathcal{G}_i\}_{i=1}^N)$, that is then overlaid on top of the rendered scene. This, combined with the abstract structure visualization to improve the ability to see uncertainty in scene structure, forms the basis for a scene graph visualization interface.

One possible aggregation method is simply taking the disjoint union $\text{agg}(\{\mathcal{G}_i\}_{i \in 1:N}) = \bigsqcup_{i=1}^N \mathcal{G}_i$. Figure 3-3b shows this has the effect of overlaying all of the beliefs on top of the same image. This function can give a sense of the amount of uncertainty in object positions and rotations, but obscures the underlying object, making it difficult to determine the accuracy of the pose beliefs.

Another proposed function is $\text{agg}(\{\mathcal{G}_i\}_{i \in 1:N}) = (G^*, \Theta^*, Z^*)$, which first takes the most probable structure $G^* = \text{argmax}_G \hat{p}(G, \Theta, Z)$ among the samples. The discrete parameters Z^* are then selected arbitrarily from one of the graphs that has this structure. The continuous parameters Θ^* are then determined by the average of that scene's objects' absolute poses $x_o^* = \text{avg}(\{x_{(i,o)}\}_{i \in 1:N})$ across all other samples (the average position uses a simple sum, while the average orientation is calculated using the method described by Markley et. al in [12]). Figure 3-3c shows an example of this method. In contrast to the Figure 3-3b, this method clearly shows the accuracy of pose beliefs, but loses information about uncertainty in the continuous parameters. These trade-offs demonstrate choosing a visualization is contextually dependent on if the accuracy or uncertainty is more relevant.

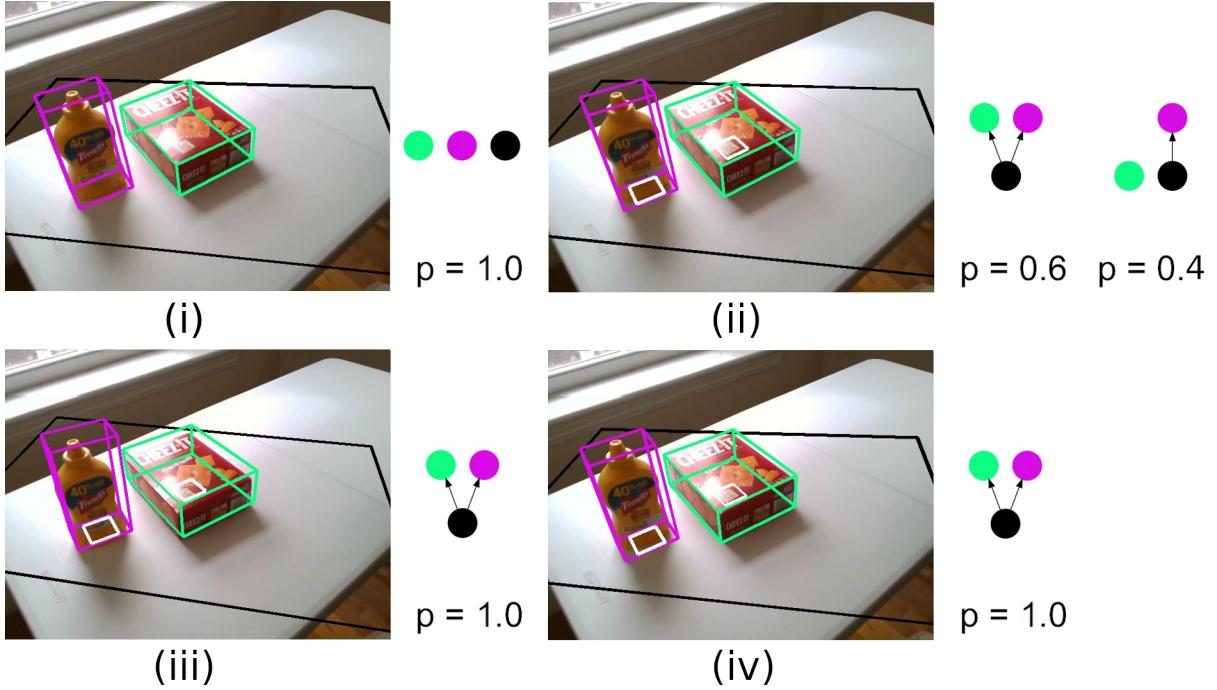


Figure 3-4: Visualization of sampling and rejuvenation moves in a particle filter. A complete video of this scene can be seen at: https://www.youtube.com/watch?v=0_0TvrGC65Q. (i) shows the first time step with the overlaid beliefs from the initial time step prior (initialized to the observed neural pose estimates), before structure inference. (ii) shows the beliefs over the first time step, after running 1 step of the rejuvenation kernel. (iii) shows the beliefs over the second time step, after sampling from the dynamics model. (iv) shows the beliefs over the second time step, after running 20 steps of the rejuvenation kernel.

3.2.4 Visualizing inference in a particle filter

Finally, this section demonstrate a real-world application by showing the actual usage of these visualization utilities in a particle filter with a corresponding complex inference program. This demonstrates how the methods introduced in this chapter can be used in practice for visualizing and debugging complex inference programs. The scene graph model leverages the prior from 2.1.3 and dynamics from 2.1.4. Object poses are initialized to their observed neural detections in the first time step. The inference program is based off of a rejuvenation MCMC kernel, after sampling from the prior for the first and second time step. This kernel is in turn composed of an interleaved RJMCMC kernel (see 2.2) for discrete structure, and a drift kernel for

continuous parameters. The particle filter was ran with 10 particles, and 20 iterations of the rejuvenation kernel. Figure 3-4 uses the method in Figure 3-3c to aggregate the particle filter state into a stable visual estimate of the mean object poses, and to view the distribution over inferred structure.

Chapter 4

Analysis of Posteriors and Inference by Enumeration

In a model as complex as a generative scene graph program, it becomes essential to carefully check and analyze the behavior of the posterior to ensure it is a good representation of the data. This chapter describes a set of synthetic experiments that carefully inspect isolated aspects of the model to ensure their posteriors exhibit reasonable behavior that conforms to intuition.

4.1 Enumerative inference

In conducting an analysis of a generative scene graph program, it is useful to inspect the posterior distribution induced by constraining some subset of the model. Calculating the full posterior is intractable (indeed, this is why inference is necessary in the first place). However, if most of the model’s latent variables are constrained, it’s reasonable to enumerate over a small set of free variables, and calculate a normalized density for a range of their values, which is a close approximation to the posterior conditioned on the constrained variables. Note that this approach is limited in its ability to examine interactions between variables, and in generality the posterior conditioned on just the observed neural pose detections can behave radically differently from the posterior generated by constraining most of the model. Nonetheless,

this chapter shows that these projections can still be a rich source of information for understanding and improving scene graph priors, especially model hyperparameters.

4.2 Structure posterior

This section looks at the model’s predictions for structure between objects, as a function of the observed vertical offset y between their closest faces. The analysis conducted considers the static model and dynamic model respectively, and looks at the behavior of the structure posterior under multiple settings of model hyperparameters. It ultimately demonstrates the high variability of the posterior’s qualitative behavior with respect to hyperparameters; this in turn shows the vital importance of proper hyperparameter tuning in ensuring the correctness of scene graph models.

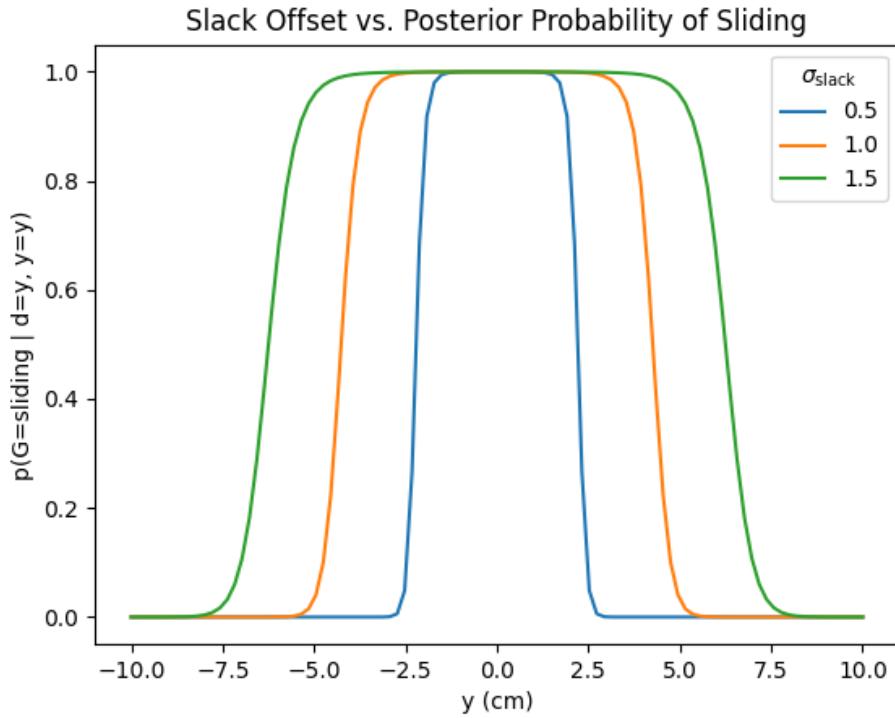


Figure 4-1: Probability that an object is sliding in a static scene with different settings of σ_{slack} , and observed/latent vertical offset y .

4.2.1 Static model

An experimental scene was created with two objects, in order to visualize the enumerated structure posterior for G , given an observed one-dimensional vertical offset y between the object’s closest faces. In the model, y also denotes the vertical gap in the observed object’s faces, d denotes the sliding object’s *slack offset*, with prior $p_{\text{norm}(0, \sigma_{\text{slack}})}(d)$ where σ_{slack} is the hyperparameter for the slack prior (see 2.1.2 for a reminder of the slack variable model in sliding contacts). The latent parameters are restricted to be the same as the observed poses, and the observed poses of the objects are also taken to be the same, except for their one-dimensional offset y . The structure prior is uniform, so all structures have equal prior probability.

Figure 4-1 shows the posterior probability that G is in a “sliding” configuration, given an offset y , for a few different settings of σ_{slack} . Predictably, the wider the distribution on slack, the larger a gap the model is willing to allow while still classifying the objects as “sliding”. Note the relatively sharp transition in the sliding probability from almost 0 to almost 1, at a discrete cutoff point; later it’s discussed if this could be a result of a poor model for the slack variable.

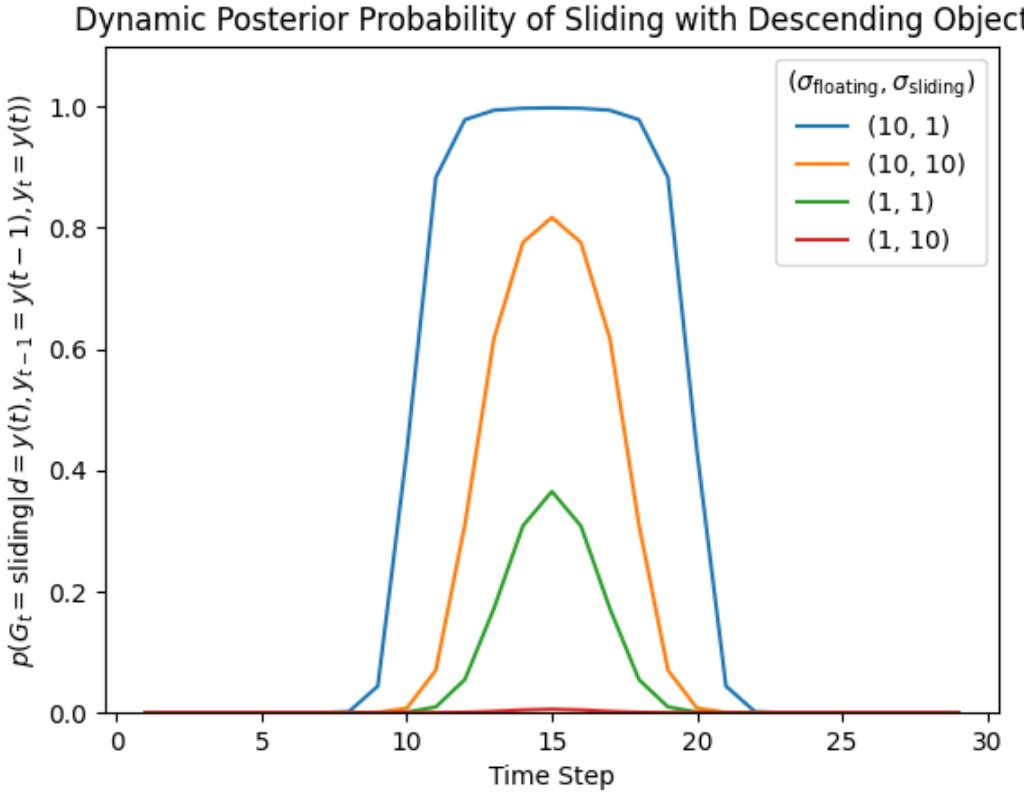


Figure 4-2: Probability that the scene structure G_t is in a “sliding” configuration at time t , in a dynamic scene where the object’s observed position and slack offset is given by $y(t) = 10 - 0.67 \cdot t$ cm, with hyperparameters $(\sigma_{\text{floating}}, \sigma_{\text{sliding}})$.

4.2.2 Dynamic model

The second experimental scene is very similar to the first, except for the introduction of a dynamic element to examine the effect of the dynamics parameters. Across 30 time steps, the top object (and its observation) is lowered along y from 10 cm above the bottom object, to 10 cm interpenetrating with the bottom object. For simplicity, the prior structure is once again uniform at each time step, and thus temporally independent. Let $\sigma_{\text{slack}} = 1$ cm, and let the floating position dynamics be a 3D normal distribution with standard deviation σ_{floating} , while the sliding dynamics are a 2D normal distribution with standard deviation σ_{sliding} .

Figure 4-2 shows the posterior probability that G_t is in a “sliding” configuration, given the observed/latent offset y , for each time step t of the dynamic scene. Adding

dynamics has added completely new behavior to the structure posterior, dependent on the additional hyperparameters. The pose displacement is always -0.67 cm every time step, which is smaller than all values tested for σ_{floating} and σ_{sliding} , implying that the most accurate dynamics model in this figure is the one with the tightest distribution. The change in posterior probability in sliding structure is a consequence of the Bayesian Occam's Razor; the posterior probability for sliding is greatest (blue curve) when the sliding dynamics model is much more confident (concentrated) than the floating dynamics model, and is least in the opposite case (red curve). This is the first instance with the Bayesian Occam's Razor adding complications to the development of scene graph models, but there are additional cases where the concentration of different branches of the model has a huge impact on the behavior of the posterior.

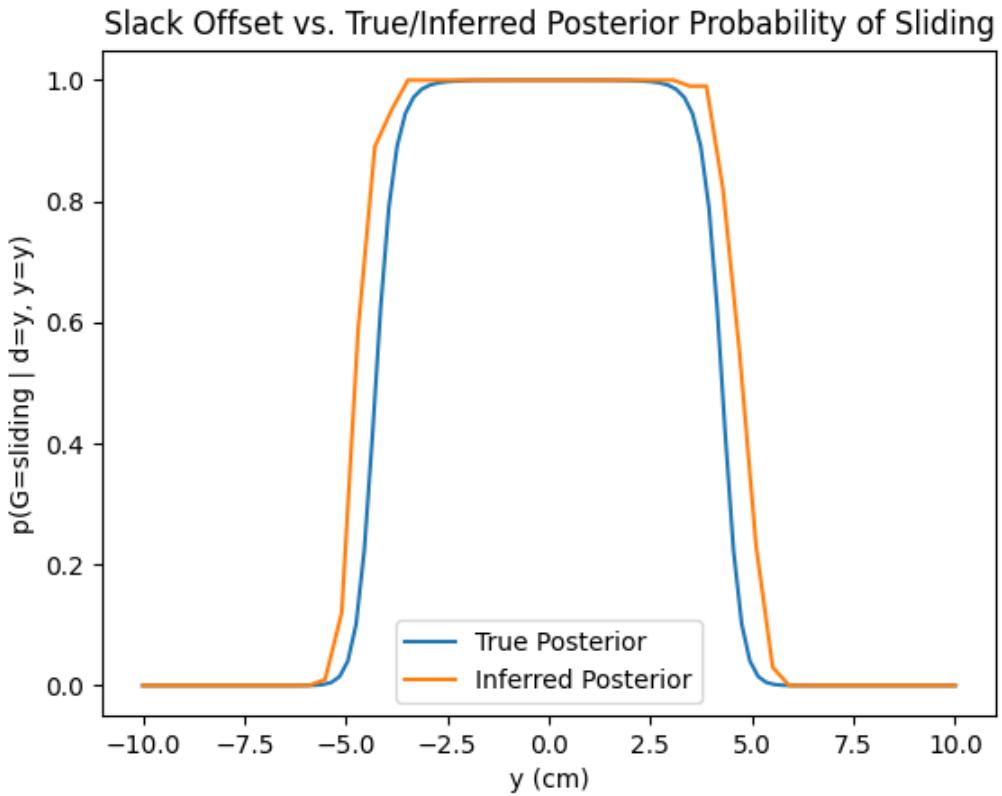


Figure 4-3: RJMCMC structure inference results. For each value of d , the approximate probability of the top object sliding is inferred as the average structure of 100 particles run with 3 iterations of the reversible jump moves applied to the top object.

4.2.3 RJMCMC structure inference

This section includes a brief comparison of the RJMCMC structure inference procedure to the enumerated posterior. This experiment replicates the scenario shown in Figure 4-1 with $\sigma_{\text{slack}} = 1$ cm, and compares the performance of the inferred posterior probability of a sliding structure. The figures shows RJMCMC accurately recovers the posterior in this situation, which is a successful first test of the correctness of the algorithmic implementation of structure inference on scene graphs.

4.3 Contact slack model

The second analysis conducted examines the enumerated posterior over the slack offset d . Having a sensible posterior for slack variables is important for having a reasonable model over structure; if the slack model provides a poor explanation for any observed gap between “contacting” objects, the structured model will also provide a poor explanation for the scene, artificially boosting the probability the model assigns to floating structures. Recall the prior model over slack is $p_{\text{slack}}(d) = p_{\text{norm}(0, \sigma_{\text{slack}})}(d)$.

4.3.1 Varying the gap between sliding objects

A similar scene to the previous section is constructed with two objects, to visualize the enumerated slack offset posterior for d , given a fixed sliding structure. The objects are again restricted to share the same observed pose, except for a one-dimensional vertical offset y between their contacting faces, but the latent offset d in their continuous parameters is left free (we enumerate over this instead). In the prior model let $\sigma_{\text{slack}} = 1$ cm, and let $p_{\text{inlier}} = 1$. Denote the density over the observed vertical offset x as $p_{\text{obs}}(x|y) = p_{\text{norm}(y, \sigma_{\text{inlier}})}(x)$, and let $\sigma_{\text{inlier}} = 0.5$ cm. Figure 4-4 shows the posteriors corresponding to various settings of the observed offset y . Because the density $p(d|G = \text{sliding}, y = y) = p_{\text{obs}}(d|y = y) \cdot p_{\text{slack}}(d)$ is a product of normal distributions, the posterior is also a normal distribution, with mean located in-between the means of the two modeling distributions. This implies the most probable gap between two

objects given a sliding structure is counter-intuitively in-between 0 and the observed gap y ; this may explain why there is such a sharp cutoffs in the probability of a sliding structure in Figures 4-1 and 4-2. As the observed gap between two objects becomes larger, the model over slack offset quickly becomes a poor explanation of the observation. A more reasonable model would place high mass around $d = y$, as this would explain the observation most accurately.

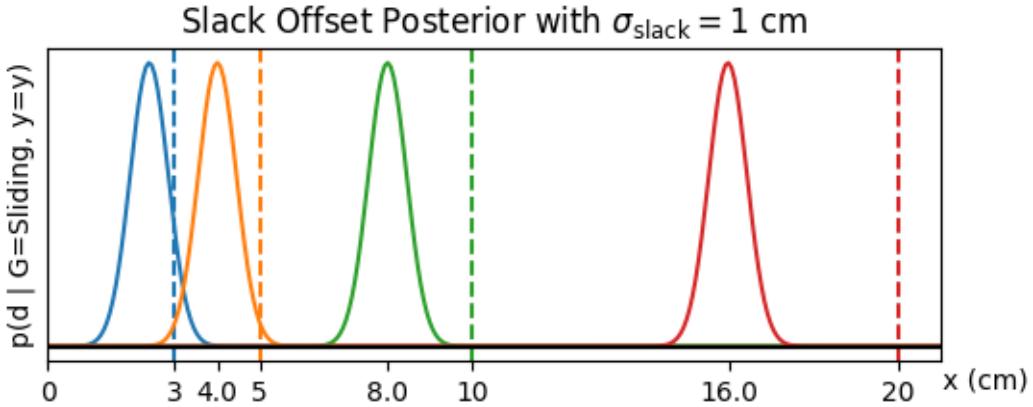


Figure 4-4: Posterior distribution on slack offset d . The dashed lines represent the set of observed y values, while the curve of the same color represents the slack offset posterior, given the observed y and sliding structure.

4.3.2 Hyperprior over slack standard deviation

This section describes the addition of a hyperprior over σ_{slack} , to more accurately allow for uncertainty in the size of the slack term. The slack offset is now a variable sampled from an exponential distribution $\sigma_{\text{slack}} \sim p_{\exp(0.2)}(\cdot)$ with rate parameter $\lambda = 0.2$. Figure 4-5 shows the joint posterior over σ_{slack} and the slack offset d , for an observed vertical offset $y = 5$ cm. The figure shows that the mode of the posterior is located much closer to the observation than without a hyperprior over σ_{slack} . Indeed, as seen in Figure 4-5, the posterior exhibits much more reasonable behavior when letting $\sigma_{\text{slack}} = \sigma_{\text{slack}}^*$. This analysis thus concludes that adding a hyperprior over slack is a viable way to increase the accuracy of the model with a sliding structure, especially when the observed gap between objects is large.

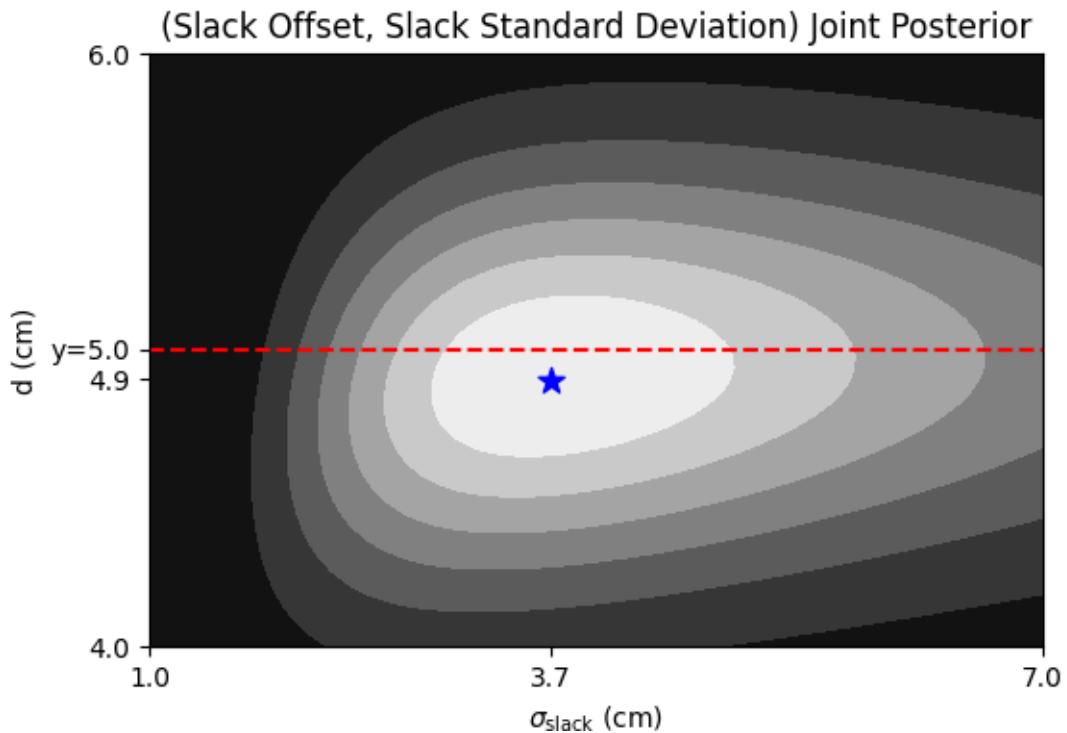


Figure 4-5: Joint posterior $p(d, \sigma_{\text{slack}} | y = 5.0)$. The dashed red line indicates the observed position $y = 5.0$ of the top object. The blue star indicates the maximum a posteriori $(\sigma_{\text{slack}}^*, d^*)$.

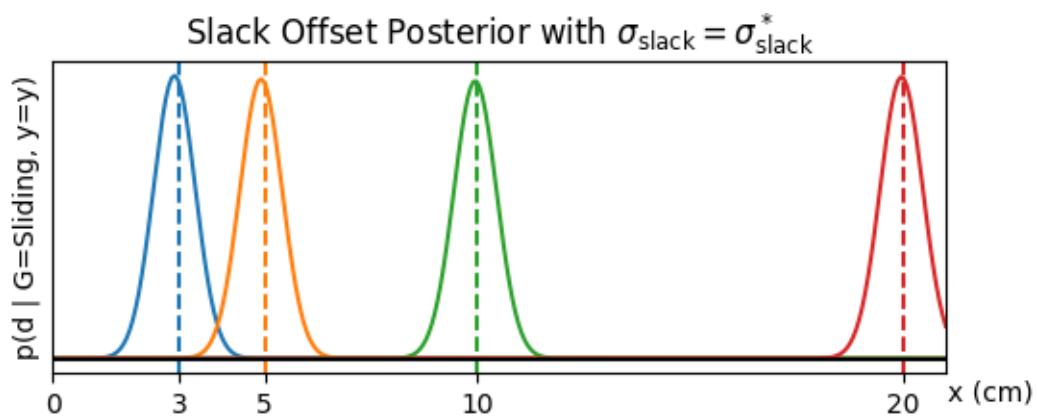


Figure 4-6: Slack offset posterior for d , conditioned on $\sigma_{\text{slack}} = \sigma_{\text{slack}}^*$. The dashed lines represent the set of observed y values, while the curve of the same color represents the posterior over slack given the observed y and sliding structure.

4.4 Neural outlier model

An essential component of the proposed scene graph model is a robust model of the output of a neural detector. The dynamics model presented in 2.1.4 discourages large movements in the latent poses of objects, while the prior observational model is a simple mixture of a uniform outlier distribution over poses in a finite bounded box region, with component $p(\text{outlier}) = 0.1$, and a direct product of a normal and VMF inlier distribution, with component $p(\text{inlier}) = 0.9$. Ideally, the product of this model and the dynamics model would allow the scene graph model to confidently predict that large leaps in the latent position over a single time step are less probable than that specific observation being an outlier “flicker” detection, sampled randomly within the scene.

Figure 4-7 shows a scene in which an object’s observed pose jumps from a latent position at the origin, $x_1 = 0$, to one located a distance y_2 . On the left, when the dynamics and observational model are equivalent width, the posterior predicts that the detection is more likely to be an outlier than not at around $y_2 = 9.6$ cm. On the right, when the dynamics is significantly more broad than the observational model, the posterior predicts that the detection is more likely to be an inlier, all the way up to $50\text{ cm} = 5\sigma_{\text{floating}}$ away from the previous position. The prior probability that an object travels a distance greater than $5\sigma_{\text{floating}}$ away from the mean is less than $5.73e - 7$, which is dramatically less than the prior probability of an object being an outlier $p(\text{outlier}) = 0.1$.

The model on the right is counter-intuitively predicting that the object made an extraordinarily unlikely jump. If we examine the log probability density functions evaluated at the modes of the latent pose joint distributions, for the outlier and inlier

models respectively, we begin to see why:

$$\begin{aligned}
& \log p(x_2 = 0, y_2 = 50, \text{outlier} | x_1 = 0) = \\
& \quad \log p(\text{outlier}) + \\
& \quad \log p(x_2 = 0 | x_1 = 0, \text{outlier}) + \\
& \quad \log p(y_2 = 50 | \text{outlier}) = \\
& \quad -2.303 + \textcolor{blue}{10.717} - (3 \cdot 0. + 2.289) = 6.125 \\
& \log p(x_2 = 50, y_2 = 50, \text{inlier} | x_1 = 0) = \\
& \quad \log p(\text{inlier}) + \\
& \quad \log p(x_2 = 50 | x_1 = 0, \text{inlier}) + \\
& \quad \log p(y_2 = 50 | x_1 = 0, x_2 = 50, \text{inlier}) = \\
& \quad -0.105 - 1.783 + \textcolor{red}{17.625} = 15.737
\end{aligned}$$

The dynamics model's density (blue) weights heavily in favor of the observation being an outlier. However, the inlier pose density (red) is so tightly concentrated compared to the outlier pose model, that it dominates the calculation of the overall model weight. The posterior tells us that the model would rather have the latent pose make an incredibly unlikely jump, if it means it can explain the observed pose confidently using the inlier observational model. This is another occurrence of the Bayesian Occam's Razor, where the most concentrated component of the model (in this case the observation model), determines where the majority of the posterior mass is distributed.

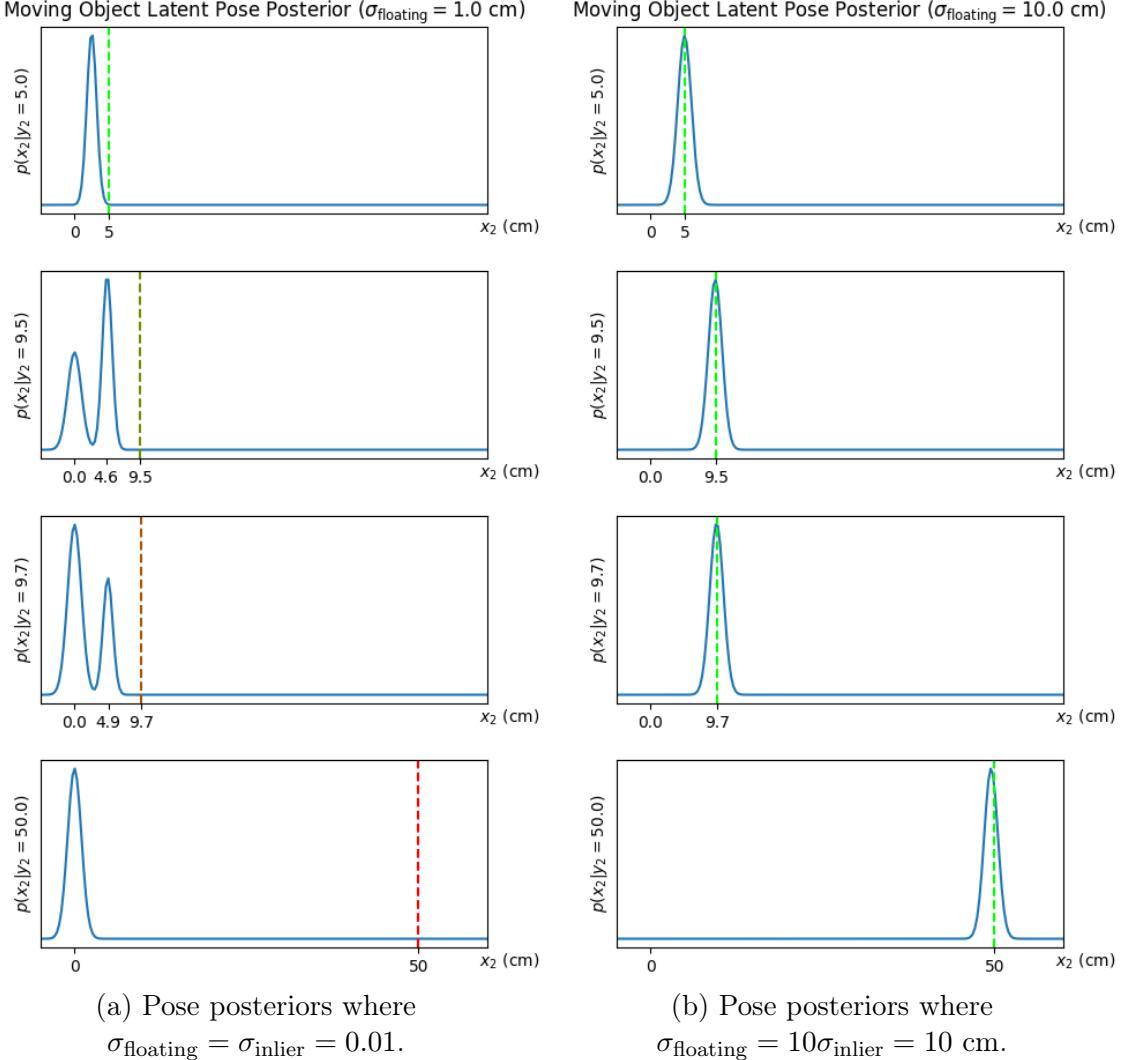


Figure 4-7: Posteriors over the latent pose of an object $p(x_2|y_2) = p(x_2|y_2, x_1 = 0)$, under two different settings of the dynamics parameter. The noisy observational model's inlier distribution is given by $\sigma_{\text{inlier}} = 1 \text{ cm}$, while the prior probability of an outlier is $p_{\text{outlier}} = 0.1$. The dashed line represents the observed pose y_2 , with color representing the probability that y_2 is an outlier (totally green means $p_{\text{inlier}} = 1$, while totally red means $p_{\text{outlier}} = 1$).

4.5 Lessons for improving scene graph priors

There are two big lessons from these experiments regarding the improvement of the scene graph model. The first lesson is that the qualitative behavior of the posterior is highly sensitive to the proper tuning of hyperparameters. As seen in the analysis of the structure posterior, hyperparameter settings had a crucial impact on the performance of the model. Furthermore, this section showed how introducing dynamics over continuous parameters impacts the posterior probability of a contact edge existing between two objects. Interactions like these can be highly counter-intuitive, and difficult to explore using synthetic tests like the ones explored here. It's for these reasons that in future development of scene graph models, training model hyperparameters and analyzing model performance using real-world data will be a vital step in improving accuracy. The subsequent chapter explores this further, and especially how real-world data can be used to test and improve scene graph models.

The second lesson is the importance of considering how the *relative* concentrations of different component distributions in the model will affect the posterior. More specifically, the Bayesian Occam's Razor suggests that distributions with very high concentrations of their mass will tend to have an overinflated impact on where mass ends up in the posterior. In the simplest case, the relative concentration of distributions in different model structures have a large impact on which model structures are more probable in the posterior. In the extreme case, this can lead to entire parts of the prior model being neglected in favor of assigning high weights to areas that maximize the density of the especially peaky model component. The proposed scene graph model has a large family of possible model structures, with different continuous parameter distributions of varying concentration and dimension. *All* of these distributions, and their relative concentration, matter for the calculation of the structure posterior, and thus must be considered *jointly* in order to have a truly accurate understanding of the posterior in the full model.

Chapter 5

Learning and Testing from Real-World Data

Enumerative inference on synthetic traces can provide low-level confirmation about basic properties of our model, under simple qualitative conditions. However, real-world analysis provides a much richer set of possibilities in terms of analyzing and improving our model. In this chapter, we collect a set of real-world videos containing common household items from the YCB object dataset, along with annotated values for all of the variables in the static (single frame) version of our model. The data is then used in a gradient search to train a subset of hyperparameters that have thus far been hand-selected through trial-and-error, but may not be properly tuned to model real-world data. We also measure the performance of our proposed structure inference algorithm on predicting the ground-truth structure from a set of object poses.

5.1 Data specification

We collect a set of experiments with static objects to analyze the model in the static case. Noise is added via occluders that temporarily block the camera’s line of sight to the objects. The raw video was collected using the Intel RealSense D435 RGB/Depth camera. Only the RGB information is used for the neural detector, but depth data was collected to enable plane detection, and for future tests on models

containing likelihood over depth data. Observed poses are generated using the publicly provided nVidia DOPE detector GitHub repository. Ground truth outlier status is manually annotated for each frame in the collected videos. Note that using the vertices, we can also use the scene graph structure to determine which detections are false positives or negatives.

Ground truth structure is manually annotated once per scene, and the ground truth poses are estimated as the average over all an object’s detected *inlier* poses. While this obscures systematic bias across all neural detections in the data set, it nonetheless provides a useful relative analysis of the stability of the neural detector in the presence of varying levels of occlusion. This data set generation process is easily scalable, as all it requires manually is the collection of the video, a once-per-scene annotation of the static ground truth structure, and a per-frame annotation of which neural detections were decently well-localized inliers. Table 5.1 lists the data collected for each scene, while Figure 5-1 shows some representative samples from our data set.

Name	Data	Description
<i>Observed poses</i>	(String, List[Pose])	nVidia Deep Object Pose Estimator neural detections
<i>Ground truth outlier status</i>	(String, List[Bool])	Per-Frame, per-object flag indicating if an object’s neurally generated pose estimate is a noisy outlier
<i>Ground truth scene graph</i>	Scene Graph	Observed static structure, and object poses estimated as the average of inlier detections.

Table 5.1: Description of data collected in our real-world experiments with YCB objects.

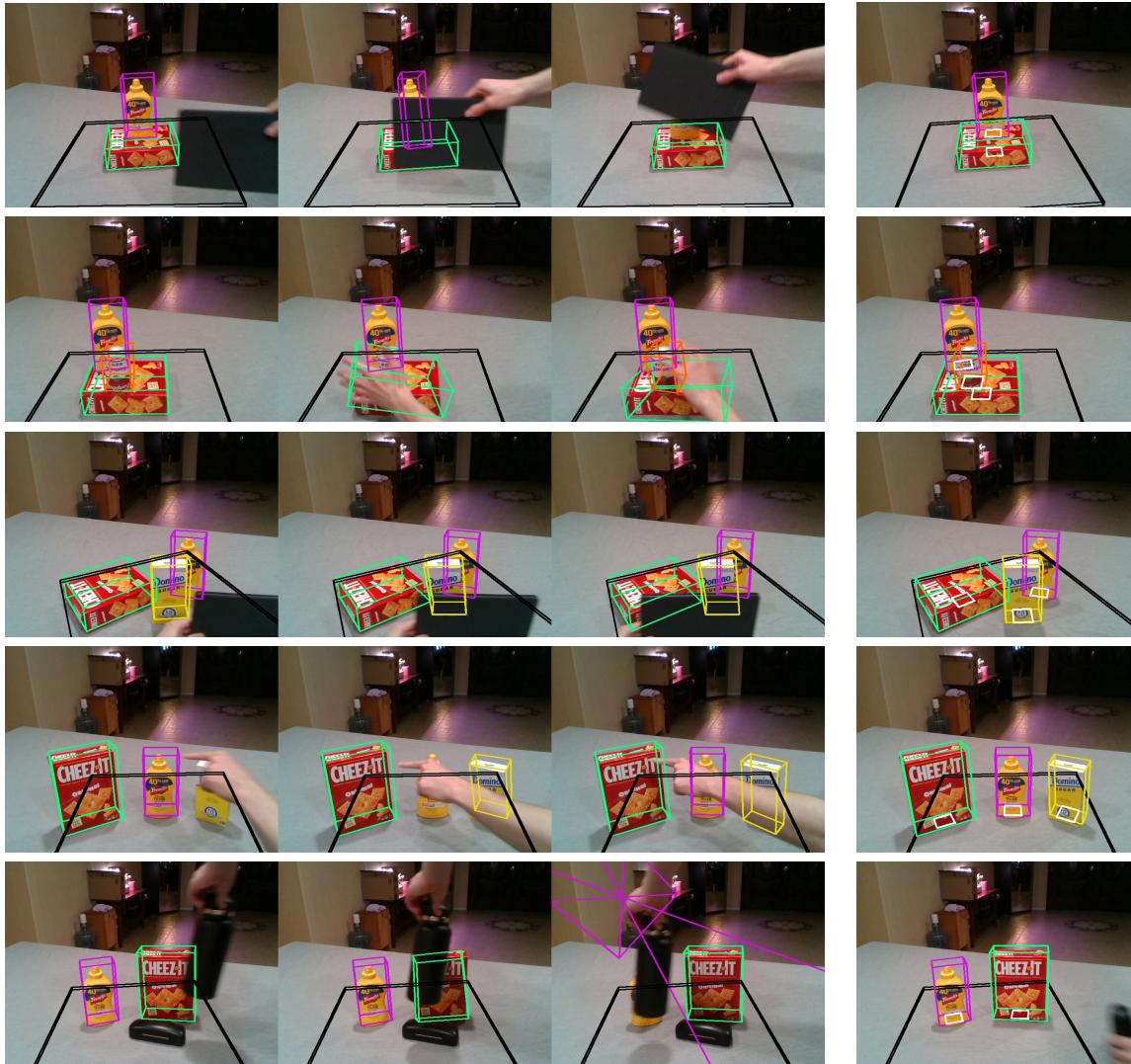


Figure 5-1: A selection of frames from 5 of the 8 captured videos. (Left) We use a dynamic occluder to generate noise in the observed neural network detections, in which objects have their poses estimated incorrectly, or even flicker out of existence. (Right) Each scene is represented by a single static scene graph, with manually annotated structure, and object poses estimated as the average of the inlier neural detections.

5.2 Metrics

In evaluating our scene graph model and inference procedure, we need concrete quantitative metrics that can give us a measure how well each component performs. We propose two concrete metrics that can be used in benchmarking and testing: one for modeling and the other for structure inference.

5.2.1 Modeling: marginal likelihood

The most immediate metric measuring the evidence for our model given the data is the log likelihood. Since we've collected complete data for our model, the likelihood simply ends up being the joint probability density of the trace. In Gen, given a collection of traces $t \in \mathcal{T}$ and hyperparameters θ , the log-likelihood is:

$$\ell(\mathcal{T}; \theta) = \sum_{t \in \mathcal{T}} \log p(t_i; \theta)$$

5.2.2 Structure inference: graph edit distance

When evaluating the performance of structure inference, we would like a measure of the similarity of the most probable inferred structure to the ground truth. This gives a measure of how far our algorithm is from ground truth when tasked with classifying the structure of an unambiguous scene. In particular, we simplify our metric to how well our algorithm correctly determines which objects in a scene are “contacting”. To do this, we reduce the structure of the predicted scene graph G_1 and target scene graph G_2 to a simple set of undirected edges. The graph operations we consider “edits” are edge addition and edge removal. Then, the edit distance is just the size of the symmetric difference $E_1 \Delta E_2$ of the edge sets, or more explicitly

$$\text{Edit}(E_1, E_2) = |E_1 \Delta E_2| = |(E_1 \cup E_2) \setminus (E_1 \cap E_2)|$$

5.3 Learning hyperparameters with gradient ascent

We can perform a data-driven maximum, we can now use gradient-based optimization to automatically tune the static model hyperparameters to our data, and measure the increase in log-likelihood. We select 3 hyperparameters $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \sigma_3)$ to optimize over: the standard deviation $\sigma_1 = \sigma_{\text{inlier}}$ for the observed 3D positions, the standard deviation $\sigma_2 = \sigma_{\text{slack}}$ for the 1D slack offsets, and the standard deviation $\sigma_3 = \sigma_{\text{slidingXY}}$ for the latent (x, y) contact parameters. We'd like to find

$$\max \ell(\mathcal{T}; \boldsymbol{\sigma}) = \sum_{t \in \mathcal{T}} \log p(t; \boldsymbol{\sigma})$$

To do so, we perform a scheduled gradient ascent over the hyperparameters with respect to the objective function. We initialize $\boldsymbol{\sigma}_0 = (0.01, 0.01, 0.25)$ (meters). We run 1,000 iterations of gradient descent, and for each iteration t we update parameters according to

$$\boldsymbol{\sigma}_t = \boldsymbol{\sigma}_{t-1} + \alpha_0 r^t \cdot \nabla \boldsymbol{\sigma}_{t-1}$$

where $\alpha_0 = 1 \times 10^{-8}$ and $r = 0.999$. Gen's dynamic DSL supports automatic calculation of gradients of the joint density evaluated on the conditioned addresses, with respect to model hyperparameters, and for performing updates on parameters with a provided learning rate. Thus our procedure is mostly automated once we have the set of traces \mathcal{T} .

Figure 5-2 shows the results of gradient descent. The algorithm converges rather rapidly for σ_{inlier} and σ_{slack} , but over a longer time we see $\sigma_{\text{slidingXY}}$ begin to converge as well. This longer convergence time suggests that $\sigma_{\text{slidingXY}}$ may have a smaller impact overall on the likelihood compared to the other two parameters.

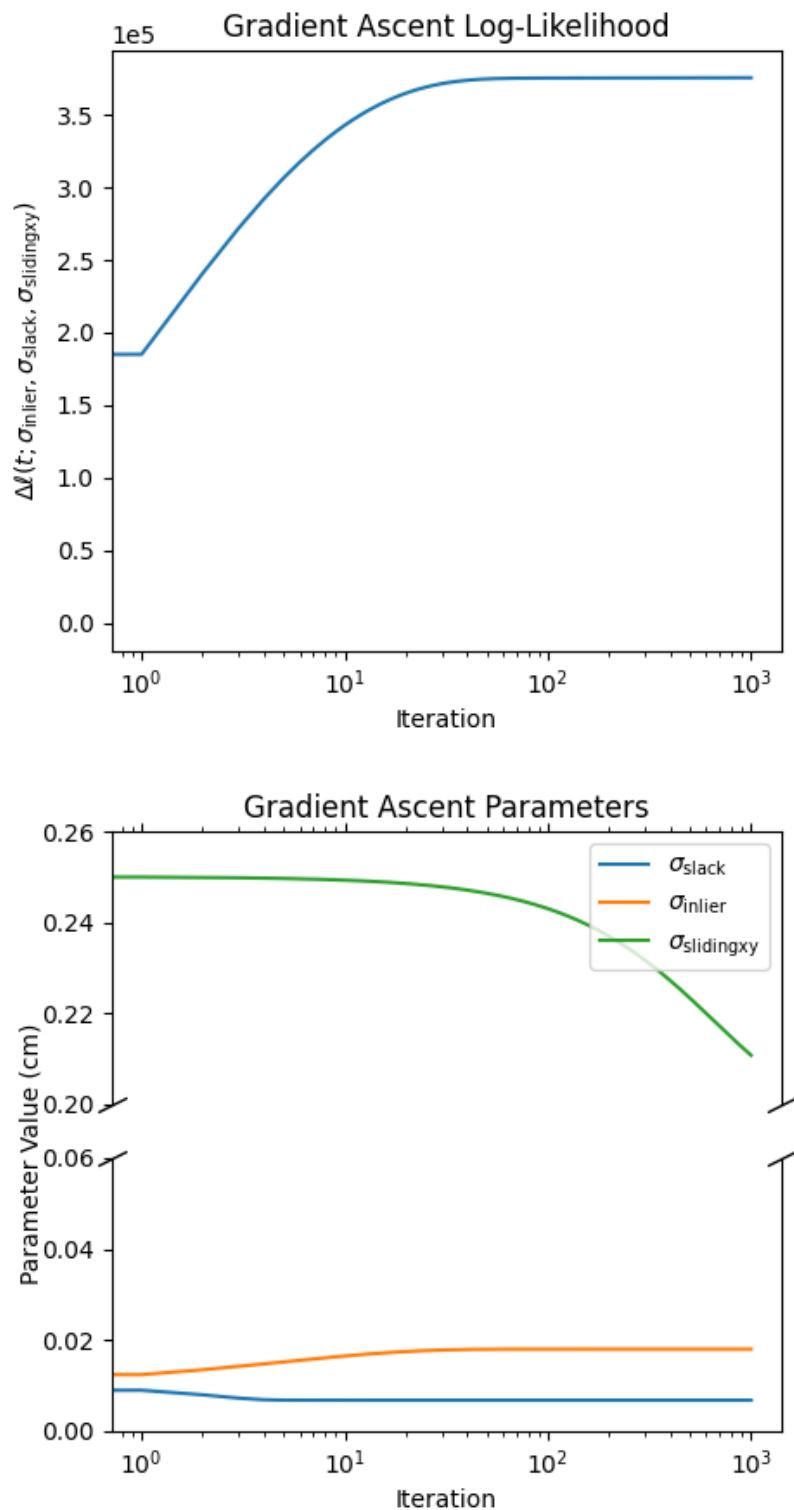


Figure 5-2: Results of gradient ascent on $\boldsymbol{\sigma}$. (Top) shows the change in log likelihood $\Delta \ell(\mathcal{T}; \boldsymbol{\sigma}) = \ell(\mathcal{T}; \boldsymbol{\sigma}_t) - \ell(\mathcal{T}; \boldsymbol{\sigma}_0)$. (Bottom) shows the values of the parameters $\boldsymbol{\sigma}_t$.

5.4 Inlier detection observational model

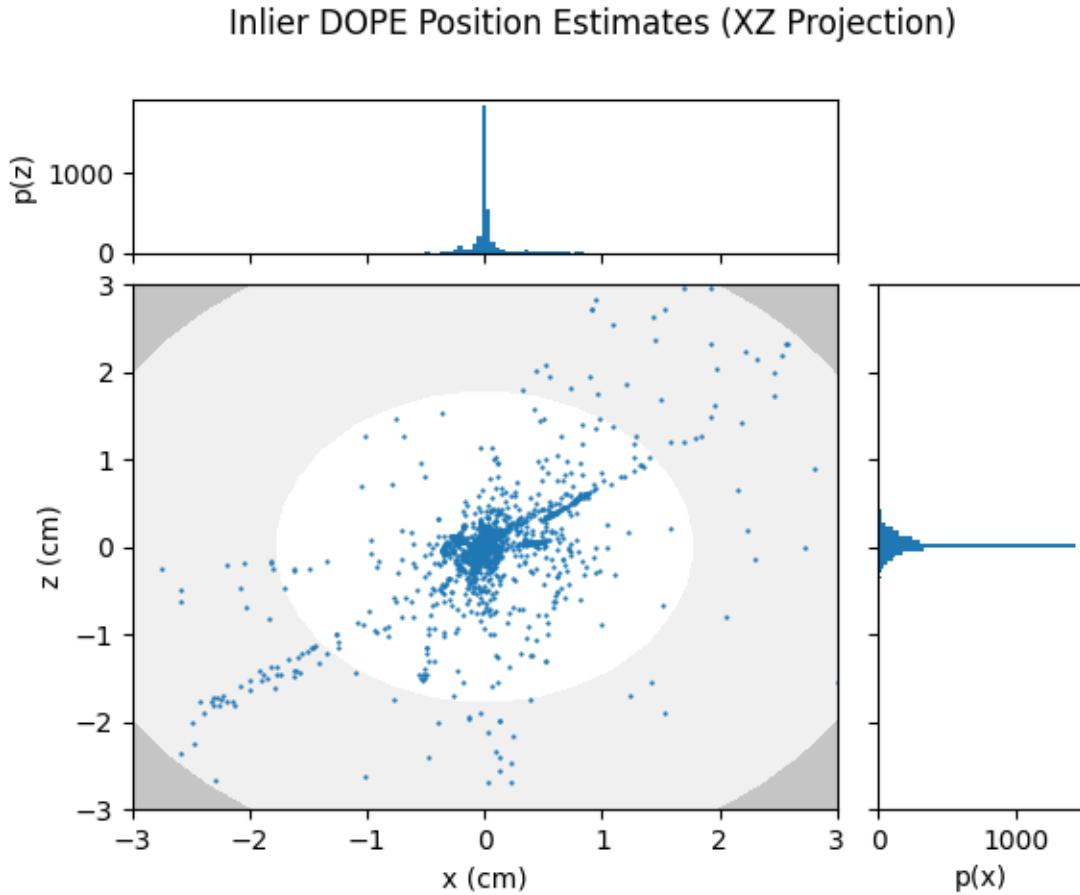


Figure 5-3: Scatter plot of the projection of nVidia DOPE’s observed pose estimates to the x and z position axes, along with corresponding marginal histograms for each axis respectively. The plotted contour is the overlaid noisy observation model for inlier detections (in the projection, a 2D multivariate normal distribution), with the standard deviation hyperparameter set to the maximum-likelihood value for σ_{slack} that we optimized for above.

The lowest-level data in our static model are observations of neural object detections. A-priori, we modeled this data as a simple mixture of an “inlier” and “outlier” distribution. We presupposed the noise for detected inlier positions would be normally distributed, with standard deviation σ_{slack} . In the last section, we jointly fit this hyperparameter with two others, which gives us a tuned version of our inlier model.

Figure 5-3 provides a comparison of a simple observational model in the positional dimensions, fit to maximum likelihood with respect to inlier pose detections. We note

a few details from this plot, and what they suggest for the observational model.

The first is that the distribution systematically overestimates the number of observations in the “middle” part of the space. A large number of detected poses are either highly concentrated around the mean detection, or are much further out in the tails of the distribution. A normal distribution may not be the most accurate model of the behavior of the neural detector; a distribution with heavier tails, such as a Cauchy, is likely to better model of the detected positions.

The second is the presence of “clusters” of erroneous detections that suggest failures in the neural detectors are strongly correlated across time. Indeed, the high concentration of poses around the average inlier detection means that there could be substantial correlated systematic error in an object’s detection across an entire scene, that is not shown in Figure 5-3. For example, the average inlier pose detection for the Domino sugar box presented on the right side of the fourth row in Figure 5-1 is visibly mis-estimated. Assuming that observational noise is falsely uncorrelated can bias our posterior latent pose to be centered around a systematically wrong detection. It may instead be possible to model the correlation between these errors, and use additional information from the image, like the presence of occluders, to obtain a more accurate estimate of the true object pose.

5.5 Testing structure inference

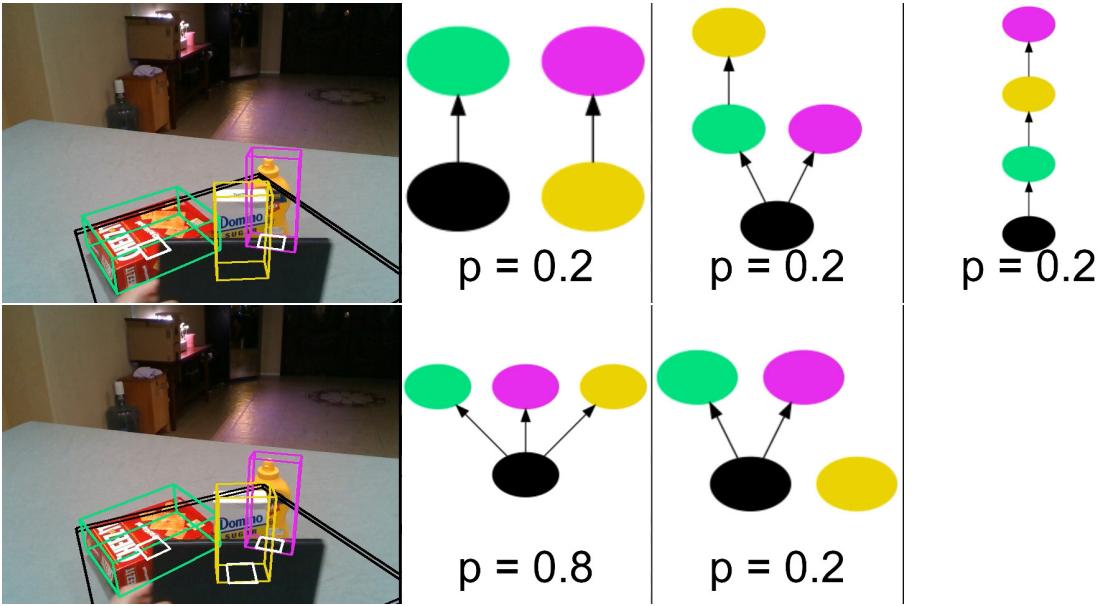


Figure 5-4: Example of the performance on RJMCMC on inferring the structure posterior in a frame from scene #5. (Top) The uniform prior over scene graph structure contains a random assortment of inaccurate classifications. (Bottom) The RJMCMC structure inference procedure predicts the correct ground-truth structure with high probability.

Finally, we use our real-world data to evaluate the performance of our RJMCMC procedure for structure inference, on the posterior for our unoptimized and optimized models. We aim to see how well our procedure is able to classify the ground truth structure, given the correct latent poses of the objects in a scene¹ However, we only have the ground truth poses for 8 unique static scenes, which limits the number of independent trials we can perform.

To ameliorate this, we augment the data with some sampled noise. When performing continuous parameter inference our latent pose posterior will have some non-zero width, meaning we should allow for some variability in the estimated underlying pose, without significantly changing the structure posterior. To simulate this, we augment

¹We omitted extending this gradient search to latent poses, due to an implementation challenge in propagating gradients through custom data structures in Gen. These missing features, if fixed, would allow inferring continuous parameters in the inlier case via gradient-based methods, enabling efficient full scene graph inference.

our data by very slightly perturbing the latent poses for each frame, without qualitatively changing the underlying scene graph that best explains these latent poses; the position is perturbed by gaussian noise sampled with a standard deviation of 0.5 cm, and the orientation is perturbed by VMF noise sampled with concentration parameter 4000. This adds enough variability to give a more robust test of our structure inference algorithm, especially when comparing the unoptimized and optimized models.

For each frame, we set the static model trace to the underlying scene graph in 5 independent particles. For each particle, we then resample the scene graph structure G and discrete parameters Z from a uniform prior, without changing the absolute 6DoF poses or observations for each object. We then sweep our reversible jump structure move across all objects O for 5 iterations. Figure 5-5 shows the graph edit distance between the ground-truth structure and the most frequent structure among the particles, in the unoptimized and optimized case respectively. Figure 5-4 shows an example of the prior structure and inferred posterior respectively for a frame in sequence #5. Inference in both cases consistently finds a structure within two edges of the ground-truth, and most often finds the correct structure.

The optimized procedure performs marginally better than the unoptimized case, although the difference is minimal. Given this, and the fact that our hyperparameters didn't change much in the gradient ascent procedure, suggests that our hand-tuned values were already decent to begin with, at least with respect to the structure posterior. Recall that in our synthetic analysis of these hyperparameters shown in Figure 4-1, the posterior probability of an edge between two objects transitions rather sharply between 0 and 1, depending on the distance of their contacting faces. We might then expect that for a range of "decently-tuned" hyperparameter values, the structure posterior would perform roughly the same, with a sharp decrease in model performance once the hyperparameters fell out of this range.

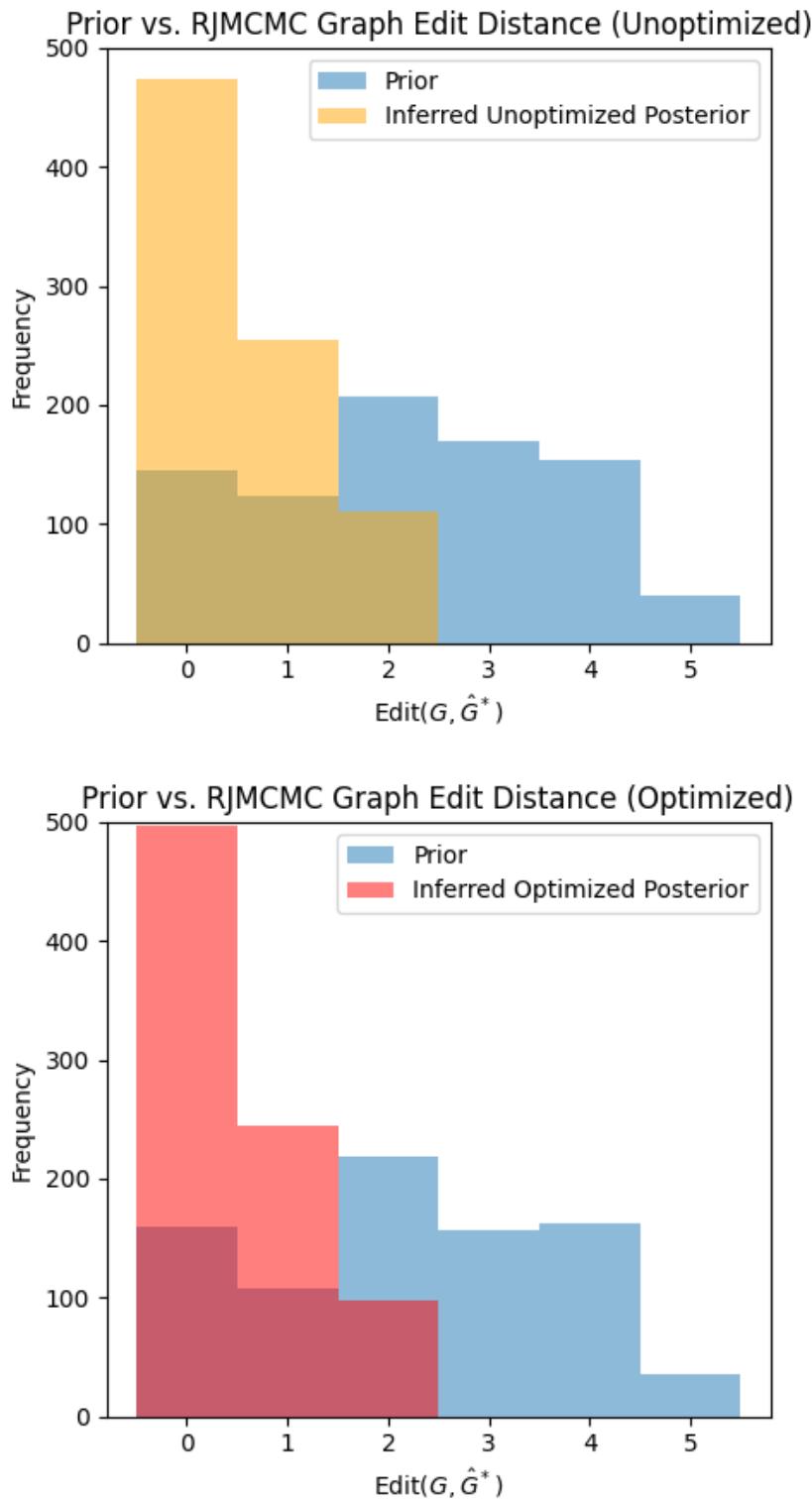


Figure 5-5: Graph edit distance between the ground truth structure G and the mode \hat{G}^* of the inferred structure posterior, for the unoptimized model (top) and optimized model (bottom).

Chapter 6

Conclusion

This thesis has focused extensively on developing utilities and infrastructure for working with generative scene graphs in a practical and principled way. We've shown the implementation of a preliminary dynamic scene graph model, and structure inference procedure in the probabilistic programming system Gen. We've developed capabilities for visualizing and understanding distributions over scene graphs. We've analyzed and benchmarked subcomponents of our scene graph model, demonstrating methods for improving complex generative scene graph programs. Finally, we collected real-world data to train and test the accuracy of scene graph modeling and inference, while

6.1 Future directions

The capabilities developed in this thesis are sufficient for future researchers and engineers to explore a wide range of exciting basic research and applications in 3D generative scene graph modeling and inference. We discuss some future directions that can build off this work in applying generative scene graph models.

Filtering over noisy neural object detections We introduced a novel likelihood that models the output of bottom-up deep neural pose detectors. The accuracy of such a model crucially depends on our ability to infer and correct failures that

are present in neural detections. This work made initial steps toward improving the prior and likelihood models, but robustly inferring clean 3D scene structure from noisy neural detections remains a key challenge, that is now approachable with this infrastructure.

Random scene generation Generating random “realistic” scenes is a key simulation and synthetic data generation problem. Properly tuned scene graph priors can be used to generate random scenes with controllable variability. Combining hyperparameter learning over specifiable semantic constraints on the relationships of objects allows an expressive class of models that can be replicate essential features of real-world scenes. Such data could be used in a wide variety of applications, from synthetic data augmentation, to world generation in video games.

Cognitive architecture for common sense The hypothesis that probabilistic generative models can replicate naturalistic intuition and commonsense reasoning extends from a Bayesian interpretation of cognitive science [10]. This work was completed as part of the Cora project at MIT, whose mission is to develop a system for common-sense reasoning, leveraging modern probabilistic programming systems. With utilities for practically developing and testing scene graph models, we lower the barrier to using them as a central representation in a cognitively-inspired architecture.

Picture 2.0: adding motion and scene graphs Picture offered one of the domain-specific probabilistic programming languages for visual scene perception [8]. The original language focused on simply-structured, closed-universe representations of single-frame scenes. Scene graph models offer the potential for a more extensible scene perception language, expanding the scope of possible models to a dynamic, open-universe that is categorically more expressive.

Bibliography

- [1] Peter W Battaglia, Jessica B Hamrick, and Joshua B Tenenbaum. Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences*, 110(45):18327–18332, 2013.
- [2] Marco F. Cusumano-Towner, Feras A. Saad, Alexander K. Lew, and Vikash K. Mansinghka. Gen: A general-purpose probabilistic programming system with programmable inference. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, pages 221–236, New York, NY, USA, 2019. ACM.
- [3] John Ellson, Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Gordon Woodhull. Graphviz and dynagraph – static and dynamic graph drawing tools. In *GRAPH DRAWING SOFTWARE*, pages 127–148. Springer-Verlag, 2003.
- [4] Peter J Green and David I Hastie. Reversible jump mcmc. *Genetics*, 155(3):1391–1403, 2009.
- [5] Gregory Izatt and Russ Tedrake. Generative modeling of environments with scene grammars and variational inference. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6891–6897. IEEE, 2020.
- [6] Varun Jampani, Sebastian Nowozin, Matthew Loper, and Peter V Gehler. The informed sampler: A discriminative approach to bayesian inference in generative computer vision models. *Computer Vision and Image Understanding*, 136:32–44, 2015.
- [7] Justin Johnson, Agrim Gupta, and Li Fei-Fei. Image generation from scene graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1219–1228, 2018.
- [8] Tejas D Kulkarni, Pushmeet Kohli, Joshua B Tenenbaum, and Vikash Mansinghka. Picture: A probabilistic programming language for scene perception. In *Proceedings of the ieee conference on computer vision and pattern recognition*, pages 4390–4399, 2015.
- [9] Tejas D. Kulkarni, Will Whitney, Pushmeet Kohli, and Joshua B. Tenenbaum. Deep convolutional inverse graphics network. *CoRR*, abs/1503.03167, 2015.

- [10] Thomas L Griffiths, Charles Kemp, and Joshua B Tenenbaum. Bayesian models of cognition. 2008.
- [11] Vikash K Mansinghka, Tejas D Kulkarni, Yura N Perov, and Joshua B Tenenbaum. Approximate bayesian image interpretation using generative probabilistic graphics programs. *arXiv preprint arXiv:1307.0060*, 2013.
- [12] F Landis Markley, Yang Cheng, John L Crassidis, and Yaakov Oshman. Averaging quaternions. *Journal of Guidance, Control, and Dynamics*, 30(4):1193–1197, 2007.
- [13] Pol Moreno, Christopher KI Williams, Charlie Nash, and Pushmeet Kohli. Overcoming occlusion with inverse graphics. In *European Conference on Computer Vision*, pages 170–185. Springer, 2016.
- [14] Moshiko Raboh, Roei Herzig, Jonathan Berant, Gal Chechik, and Amir Globerson. Differentiable scene graphs. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1488–1497, 2020.
- [15] Lukasz Romaszko, Christopher KI Williams, Pol Moreno, and Pushmeet Kohli. Vision-as-inverse-graphics: Obtaining a rich 3d explanation of a scene from a single image. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 851–859, 2017.
- [16] Henry Sowizral. Scene graphs in the new millennium. *IEEE Computer Graphics and Applications*, 20(1):56–57, 2000.
- [17] Jiajun Wu, Erika Lu, Pushmeet Kohli, Bill Freeman, and Josh Tenenbaum. Learning to see physics via visual de-animation. In *NIPS*, pages 153–164, 2017.
- [18] Jiajun Wu, Joshua B Tenenbaum, and Pushmeet Kohli. Neural scene de-rendering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 699–707, 2017.
- [19] Ben Zinberg, Marco F. Cusumano-Towner, and Vikash K. Mansignhka. Structured differentiable models of 3d scenes via generative scene graphs. *NeurIPS*, 2019.