

**Learning to See by Moving:  
Self-supervising 3D Scene Representations for  
Perception, Control, and Visual Reasoning**

Hsiao-Yu Fish Tung

March 2021

CMU-ML-21-100

Machine Learning Department  
School of Computer Science  
Carnegie Mellon University  
Pittsburgh, PA 15213

**Thesis Committee:**

Katerina Fragkiadaki, Chair  
Tom Mitchell  
Chris Atkeson  
Jitendra Malik (Berkeley)

*Submitted in partial fulfillment of the requirements  
for the degree of Doctor of Philosophy.*

**Keywords:** Embodied vision, Geometry-aware Recurrent Networks (GRNNs), Self-supervised learning, Spatial reasoning, Language Grounding, Intuitive Physics, Concept Learning, Manipulation

*For my Family*



## Abstract

We propose learning frameworks for artificial agents to learn several aspects of visual common sense (instantiate and retrieve object concepts, reason about space and 3D geometry, manipulate diverse objects) while moving and interacting with 3D environments. Current state-of-the-art visual systems can achieve human-level object recognition performance on Internet photos, but their performance degrades drastically when applied to videos captured by a moving camera. The performance gap is due to the great difference in the image statistics: in Internet photos, objects are centered, unoccluded, in canonical scales and poses; in photos captured by mobile agents, objects come in a wide variety of scales, poses, locations, and occlusion configurations. How can machines learn to see without relying upon humans to detect and center the interesting content in images and videos?

We explore neural architectures and training schemes for learning visual scene representations that can work under a moving camera, and can exploit the moving camera viewpoint to self-improve without human annotations. This thesis re-visits the paradigm of vision as inference of a 3D scene representation, also known as “vision as inverse graphics”. Nevertheless, instead of inferring explicit 3D representations such as meshes or pointclouds, we infer learnable 3D feature representations from RGB or RGBD inputs. The feature representations can be optimized by training end-to-end with many task objectives, including object detection, view prediction, object dynamics prediction, and object manipulation. The proposed models integrate recent advances in Simultaneous Localization And Mapping (SLAM) and deep learning. Similar to SLAM, our model generates stable 3D scene representations that retain information regarding size, shape, and spatial arrangements of objects, which permit object permanence to emerge across camera viewpoints, despite changes in the field of view. Different from SLAM, which constructs a 3D point cloud map of a scene by piecing together multi-view images, our model learns to infer a complete 3D scene feature map even from a single view. The feature map encodes task-relevant semantic information, much more than just object occupancy or 3D surfaces.

We demonstrate the effectiveness of the proposed differentiable 2D-to-3D feature mapping in multiple tasks, including detecting objects in 3D, predicting 3D object interactions, manipulating diverse objects, recognizing visual concepts, grounding language expressions, and generating 3D scenes that comply with a language utterance. We show the proposed models can self-supervise themselves using unlabelled data and outperform supervised models in the tasks above.

## Acknowledgments

First and foremost, I am extremely grateful for my advisor Katerina Fragkiadaki for her continuous support and guidance throughout my PhD. I thank her for providing me the freedom to work on a diverse range of problems and, at the same time, try to guide me to work on the critical and challenging problems. I also thank her for putting a high bar on the quality of all my presentations and for spending a lot of time helping me improve. Katerina has great connections inside and outside CMU, and she showed strong support when I told her I was thinking about getting a job in academia. She has made a great influence on me in both research and in life.

I sincerely appreciate Jitendra Malik, Tom Mitchell and Chris Atkeson for serving as my thesis committee. Jitendra, Tom and Chris are the most influential people in computer vision, language understanding, robotics, AI, and beyond. It is my honor to have all of them on my committee. I am extremely thankful for their support and valuable feedback on the content of the thesis and the proposed work and the presentation. I especially thank Chris for providing insightful and creative thoughts throughout our collaborations, and his support and help for job search.

I also thank my master advisor, Alex Smola, for inspiring and helping me to pursue a career in research. I would not be what I am today without his help. I was always in awe of his creativity and boldness in generating cool research ideas. I think that still influences the way I do and think about research today.

I am happy to have the opportunity to work with many amazing researchers through my PhD. I am thankful for Abhinav Gupta and David Fouhey for encouraging me on my first project in computer vision. Changing research field is usually a stressful decision for junior PhDs, but, thanks to their help, the process was actually quite exciting and enjoyable. I am fortunate enough to have the chance to intern at openAI with Wojciech Zaremba, Peter Welinder and the whole OpenAI robotics team, from whom I learn about robotics, deep RL and vision for RL. A large portion of my thesis is inspired by the summer project I had been working on. I thank Ersin Yumer, who mentored me during my internship at Adobe, for teaching me many cool things in computer graphics. I thank Sheng Li, my mentor at the intel AI lab, for being very patient to me and teaching me valuable knowledge in computer architecture. I also want to show my great appreciation to my Google mentor, Andrew Dai, for his patience and for teaching me things about SOTA language models. I also thank Phillip Isola, Fuxin Li, Leila Wehbe, Jiajun Wu for helping me and giving me good suggestions when we organized the CVPR common sense workshop together.

During my undergraduate study at National Taiwan University, I had met amazing faculties who inspired me to do research in machine learning. I want to thank Hsien-Tien Lin for designing the first machine learning courses in NTU. I still remember I was stunned by the fact that we can use mathematical equations to simulate “learning,” and I knew this is what I would love to spend a large portion of my life studying. I was fortunate enough to work with Hsien-Tien Lin, Chih-Jen Lin and Shou-De Lin on the KDD cup competitions. They taught us a lot of things about problem solving with machine learning techniques. I also got to know many good friends who were interested in ML, and I am happy that I can occasionally meet some of them in schools, conferences and during internships.

My PhD wouldnt be so amazing without my colleagues in the lab. I thank Adam Harley for

his help on the deadlines, proposals, writings, and all the things we need to do when we organize the CVPR workshop. I thank Zhou Xian for teaching me many things about robots and hardware, and for having deep talks about life and dreams. I feel so lucky to have the chance to work with super talented and energetic junior students, Ricson Chang, Mihir Prahudasai, Shamit Lal, Darshan Patil, Syed Ashar Javed, and Jingyun Yang, who all made significant contribution to this dissertation. I also thank my collaborators during my PhD: William Seto, Liang-Kun Huang, Gaurav Pathak, Ashwini Pokle, Yunchu Zhang, Maximilian Sieb, Emmanouil Antonios Platanios, Fangyu Li, Shrinidhi K. Lakshmikanth, my sister Hsiao-Wei Tung, Dougal J. Sutherland, Heiko Strathmann, Soumyajit De, Aaditya Ramdas, Yining Wang, and Animashree Anandkumar. I thank them for their contributions and for everything they have taught me.

I also want to thank my friends in CMU. I thank Wei-Chiu Ma and Chen-Hsuan Lin, my friends since undergraduate study, for all the long talks about research and life. I thank Po-Wei Wang, Chieh Lin, Wei-Yu Chen, Leqi Liu, Yang Fang and many others, for their caring and for the holiday dinners. I thank Aria Weng for helping me with the CVPR workshop. I also thank my dear friends from SMoLa Lab: Manzil Zaheer, Jay-Yoon Lee, Zichou Yang, Chao-Yuan Wu, Yining Wang, Adams Yu Wei, Mu Li, Yuxiang Wang for teaching me so many things when I was a master student and for showing their supports even after they had graduated. I also thank my close friends since master, Chia-Ying Tsai, Shih-Yun Lo, Ru-Shuan Hua, and Elain Chen, for all the thanksgiving trips and girl talks. I also thank our department Mom, Diane Stidle, for helping me throughout the study, and for all the wonderful events and holiday cards.

The thesis is dedicated to my family, Pao-Chao Tung, Yin-Yin Lin, Hsiao-Wei Tung, Hsiao-Fan Tung, and Hsiao-Yen Tung, for all the years of your love and support. I especially thank Hsiao-Wei Tung for taking care of me during the hardest time during my PhD, and for all the long talks and hugs. I also thank my partner, Andrew Huang, for supporting me and giving me great advice about life.





# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Perception: Learning to see a stable world with objects . . . . .	3
1.2	Action + Physics: Learning to imagine how objects can move and how to interact with them . . . . .	5
1.3	Concept learning: Learning to construct memory and associate current observations with past memory . . . . .	6
1.4	Language understanding: Learning to interpret language through visual simulation	6
1.5	Dissertation structure . . . . .	7
<b>I</b>	<b>Perception: Learning to see a stable world with objects</b>	<b>9</b>
<b>2</b>	<b>Building Embodied Perception with Geometry-Aware Recurrent Networks</b>	<b>11</b>
2.1	Introduction . . . . .	11
2.2	Geometry-aware recurrent networks . . . . .	13
2.2.1	View prediction . . . . .	15
2.2.2	3D object detection and segmentation . . . . .	17
<b>3</b>	<b>Learning to See Moving Objects without 3D Labels</b>	<b>21</b>
3.1	Introduction . . . . .	21
3.2	Semi-supervised learning of 3D object detection . . . . .	22
3.2.1	View-contrastive rendering . . . . .	23
3.2.2	Experiments . . . . .	24
3.3	Unsupervised 3D moving object detection . . . . .	25
3.3.1	Experiments . . . . .	26
<b>II</b>	<b>Action + Physics: Learning to imagine how objects can move and how to interact with them</b>	<b>29</b>
<b>4</b>	<b>Learning View-invariant Intuitive Physics Models for Manipulation</b>	<b>31</b>
4.1	Introduction . . . . .	31
4.2	Object-Factorized Environment Simulators (3D-OES) . . . . .	32
4.2.1	3D Object Graph Neural Networks for Motion Forecasting . . . . .	34

4.2.2	Model Predictive Control with 3D-OES . . . . .	35
4.3	Experiments . . . . .	35
4.3.1	Data collection details . . . . .	37
4.3.2	Action-Conditioned 3D Object Motion Forecasting . . . . .	38
4.3.3	Visualization of the 3D motion predictions . . . . .	39
4.3.4	Neural rendering and counterfactual simulations . . . . .	40
4.3.5	Pushing with Model Predictive Control (MPC) . . . . .	40
4.3.6	Sim-to-Real Transfer . . . . .	43
<b>5</b>	<b>Visually-Grounded Library of Behaviors for Generalizing Manipulation Across Objects, Configurations and Views</b>	<b>45</b>
5.1	Introduction . . . . .	45
5.2	Method . . . . .	47
5.2.1	Visually-Grounded Behavior Selector . . . . .	48
5.2.2	Building a Library of Behaviors . . . . .	50
5.3	Experiments . . . . .	50
5.3.1	Simulation Experiment Setups . . . . .	51
5.3.2	Single Behavior versus a Library of Behaviors . . . . .	52
5.3.3	The necessity of building the selector with the proposed view-invariant and affordance-aware 3D Representations . . . . .	53
5.3.4	Comparison with other grasping baselines . . . . .	53
5.3.5	Real robot results . . . . .	55
<b>III</b>	<b>Concept learning: Learning to construct memory and associate current observations with past memory</b>	<b>57</b>
<b>6</b>	<b>Unsupervised learning of 3D visual Concepts by Corresponding and Quantizing Detected Objects</b>	<b>59</b>
6.1	Introduction . . . . .	59
6.2	3D Quantized-Networks (3DQ-Nets) . . . . .	61
6.2.1	Quantizing objects into prototypes . . . . .	63
6.2.2	Cross-scene 3D correspondence mining . . . . .	64
6.2.3	Iterative learning of object detection, visual features, and clustering . . . . .	65
6.3	Experiments . . . . .	66
6.3.1	Few-shot object category labelling . . . . .	67
6.3.2	Clustering with 3D pose-aware quantization . . . . .	67
6.3.3	3D feature learning with 3D correspondence mining . . . . .	68
6.3.4	Joint training of 3D object detection, feature learning and clustering . . . . .	73
6.3.5	Scene parsing using prototypes . . . . .	73
<b>7</b>	<b>Few-shot Concept learning and VQA with Disentangled 3D concepts</b>	<b>77</b>
7.1	Introduction . . . . .	77
7.2	Disentangling 3D Prototypical Networks (D3DP-Nets) . . . . .	79

7.2.1	Object shape/style disentanglement . . . . .	80
7.2.2	3D disentangled prototype learning . . . . .	81
7.3	Experiments . . . . .	82
7.3.1	Few-shot object shape and style category learning . . . . .	82
7.3.2	Few-shot visual question answering . . . . .	84
7.3.3	3D scene generation from language utterances . . . . .	86

## **IV Language understanding: Learning to interpret language through visual simulation 89**

<b>8</b>	<b>Grounding Language in the Learned Visual simulator</b>	<b>91</b>
8.1	Introduction . . . . .	91
8.2	Language grounding on 3D visual feature representations . . . . .	93
8.2.1	Language-conditioned 3D visual imagination . . . . .	94
8.2.2	Detecting referential expressions in 3D . . . . .	95
8.2.3	Instruction following . . . . .	97
8.3	Experiments . . . . .	98
8.3.1	Language conditioned scene generation . . . . .	99
8.3.2	Affordability inference of natural language utterances . . . . .	99
8.3.3	Detecting spatial referential expressions . . . . .	100
8.3.4	Manipulation instruction following . . . . .	102
<b>9</b>	<b>Conclusion and Future Directions</b>	<b>103</b>
	<b>Bibliography</b>	<b>107</b>



# List of Figures

- 1.1 **What a baby sees when moving.** As the baby moves around in the scene, this can cause objects to come in and out of the field of view, to change size dramatically. There are a lot of occlusions and objects are usually not in the center of the images. Often, we cannot hardly tell what the objects are based on a single image (see the last frame). Although the video is so noisy, somehow the baby can make sense of it. The baby can do a lot things like navigating or playing, and can learn a lot things from this type of noisy inputs. In this thesis, we study how we can build artificial agents that can do the same. . . . .

1.2 **Geometry-Aware Recurrent Networks (GRNNs)** improve the way neural networks can aggregate information across video frames captured under camera motion. During camera motion, objects and their features in the 2D feature maps might present in different locations in the 2D pixel space. GRNNs learn to map these features in different pixel locations to the same location in the 3D feature maps by explicitly estimating the relative camera poses between frames and moving the features according the the camera pose estimation. . . . .

2

3
- 2.1 **Internet vision versus robotic vision.** Pictures taken by humans (top row) (and uploaded on the web) are the *output* of visual perception of a well-trained agent, the human photographer. The content is skillfully framed and the objects appear in canonical scales and poses. Pictures taken by mobile agents, such as a NAO robot during a robot soccer game (bottom row), are the *input* to such visual perception. The objects are often partially occluded and appear in a wide variety of locations, scales and poses. We present recurrent neural architectures for the latter, that integrate visual information over time to piece together the visual story of the scene. . . . .

2.2 **Geometry-aware Recurrent Neural Networks (GRNNs)** integrate visual information over time in a 3D geometrically-consistent deep feature memory of the visual scene. At each frame, RGB images are *unprojected* into corresponding 3D feature tensors, which are oriented to the coordinate frame of the memory map built thus far (2nd row). A 3D convolutional GRU memory is then updated using the egomotion-stabilized features as input. . . . .

11

12

2.3	<b>GRNN architecture.</b> At each time step $t$ , an RGB image $I_t$ is the input to a 2D U-net. The resulting 2D deep feature maps are unprojected to 4D tensors $V_t$ , which in turn are input to a 3D U-net (we do not show the optional combination with unprojected depthmaps for clarity). The resulting 3D deep feature maps $\bar{V}$ are oriented to cancel the relative camera motion between the current viewpoint and the coordinate system of the 3D GRU memory state $\mathbf{M}_{t-1}$ , as estimated by an egomotion estimation module. The resulting oriented 3D deep feature maps $\bar{V}_t$ update the 3D GRU memory state and output $\mathbf{M}_t$ . The updated state of the GRU module is then projected from specific viewpoints and decoded into a corresponding RGB image for view prediction, or fed into a 3D MaskRCNN to predict 3D object bounding boxes and object voxel occupancies. . . . .	13
2.4	<b>View prediction results</b> for the proposed GRNNs and the tower model of Eslami et al. [23]. Columns from left to right show the three input views, the groundtruth image from the query viewpoint, the view predictions for GRNNs and for the tower baseline. The first two rows are from the ShapeNet arrangement test set of [11], the next two rows are from the Shepard-Metzler test set of [23], and the following two rows are from the <i>Rooms-ring-camera</i> dataset also from [23]. The last four rows show generalization to scenes with four objects from the ShapeNet arrangement dataset, while both models were trained only on scenes with two objects. GRNNs outperform the baseline by a large margin and <i>strongly</i> generalize under a varying number of objects. . . . .	17
2.5	<b>Scene arithmetic</b> with GRNNs and the model of Eslami et al. [23] (tower). Each row is a separate "equation". We start with the representation of the scene in the leftmost column, then subtract (the representation of) the scene in the second column, and add the (representation of the) scene in the third column. We decode the resulting representation into an image. The groundtruth image is shown in the forth column. It is much more visually similar to the prediction of GRNNs than to the tower baseline. . . . .	18
2.6	<b>3D object detection and segmentation</b> with GRNNs. In the first and second row on the left we show the input images over time, and their corresponding object detection results for a top view, respectively. Blue voxels denote groundtruth objects and the predicted bounding boxes are shown in <b>red</b> and <b>green</b> . On the right, we show segmentation results for the third time step, visualizing the results from two views. Predicted 3D boxes and their corresponding predicted masks are show in red and green, and we show in blue the corresponding groundtruth. Best seen in color. . . . .	19
3.1	<b>Semi-supervised 3D object detection.</b> Pre-training with view-contrastive prediction improves results, especially when there are few object 3D bounding box	
3.2	<b>3D feature flow and object proposals, in dynamic scenes.</b> Given the input frames on the left, our model estimates dense egomotion-stabilized 3D flow fields, and converts these into object proposals. We visualize colored point-clouds and flow fields in a top-down (bird's eye) view. . . . .	23
3.3	<b>Unsupervised 3D moving object detection with a <i>stationary</i> camera.</b> . . . .	26
		27

3.4	<b>Unsupervised 3D moving object detection with a <i>moving</i> camera</b> . . . . .	27
4.1	<b>3D-OES</b> predict 3D object motion under agent-object and object-object interactions, using a graph neural network over 3D feature maps of detected objects. Node features capture the appearance of an object node and its immediate context, and edge features capture relative 3D locations between two nodes, so the model is translational invariant. After message passing between nodes, the node and edge features are decoded to future 3D rotations and translations for each object. . . . .	33
4.2	<b>Forward unrolling of our dynamics model and the <i>graph-XYZ</i> baseline.</b> Left: pushing. Right: falling. In the top row, we show (randomly sampled) camera views that we use as input to our model. The second row shows the ground-truth motion of the object from the front view. Rows 3, 4 show the predicted object motion from our model and the <i>graph-XYZ</i> baseline from the same front camera viewpoint. Our model better matches the ground-truth object motion than the <i>graph-XYZ</i> baseline. The latter does not capture object appearance in any way. . . . .	40
4.3	<b>Neurally rendered simulation videos from three different views</b> Left: groundtruth simulation videos from the dataset. The simulation is generated by the Bullet Physics Simulation. Right: neurally rendered simulation video from the proposed model. Our model forecasts the future latent feature by explicitly warping the latent 3D feature maps, and we pass these warped latent 3D feature maps through the learned 3D-to-2D image decoder to decode them into human interpretable images. We can render the images from any arbitrary views and the images are consistent across views. . . . .	41
4.4	<b>Neurally rendered simulation videos of counterfactual experiments.</b> The first row shows the ground truth simulation video from the dataset. Only the first frame in this video is used as input to our model to produce the predicted simulations. The second row shows the ground truth simulation from a query view. Note that our model can render images from any arbitrary view. We choose this particular view for better visualization. The third row shows the future prediction from our model given the input image. The following rows show the simulation after manipulating an objects (in the blue box) according the instruction on the left most column. . . . .	42
4.5	<b>Collision-free pushing on a real-world setup.</b> The task is to push a mouse to a target location without colliding into any obstacles. Our robot can successfully complete the task with 3 push attempts. . . . .	43
4.6	Real-world setup with Baxter . . . . .	44
4.7	Objects for real-world experiments . . . . .	44
5.1	We propose a novel policy representation that generalizes to unseen objects and camera views. In contrast to prevalent approaches that learn <i>state-to-action</i> or <i>image-to-action</i> mappings, our proposed model decomposes a policy into a behavior selection module that uses visual observations and a library of behaviors to select from that uses abstract state representations as inputs. . . . .	46

5.2	Overview of the proposed framework. Our model consists of (a) a behavior selector $G$ that learns to map RGB-D images $I$ to an affordance-aware, view-invariant 3D feature space that reflects how objects change by applying a behavior, and (b) a library of behaviors, where each behavior $\pi_i$ can either be a controller or a policy learned via RL. . . . .	48
5.3	Visualization of a sample of behaviors and their corresponding object clusters. In each section of the figure, the behavior described after the arrow is the output of the affordance-aware behavior selector when it takes any of the objects visualized in the section as input. . . . .	54
5.4	Real robot setup (left) and objects used (right). . . . .	55
5.5	<b>Grasping Results on a real robot.</b> The robot can successfully pick up different objects and transfer it to a target location in the air. . . . .	56
6.1	<b>Top: Model overview.</b> Our model takes as input RGB-D images of scenes, and outputs 3D prototypes of the objects. <b>Bottom: Evaluation tasks.</b> (a) Scene parsing: Given a new scene, we match each detected object against the prototypes using a rotation-aware check to infer its identity and pose. (b) Image generation: We visualize prototypes with a pre-trained 3D-to-2D image renderer. (c) Few shot object labelling: Assigning a label to a prototype automatically transfers this label to its assigned instances. . . . .	60
6.2	Architecture for <b>3D Quantized-Networks (3DQ-Nets)</b> . Given multi-view RGB-D images of scenes as input during training, our model learns to map a single RGB-D image to a completed scene 3D feature map at test time, by training for view prediction (b). The model additionally uses cross-scene and cross-object 3D correspondence mining and metric learning, to make the features more discriminative (c). Finally, using these learned features, our model quantizes object instances into a set of pose-canonical 3D prototypes using rotation-aware matching (d). These learned prototypes help improve our object detector by providing confident positive 3D object box labels (e) . . . . .	62
6.3	<b>Cross-scene 3D correspondence mining.</b> (a) We show that our approach relies on part-level correspondences obtained by matching the features of the query region (in pink) to a pool of object-centric 3D features maps. (b) These part-level correspondences are verified based on how well their surrounding voxels match with one another in a spatially consistent manner. (c) Finally we train our 2.5D-to-3D lifting module by doing metric learning using the verified positive regions and randomly sampled negatives. . . . .	64
6.4	<b>Detection improvement over 4 iterations.</b> The first row shows the input image and the proposals of the object detector. The second row shows the annotations assigned to the proposals using the 3D prototype distance and 3D center-surround score. We show that our detector improves over time without any ground truth 3D proposals. . . . .	66
6.5	(a) <b>Unsupervised classification accuracy</b> with varying length of prototype dictionary in CARLA. (b) <b>Scene reconstruction</b> results using the learned prototypes from our model and the baselines. . . . .	68



6.6	<b>3D object retrieval results</b> obtained by retrieving image patch using features learned from different feature learning methods, including rgbocc, rgbocc+vcdict, and rgbocc+3D correspondence mining (3DMine) methods. We visualize the retrieval results on CARLA, BigBIRD, and CLEVR datasets. The green boxes indicate that the retrieved image patches belongs to the same object instance as the query, but is in a different viewpoint. The blue boxes indicate instances with the same ground truth object category labels. . . . .	70
6.7	3D object retrieval results obtained by rgbocc+3D correspondence mining on Replica dataset. . . . .	70
6.8	<b>Patch-Level retrieval results</b> on CARLA, BigBIRD, and CLEVR datasets. For each query-prediction row pair, the first row shows the input RGB images and the second row shows bird's eye view projection of the RGB-D point cloud. The blue patches in the bird's eye view visualizations (2nd row) show the 2D projection of the query/retrieved 3D patch. . . . .	71
6.9	Patch based 3D object retrieval results on Replica dataset. . . . .	71
6.10	<b>Rotational alignment results</b> showing relative pose estimation between two randomly posed RGBs of the same object category. For each of the $3 \times 7$ grids, the first row shows 7 input RGB images of the same object category in different poses. The second row shows the projection of the RGB-D point cloud in a birds eye view. The last row shows the projection of the same RGB-D point but warped to the pose that best matches with the object in the first. Results are shown on CARLA, BigBIRD and CLEVR datasets. . . . .	72
6.11	<b>Real world scene parsing results.</b> . . . .	73
6.12	Scene parsing results for CARLA dataset. . . . .	74
6.13	Scene parsing results for CLEVR dataset. . . . .	75
6.14	Scene parsing results for Replica dataset. . . . .	76
7.1	Given a single image-language example regarding new concepts (e.g., blue and carrot), our model can parse the object into its shape and style codes and ground them with <i>Blue</i> and <i>Carrot</i> labels, respectively. On the right, we show tasks the proposed model can achieve using this grounding.(a) It can detect the object under novel style, novel pose, and in novel scene arrangements and viewpoints. (b) It can detect a new concept like <i>blue broccoli</i> . (c) It can imagine scenes with the new concepts. (d) It can answer complex questions about the scene. . . . .	78
7.2	<b>Architecture for disentangling 3D prototypical networks ( D3DP-Nets).</b> (a) Given multi-view posed RGB-D images of scenes as input during training, our model learns to map a single RGB-D image to a completed scene 3D feature map at test time, by training for view prediction. From the completed 3D scene feature map, our model learns to detect objects from the scene. (b) In each 3D object box, we apply a shape-style disentanglement autoencoder that disentangles the object-centric feature map to a 3D (feature) shape code and a 1D style code. (c) Our model can compose the disentangled representations to generate a novel scene 3D feature map. We urge the readers to refer the video in the supplementary material for an intuitive understanding of the architecture . . . . .	79

7.3	<b>D3DP-VQA Modular Networks.</b> Given a question-image pair and a list of learned prototype dictionaries (left), D3DP-Nets parse the visual scene to object shapes, styles, locations and sizes codes (top-right), while the semantic language parser converts the question to an executable program. The generated program is executed sequentially to answer the question (bottom-right). Note that in order to associate different poses of the same shape (Filter Shape), our model does a rotation-aware search between the indexed prototype and the candidate objects. . . . .	82
7.4	<b>Replica dataset.</b> On the left, we show two objects in different scenes belonging to the same shape category ‘Plant’. On the right, we show two objects belonging to the same style category ‘Cream’. . . . .	83
7.5	t-SNE visualization on styles codes. . . . .	84
7.6	(a) The left scene/question pair is from the in domain test set, and the right scene/question pair is from the one shot test set. The colors, materials, sizes, and spatial relationships tested in both splits are the same. The only difference is that the one shot test set contains shapes the model did not see while training and was only exposed to one example before the testing phase. (b) The prototype images shown to the model before starting the one shot testing phase. . . . .	86
7.7	Generating novel scenes using only a single example for each style and content class. . . . .	87
8.1	<b>Embodied language grounding with implicit 3D visual feature representations.</b> Our model associates utterances with 3D scene feature representations obtained from GRNNs. We map RGB images to 3D scene feature representations and 3D object boxes of the objects present . (column 1). We map an utterance and its dependency tree to object-centric 3D feature maps and cross-object relative 3D offsets using stochastic generative networks (column 2). We map a referential expression to the 3D box of the object referent (column 3). Last, given a placement instruction, we 3D localize the referents in the scene and infer the 3D desired location for the object to be manipulated (column 4). We use predicted location to supply rewards for trajectory optimization of placement policies. . . . .	92
8.2	<b>Mapping language utterances to object-centric appearance tensors and cross-object 3D spatial offsets</b> using conditional <i>what-where</i> generative networks. . .	95
8.3	<b>3D referential object detection.</b> We exhaustively score all possible assignments of noun phrases to detected 3D bounding boxes. Each assignment is scored based on unary appearance scores and pairwise spatial scores, as described in the text. .	96
8.4	<b>Language to scene generation</b> (Rows 1,2,4) and <b>Language to image generation</b> (Row 3) from our model and the model of Deng et al [16] for utterances longer than those encountered at training time. Both our model and the baseline are stochastic, and we sample three generated scenes/images per utterance. . . .	98
8.5	<b>Detecting referential spatial expressions.</b> Given a scene and a referential expression, our model localizes the object being referred to in 3D, while our baseline in 2D. . . . .	101

# List of Tables

2.1	<b>View prediction loss and the standard deviation</b> for the ShapeNet arrangement test set for two-object test scenes. Our model and baseline were trained on scenes that also contain two objects with different object instances. . . . .	16
2.2	Mean Average Precision (mAP) for 3D object detection and 3D segmentation for three different thresholds of Intersection over Union (IoU) (0.75,0.5,0.33) on ShapeNet arrangement test set of [11]. . . . .	19
3.1	<b>CARLA-to-KITTI transferability of view-predictive 3D feature representations.</b> We train a 3D detector module on top of the inferred 3D feature maps $\mathbf{M}$ using KITTI 3D object box annotations . . . . .	24
4.1	<b>3D object motion prediction test error during object pushing in scenes with two objects</b> for 1,3, and 5 timestep prediction horizon. . . . .	39
4.2	<b>3D object motion prediction test error during object falling in scenes with three to four objects</b> for 1,3, and 5 timestep prediction horizon. . . . .	39
4.3	Success rate for pushing objects to target locations. . . . .	43
5.1	Success rates on grasping and pushing unseen objects. . . . .	52
5.2	Success rates on grasping and pushing unseen objects using selector with varying representations. . . . .	53
5.3	Success rates on grasping for unseen objects. . . . .	54
6.1	<b>Few shot object category labelling accuracy</b> . . . . .	67
6.2	<b>Unsupervised classification accuracy</b> with dictionary size of 50 prototypes on CLEVR and BigBIRD datasets. . . . .	68
6.3	<b>Retrieval results (precision@10 nearest neighbors)</b> for different architectures and objectives for 2D and 3D visual representation learning. . . . .	69
6.4	Performance across training EM iterations of our model in CLEVR. Feature learning is measured using the same technique as Table 6.3. Object quantization uses the same measurement technique as Fig. 6.5 (a). Detection performance is measured by meanAP at IoU = 0.5. . . . .	73
7.1	Five & one shot classification accuracy for shape and style concepts in CLEVR [53], Real Veggie, and Replica datasets . . . . .	83
7.2	VQA results with model compared to ablations and 2D baselines in CLEVR [53] dataset. . . . .	85

- 8.1 **Mean average precision for category agnostic region proposals.** Our 3D RPN outperforms the 2D state-of-the-art RPN of Faster R-CNN [100]. . . . . 101
- 8.2 **F1-Score for detecting referential expressions.** Our model greatly outperforms the baseline with both groundtruth and predicted region proposals, especially for novel camera views. . . . . 102
- 8.3 **Success rates for executing instructions regarding object placement.** Policies learnt using costs over 3D configurations much outperform those learnt with costs over 2D configurations. . . . . 102

# Chapter 1

## Introduction

The embodiment hypothesis is the idea that intelligence emerges in the interaction of an agent with an environment [116]. While this is an intriguing hypothesis with compelling evidence from psychological experiments [7, 41, 79], this statement has not been properly linked to results from the machine learning side. Can the fact that we are moving agents and we are embodied in a three-dimensional space affects the way we learn and perceive the world? In this thesis, we attempt to answer this question by creating artificial agents that can develop human-like intelligence through their interactions with the 3D world.

The goal of the thesis is to build machines that can work with and can learn from the data captured by an embodied agent that physically moves in 3D scenes. This thesis focuses on embodied agents that can learn and perceive the world from their visual inputs. When the agents begin to move around, they start to observe sequences of images of the 3D environment around them. In Figure 1.1, we show what an actual embodied agent – a baby – actually sees. The images are less well-framed compared to ones we can find on the Internet. Many images do not even have a target object in the center! However, somehow from these images, the baby can effortlessly recognize the objects, their 3D geometry and properties. Later in his/her childhood, the baby also understands how objects can interact with the other, what tasks it can achieve using these objects, and how it can physically move these objects to complete the tasks.

Building embodied agents that can see, act, reason and continue to improve is challenging since there are multiple crucial modules, e.g., visual/motor/cognitive modules, and these modules are interconnected and can change over time. To start with, our research strategy is to build up an initial vision model that can start parsing the visual data into useful information that might be beneficial for building other modules. There are three key problems we must address: (1) How can we get an initial vision model that can extract useful information about the scene from images? What should be the representations of the scene? (2) How can the agents use the scene representations to improve other modules regarding acting, reasoning and language understanding? (3) How can the development in these modules further improve the agents' visual perception so the whole system can continue to improve?

Current 2D Convolutional Neural Networks (CNNs), although prevalent in visual recognition, are not suitable for building visual perception for these moving agents. While these models perform well in recognizing objects in internet photos where objects are carefully-centered and scaled, their performance drops significantly on these noisy and shaky first-person videos where



Figure 1.1: **What a baby sees when moving.** As the baby moves around in the scene, this can cause objects to come in and out of the field of view, to change size dramatically. There are a lot of occlusions and objects are usually not in the center of the images. Often, we cannot hardly tell what the objects are based on a single image (see the last frame). Although the video is so noisy, somehow the baby can make sense of it. The baby can do a lot things like navigating or playing, and can learn a lot things from this type of noisy inputs. In this thesis, we study how we can build artificial agents that can do the same.

objects are present in a wide variety of scales, poses, locations, occlusion configurations. Additionally, simply recognizing the objects in each individual frame is not enough for the agent to have a holistic understanding of the scene! To be able to interpret and act in the scene, it is critical that the embodied agents can integrate information across frames.

The proposed research provides solutions for building these embodied agents' perceptual capabilities that will empower these agents to see, act, reason, and understand about the physical world more like humans. The first module we will explore is:

- **Perception:** How can objects, scenes, their 3D geometry and semantics emerge from raw images captured by the moving agents?

In this dissertation, we propose novel neural network architectures that improves the way neural networks can learn consistent and structured scene representations from the videos captured by a moving agent. Our key idea is to stabilize the frames captured under different camera poses and fuse them to a joint 3D space to construct a stable 3D scene representation. To demonstrate the importance of stabilizing the frames and maintaining a 3D scene representation, we will show how the representation can further improve the following modules:

- **Action + Physics:** Once the agent can parse the scene into objects, how does it learn to interact with these objects? How does it learn how objects can interact with one another? Can the agent identify the plausibility of an object configuration?
- **Concept learning:** Intelligence is not only about detecting and interacting with individual objects, intelligence is also about constructing memory and making association and analogy between similar instances and configurations. How can the model learn the association? How can the model use the association to bootstrap its learning in language acquisition and manipulation?
- **Language understanding:** Language is a critical media for humans to think and communicate. How can the learned module affect the way we learn and understand language?

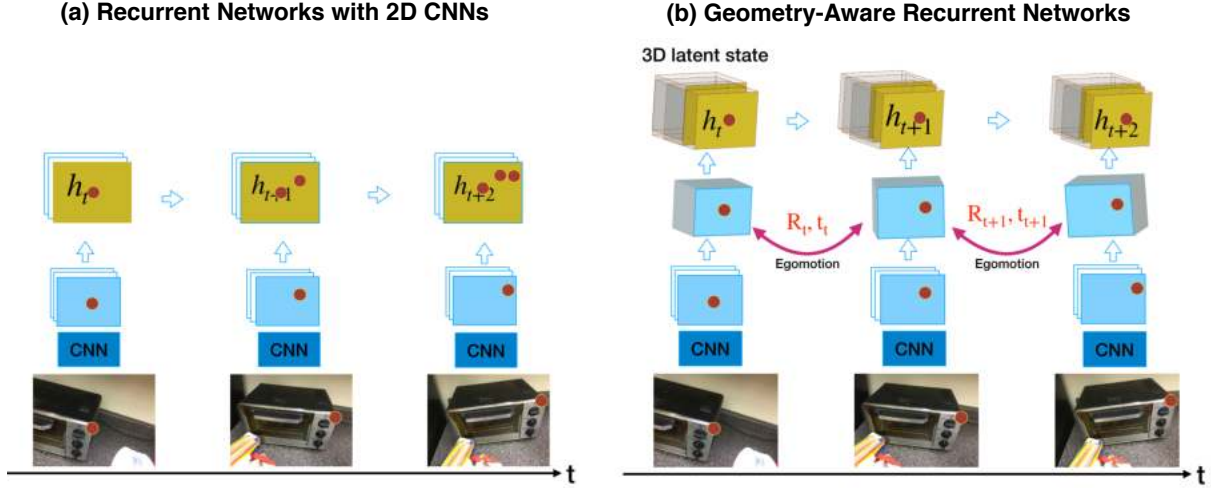


Figure 1.2: **Geometry-Aware Recurrent Networks (GRNNs)** improve the way neural networks can aggregate information across video frames captured under camera motion. During camera motion, objects and their features in the 2D feature maps might present in different locations in the 2D pixel space. GRNNs learn to map these features in different pixel locations to the same location in the 3D feature maps by explicitly estimating the relative camera poses between frames and moving the features according to the camera pose estimation.

## 1.1 Perception: Learning to see a stable world with objects

To integrate and extract information from the video captured by the embodied agent, my solution is to build a 2D-to-3D inverse-graphics engine that can map the video frames into stable 3D representations of the physical scene. Having stable visual representations that do not change during camera motion is beneficial for general scene understanding and reasoning, since the agent can now focus on the change in the scene content as opposed to the change in the camera viewpoints. Aside from stability, our representations can explicitly model objects or scenes as entities living in a three dimensional space, which is critical for planning and manipulation in the 3D world.

Current deep Convolutional Neural Networks (CNNs) are actually not suitable for generating stable and persistent representations from these noisy and shaky video frames from the moving agents. In these videos, the pixel values can change rapidly from frame to frame. Since CNNs directly operate in the pixel space, the activation maps of CNNs activations will fluctuate with the change in the pixel space. In Figure 1.2 (a), we visualize such change. When the camera moves, the activation maps of CNNs also show the objects as moving, even though the objects remains static; when the camera zooms-in and out, they show the objects as becoming larger and smaller; when the camera moves away or a person steps in front of an object, its detection disappears and it is replaced by the objects detected in the new visual frame. CNNs neglect the basic principles of object permanence and spatial awareness that a one-year-old child has developed. The lack of object permanence is particularly problematic for the embodied agent to integrate information over time, to piece together a complete story of a video scene.

To construct the stable representations of a physical scene, we propose Geometry-Aware Recurrent Networks (GRNNs) [136] that disentangle camera motions from scene appearances given an RGB or RGB-D video stream and integrate them into a *persistent* 3D feature map. To achieve such disentanglement, GRNNs learn to estimate camera motion between views and use the estimated camera motion to explicitly move and fuse the features in 3D, similar to many SLAM (Simultaneous Localization and Mapping) methods [84]. The operation ensures features coming from similar 3D location in the real world end up being aggregated and stored in similar 3D location in the 3D feature map. Such operation is implemented as differentiable geometry operations that enables the model to be trained in an end-to-end fashion given a final end task. In Figure 1.2(b), we show how the geometry operations can improve the way neural networks integrate information across frames. Although an object can present in different locations in the 2D pixel space, after the operations, the features for the objects will be mapped to the same location in the 3D feature map.

GRNNs are network architectures that are suitable for the embodied agent to integrate observations across views, but how do we learn their weights? To learn the 3D representation of GRNNs, **we propose to train them in a self-supervised manner using the images and the corresponding camera poses collected by an embodied agent while moving, which enforces the model to learn to complete the missing information from a single view.** Using the data as supervision, GRNN learns to estimate camera motion between frames. Additionally, the agent learns to predict, from a single image, how a scene looks from different viewpoints, which enforces the agent to learn to imagine and complete the missing information in the images. We empirically show GRNNs can predict correctly how an occluded object looks from another view and outperform previous state-of-the-art methods that do not explicitly use geometric operations. The learning component greatly differentiate our model from existing SLAM methods [84] which aim to construct a 3D point cloud map of a scene by piecing together multi-view images. Our model can learn to infer a complete 3D scene feature map even from a single view! In addition, the feature map can encode task-relevant semantic information, much more than just object occupancy or 3D surfaces.

**[Seeing objects and their shapes]** To understand a visual scene, one critical step is to detect objects present in the scene and recognize their 3D shapes. Building upon the learned self-supervised 3D scene representations, we further introduce modules that learn to detect objects in the scene. Since the 3D scene feature maps are complete, i.e., objects features are complete despite occlusions or camera viewpoints, learning a detector in this 3D scene feature space is easier. We show GRNNs perform and generalize well in 3D object detection and 3D shape estimation comparing to previous 2D CNN-based methods [136]. We further show GRNNs can improve their object detection performance by co-learning this with view prediction objective and can learn to detect objects without the need for any 3D annotation [35].



## 1.2 Action + Physics: Learning to imagine how objects can move and how to interact with them

Beyond passively observing scenes and detecting objects, an agent would need to actively interact with the environment to collect more interesting and informative data. To intelligently act, it is critical to know how objects can move, and how they can interact with each others. In my research, I explore how the learned visual representations can aid the learning of object dynamics and manipulation.

To learn how objects can move, we enforce the model to learn a forward physics simulator in the 3D feature space inferred from input images [137]. Our simulator can simulate how the detected objects move in response to an action, and can identify whether an object configuration is physically plausible or not. Our model has several advantages over existing image-based object dynamics models. First, since the 3D feature representation remains persistent across camera views, the resulting dynamics model also remains persistent across views: one can use images captured from any camera viewpoint as input and the resulting prediction is consistent. Current image-based methods are sensitive to viewpoint changes and can easily break when testing on unseen views. Second, we empirically show that our model can predict much more accurate object dynamics compared to methods that use 2D visual features. Additionally, our model is more interpretable: we can render a video that reflects the physics simulation in the 3D feature representation using the learned view prediction module. Lastly, in the 3D representation space, it is easy to check whether two objects intersect in 3D by checking whether their 3D occupancy masks overlap. With this simple rule, our model can correctly infer the feasibility of an object configuration.

With the learned dynamics model, we show that the agent can successfully bring objects to target locations [137]. The model encodes visual features so it can handle objects with varying shapes and colors. More importantly, since our model can infer 3D object interpenetration, our model generalizes to tasks that involve obstacles to avoid. we have empirically shown our model outperforms existing state-of-the-art methods, and can successfully transfer to a real robot platform.

While learning object dynamics is an important component for the agent to plan, manipulate and conduct spatial reasoning, learning general object dynamics is still challenging. Can our agents still learn to manipulate diverse objects even without explicitly modeling the underlying object dynamics? How can we use the 3D feature representations to aid the learning in this setup?

To manipulate objects with diverse appearances and poses, we propose models that learn to compose a library of expert policies (behaviors) where each expert policy only works on a subset of objects. Our models learn to use the 3D feature presentations as retrieval keys to select a behavior that will work under the current scene. While another straightforward approach will be to learn a direct mapping from the visual representations to actions, as suggested by previous works in model-free visuo-motor policy learning [67], we found this approach is ineffective in learning a successful policy due to the huge computational bottleneck introduced by the 3D feature representations. Thus, instead of learning to regress from the 3D feature representations to the target action, we use the 3D feature representations to retrieval scene that is close to the current scene and apply the corresponding expert policy. We show the resulting models can

handle diverse objects, arrangements, and input views, even without explicitly modeling the underlying object dynamics. We further validate the effectiveness of the proposed model on a real robot and show it can learn to push and pick-and-place objects of diverse appearance and poses.

### **1.3 Concept learning: Learning to construct memory and associate current observations with past memory**

Intelligence is not only about detecting and interacting with individual objects, but also about forming memory and making association and analogy between similar instances and configurations. In this thesis, we also explore how the 3D feature representation can improve the way models learn about visual concepts and instance association.

We propose to learn visual concepts through associating objects with their 3D feature representation inferred from images [93]. With the learned 3D feature representation, the model can identify the same object or similar objects under varying scales, poses from images captured from varying viewpoint: since our machine has object permanence, its perception is not affected by the camera viewpoint change; since the visual representation is 3D-aware, the 3D feature maps of two objects can be rotated and scaled appropriately before their features are compared to handle object instances in a variety of 3D poses and scales. We show, without any human labels, the model can learn to correctly associate similar objects and form visual concepts. When given one example of a novel object, our model can recognize it when it presents in unseen poses and locations. From a few labelled examples, the model can learn to name objects in unlabelled scenes.

How effectively we can use existing knowledge depends much on how well we organize them. To acquire new knowledge or concepts more efficiently, we need to improve the way we organize existing knowledge. To achieve this, we propose to further factorize the learned whole-object visual concepts into shapes, color, texture parts, and so on [95]. By splitting concepts into smaller entities, the model can discover more shared concepts across object instances, and can learn new concepts efficiently by exploiting the combinations between these factorized concepts. We show our model can efficiently learn concepts about shapes and color from a few labelled samples, and can use the learned concepts to conduct complex visual question answering.

### **1.4 Language understanding: Learning to interpret language through visual simulation**

Language is a critical media for humans to think and communicate. If the agents can correctly interpret human languages, then they can learn more efficiently through human instructions and documentations. How can the learned 3D visual representations, dynamic models, and concepts affect the way the agent can learn and interpret language meaning?

Humans understand language regarding a scene through visual simulation. Consider the following two sentences: “He used the newspaper to protect his face from the wind.” and “He

used the matchbox to protect his face from the wind.” Although both sentences are grammatically and syntactically identical, we interpret them differently. From the first sentence, most humans can mentally imagine a visual simulation of the scene, and the simulation will allow us to answer millions of questions such as: “Does the man use two hands or one?” “Is the newspaper folded?” “Is the man holding the newspaper?”

Inspired by the way humans learn, we propose a ground language in the developed 3D feature space where we can simulate how objects can move, and can reason about the physical plausibility of a configuration [94]. In 3D, the model can reason about the physical plausibility of a configuration by checking 3D non-intersection and object affordance. We introduce a language grounding model that learns a mapping between 3d feature tensors and natural language utterance. We have empirically shown that the models can successfully infer plausibility and implausibility of statements, localize object referents robust to camera viewpoint, guide object placement policies from natural language instructions, and outperform the existing 2D models by a large margin in all the above tasks.

## 1.5 Dissertation structure

This thesis consists of four main parts regarding four key modules we wish an embodied agent can have: perception, action, concept learning, and language understanding. The contributions of this thesis are as follows:

**Part I** is about perception – we show how we can establish an initial visual perception module that can start parsing image data collected by these moving agents into a persistent scene representation with objects in it.

In Chapter 2, we introduce the key neural architectures, GRNNs, for parsing videos captured by moving agents. We empirically evaluate the models in 3D object detection and novel view prediction. In both tasks, we show that GRNNs outperform previous state-of-the-art methods that do not explicitly use geometric operations and can generalize to complicated scenes with more objects while previous work fails. This chapter was previously published as Tung et al. [136].

In Chapter 3, we further show how GRNNs can improve their object detection performance by co-learning with a self-supervised view prediction objective. We further explore unsupervised methods using GRNNs that can segment moving objects and learn to detect them without the need for any 3D annotations. This chapter was previously published as Harley et al. [35].

**Part II** is about learning object dynamics and manipulation – beyond object detection, we further extend the model to understand how objects move and how the agents can interact with these objects.

In Chapter 4, we propose dynamic models that learn on top the persistent 3D scene representation. We show the models outperform previous vision-based dynamics models by a large margin, and can generalize to images viewpoints outside the training distribution. We further deploy the learned dynamics model on a real robotics platform and we show the models learned in the simulation can directly transfer to real. This chapter was previously published as Tung et al. [137].

In Chapter 5, we extend our learning framework to learn to manipulate diverse objects even without explicitly modeling the underlying object dynamics. To handle objects with diverse appearance and poses, we propose models that learn to compose a library of expert policies (behaviors) where each expert policy works only on a subset of objects/poses. To select a behavior that would work under the current scene, our models learn to retrieve similar scene and corresponding behavior using the learned 3D feature presentations. We show the resulting models can handle diverse objects, arrangements, and input views. We further validate the effectiveness of the proposed model on a real robot and show that it can learn to pick-and-place objects of diverse appearance.

**Part III** is about learning visual concepts – we show how the agents can learn to construct visual memory and associate observed objects with previously seen objects

In Chapter 6, we propose a few-shot concept learning module that learns to associate and group detected objects into clusters by comparing them in the learned persistent 3D feature space. We show our model can learn a new concept from a few labels by propagating the labels to instances in the same cluster. This chapter was previously published as Prabhudesai et al. [93].

In Chapter 7, we extend the concept learning module to further factorize object representation into attributes, and learn concepts on top of these attributes. We show our model can learn new concepts and answer complex questions regarding visual inputs, from a few labelled images. This chapter was previously published as Prabhudesai et al. [95].

**Part IV** is about language understanding – we show how the agents can understand language and their affordability by grounding language in the learned 3D feature representation space that supports spatial understanding and dynamics simulation.

In Chapter 8, we show how grounding language in the 3D feature space can improve the way machines understand language. These models can successfully infer plausibility and implausibility of statements, localize object referents, guide object placement policies from natural language instructions, and outperform existing 2D models by a large margin in all the above tasks. This chapter was previously published as Prabhudesai et al. [94].

We conclude and discuss future directions in Chapter 9

## **Part I**

# **Perception: Learning to see a stable world with objects**



# Chapter 2

## Building Embodied Perception with Geometry-Aware Recurrent Networks

### 2.1 Introduction

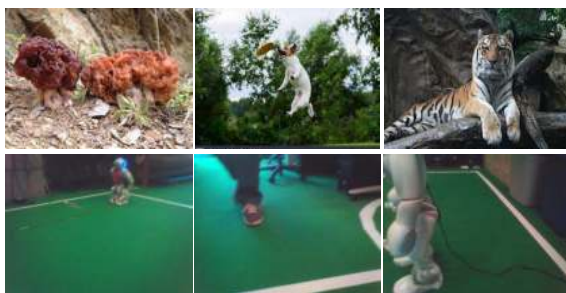


Figure 2.1: **Internet vision versus robotic vision.** Pictures taken by humans (top row) (and uploaded on the web) are the *output* of visual perception of a well-trained agent, the human photographer. The content is skillfully framed and the objects appear in canonical scales and poses. Pictures taken by mobile agents, such as a NAO robot during a robot soccer game (bottom row), are the *input* to such visual perception. The objects are often partially occluded and appear in a wide variety of locations, scales and poses. We present recurrent neural architectures for the latter, that integrate visual information over time to piece together the visual story of the scene.

Current state-of-the-art visual systems [38] accurately detect object categories that are rare and unfamiliar to many of us, such as *gyromitra*, a particular genus of mushroom (Figure 2.1 top left). Yet, they neglect the basic principles of object permanence or spatial awareness that a one-year-old child has developed: once the camera turns away, or a person walks in front of the *gyromitra*, its detection disappears and it is replaced by the objects detected in the new visual frame. We believe the ability of current visual systems to detect rare and exquisite object categories and their inability to carry out elementary spatial reasoning is due to the fact that

<sup>0</sup>This chapter is based on the paper published previously at CVPR 2019 [136].

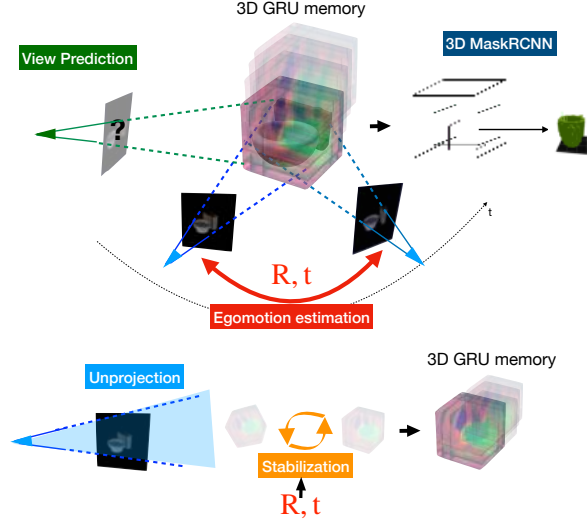


Figure 2.2: **Geometry-aware Recurrent Neural Networks (GRNNs)** integrate visual information over time in a 3D geometrically-consistent deep feature memory of the visual scene. At each frame, RGB images are *unprojected* into corresponding 3D feature tensors, which are oriented to the coordinate frame of the memory map built thus far (2nd row). A 3D convolutional GRU memory is then updated using the egomotion-stabilized features as input.

they are trained to *label object categories* from *static Internet photos* (in ImageNet and COCO datasets) using a *single frame* as input. Our overexposure to Internet photos makes us forget how pictures captured by mobile agents look. Consider Figure 2.1. Internet photos are skillfully captured by human photographers, are well framed and show objects unoccluded, in canonical locations, scales and poses (top row). Instead, photos captured by NAO robots during a soccer game show objects in a wide variety of scales, poses, locations, and occlusion configurations (bottom row). Often, it would not even make sense to label objects in such images, as most objects appear only half-visible. In the case of Internet vision, the picture is the **output** of visual perception of a well-trained visual agent, the human photographer. In the case of mobile robotic vision, the picture is the **input** to such visual perception. Thus, different architectures may be needed for each.

We present Geometry-aware Recurrent Neural Network architectures, which we call GRNNs, that learn to “lift” and integrate over time 2D image features into 3D feature maps of the scene, while stabilizing against the egomotion of the agent. They update over time a 3-dimensional latent feature state: the latent feature vectors are arranged in a 3D grid, where every location of the grid encodes a 3D physical location in the scene. The latent state is updated with each new input frame using egomotion-stabilized convolutions, as shown in Figure 6.1. GRNNs learn to map 2D input visual features to a 3D latent feature map, and back, in a differentiable manner. To achieve such differentiable and geometrically-consistent mapping between the world scene and the 3D latent feature state, they are equipped with differentiable geometric operations, such as egomotion estimation and feature stabilization, 3D-to-2D projection, and 2D-to-3D unprojection, as shown in Figure 6.1. Beyond being space-aware, we do not impose any other constraints on the learned representations: they are free to encode whatever is relevant for the downstream task.





where  $f$  is the focal length of the camera. Then,  $V_{i,j,k,:}^I$  is filled with the bilinearly interpolated 2D feature vector at that pixel location  $(x, y)$ . All voxels lying along the same ray casted from the camera center will be filled with nearly the same image feature vectors. We further unproject the input 2D depthmap  $D_t$  into a binary voxel occupancy grid  $V_t^D \in \{0, 1\}^{w \times h \times d}$  that contains the thin shell of voxels directly visible from the current camera view. We compute this by filling all voxels whose unprojected depth value equals the grid depth value. When a depth sensor is not available, we learn to estimate the depthmap using a 2D U-net that takes the RGB image as input.

We multiply each 3-dimensional channel of the feature tensor  $V_t^I$  with the binary occupancy grid  $V_t^D$  to get a final 4D feature tensor  $V_t \in \mathbb{R}^{w \times h \times d \times c}$ . The unprojected tensor  $V_t$  enters a 3D encoder-decoder network with skip connections (3D U-net) to produce a resulting feature tensor  $\bar{V}_t \in \mathbb{R}^{w \times h \times d \times c}$ .

**Egomotion estimation and stabilization** Our model orients the 3D feature memory to have  $0^\circ$  elevation using the absolute elevation angle of the first camera view. We assume this value is given, but it can also be estimated using a 2D convnet. This essentially makes the memory to always be *parallel to the ground plane*. The azimuth of the 3D feature memory is chosen to be the azimuth of the first view in the input frame sequence. We assume the camera does not translate, only rotates by varying two degrees of freedom, elevation and azimuth.

At each time step  $t$ , we estimate the relative elevation and azimuth between the current frame’s viewpoint and the feature memory. Note that we can alternatively predict the (absolute) elevation directly from each input view, without matching against the memory built thus far. For the azimuth, since we need to estimate the relative azimuth to the first view, such cross-view comparison is necessary. Specifically, the tensor  $\bar{V}_t$  is *rotated* by different azimuth and elevation angles and results in a stack of rotated feature tensors  $\bar{V}^{\text{rot}} \in \mathbb{R}^{(L \cdot K) \times w \times h \times d \times c}$ , where  $L, K$  are the total number of azimuths and elevation angles considered, respectively, after discretization. Similar to the bilinear interpolation used during unprojection, to fill in each feature voxel in a rotated tensor  $\bar{V}_{i,j,k,:}^{\text{rot}}$ , we compute the 3D location  $(X, Y, Z)$  where it is rotated from and insert the bilinearly interpolated feature value from the original tensor  $\bar{V}_t$ . We then compare each of the rotated feature maps with our current 3D feature memory  $\mathbf{M}_{t-1} \in \mathbb{R}^{w \times h \times d \times c}$  using matrix inner products, to produce a probability distribution over azimuth and elevation pairs:

$$\begin{aligned}\bar{\rho}_t(r) &= \mathbf{M}_{t-1} * \bar{V}_{\text{rot}}(r, :, :, :, :), \quad r \in 1 \cdots L \cdot K \\ \rho_t &= \text{softmax}(\bar{\rho}_t),\end{aligned}$$

where  $*$  denotes matrix inner product. The resulting rotation  $\bar{r}_t$  is obtained by a weighted average of azimuth and elevation angles where weights are in  $\rho_t$ . Finally, we orient the tensor  $\bar{V}_t$  to cancel the relative rotation  $\bar{r}_t$  with respect to our 3D memory  $\mathbf{M}_{t-1}$ , we denote the oriented tensor as  $\bar{V}'_t$ .

**Recurrent map update** Once the feature tensor has been properly oriented, we feed  $\bar{V}'_t$  as input to a 3D convolutional Gated Recurrent Unit [12] layer, whose hidden state is the memory  $\mathbf{M}_{t-1} \in \mathbb{R}^{w \times h \times d \times c}$ , as shown in Figure 5.2. This state update outputs  $\mathbf{M}_t$ . The hidden state is initialized to zero at the beginning of the frame sequence. For our view prediction experiments

where we use a fixed number of views  $T$ , we found that averaging, namely  $\mathbf{M}_T = \frac{1}{T} \sum_t \bar{\bar{V}}'_t$  works equally well to using the GRU update equations, while being much faster.

**Projection and decoding** Given a 3D feature memory  $\mathbf{M}_t$  and a desired viewpoint  $q$ , we first rotate the 3D feature memory so that its depth axis is aligned with the query camera axis. We then generate for each depth value  $k$  a corresponding projected feature map  $p_k \in \mathbb{R}^{w \times h \times c}$ . Specifically, for each depth value, the projected feature vector at a pixel location  $(x, y)$  is computed by first obtaining the 3D location it is projected from and then inserting bilinearly interpolated value from the corresponding slice of the 4D tensor  $\mathbf{M}$ . In this way, we obtain  $d$  different projected maps, each of dimension  $w \times h \times c$ . Depth ranges from  $D - 1$  to  $D + 1$ , where  $D$  is the distance to the center of the feature map, and are equally spaced.

Note that we do not attempt to determine visibility of features at this projection stage. The stack of projected maps is processed by 2D convolutional operations and is decoded using a residual convLSTM decoder, similar to the one proposed in [23], to an RGB image. We do not supervise visibility directly. The network implicitly learns to determine visibility and to choose appropriate depth slices from the stack of projected feature maps.

### 2.2.1 View prediction

Mobile agents have access to their egomotion, and can observe sensory outcomes of their motions and interactions. Training sensory representations to predict such outcomes is a useful form of supervision, free of human annotations, often termed *self-supervision* since the “labels” are provided by the embodied agent herself. Can spatial common sense, the notion of objects and scenes, geometry, visibility and occlusion relationships, emerge in a self-supervised way in a mobile agent that moves around and observes the world?

We train GRNNs to predict the image the agent would see from a novel viewpoint, given a short view sequence as input. Given the 3D feature memory and a query viewpoint, we orient the map to the query viewpoint, we project it to 2D and decode it to an RGB image, as described above. We train our view prediction using a standard cross-entropy pixel matching loss, where the pixel intensity has been squashed into the range  $[0, 1]$ . To test the how GRNNs perform in this task, we consider the following simulation datasets:

- i) *ShapeNet arrangement* from [11] that contains scenes with synthetic 3D object models from ShapeNet [9] arranged on a table surface. The objects in this dataset belong to four object categories, namely, cups, bowls, helmets and cameras. We follow the same train/test split of ShapeNet [9] so that object instances which appear in the training scenes do not appear in the test scenes. Each scene contains two objects, and each image is rendered from a viewing sphere which has  $3 \times 18$  possible views with 3 camera elevations ( $20^\circ, 40^\circ, 60^\circ$ ) and 18 azimuths ( $0^\circ, 20^\circ, \dots, 340^\circ$ ). There are 300 different scenes in the training set and 32 scenes with novel objects in the test set.
- ii) *Shepard-metzler shapes* dataset from [23] that contains scenes with seven colored cubes stuck together in random arrangements. We use the train and test split of [23].
- iii) *Rooms-ring-camera* dataset from [23] that contains rooms with random floor and wall colors, in which there are variable numbers of objects with different shapes and colors.

We compare GRNNs against the recent "tower" architecture of Eslami et al. [23], a 2D network trained under a similar view prediction loss. At each time step, the tower architecture takes as input a 2D RGB image and performs a series of convolutions on it. The camera pose from which the image was taken is tiled along the width and height axes and then concatenated with the feature map after the third convolution. Finally, the feature maps from all views are combined via average pooling. Both our model and the baseline use the same autoregressive decoder network. For fairness of comparison, we use groundtruth egomotion rather than estimated egomotion in all view prediction experiments, and only RGB input (no depth input or depth estimation) for both our model and the tower baseline. In both the baseline and our model, we did not use any stochastic units for simplicity and speed of training. Adding stochastic units in both is part of our future work.

Test results from our model and baseline on test images of ShapeNet arrangements and Shepard-metzler datasets are shown in Figure 2.4. Reconstruction test error for the ShapeNet arrangement test set is shown in Table 2.1. GRNNs have a much lower reconstruction test error than the tower baseline. In Figure 2.4, in the first four rows, the distribution of the test scenes matches the training scene distribution. Our model outperforms the baseline in visual fidelity. In Figure 2.4, in the last four rows, the test scene distribution does not match the training one: we test our model and baseline on **scenes with four objects, while both models are trained on scenes with exactly two objects**. In this case, our model shows *strong generalization* and outperforms by a margin the geometry-unaware baseline of [23], the latter refuses to see more than two objects present. We argue the ability to spatially reason should not be affected by the number of objects present in the scene. Our results suggest that geometry-unaware models may be merely memorizing views with small interpolation capabilities, as opposed to learning to spatially reason.

**Scene arithmetics** The learnt representations of GRNNs are capable of scene arithmetics, as we show in Figure 2.5. The ability to add and subtract individual objects from 3D scenes just by adding and subtracting their corresponding latent representations demonstrates that our model disentangles what from where. In other words, our model learns to store object-specific information in the regions of the memory which correspond to the spatial location of the corresponding object in the scene.

	Tower (Baseline)	GRNNs (Ours)
ShapeNet	$0.109 \pm 0.029$	<b><math>0.084 \pm 0.017</math></b>
Shepard-Metzler	$0.081 \pm 0.017$	<b><math>0.073 \pm 0.014</math></b>

Table 2.1: **View prediction loss and the standard deviation** for the ShapeNet arrangement test set for two-object test scenes. Our model and baseline were trained on scenes that also contain two objects with different object instances.

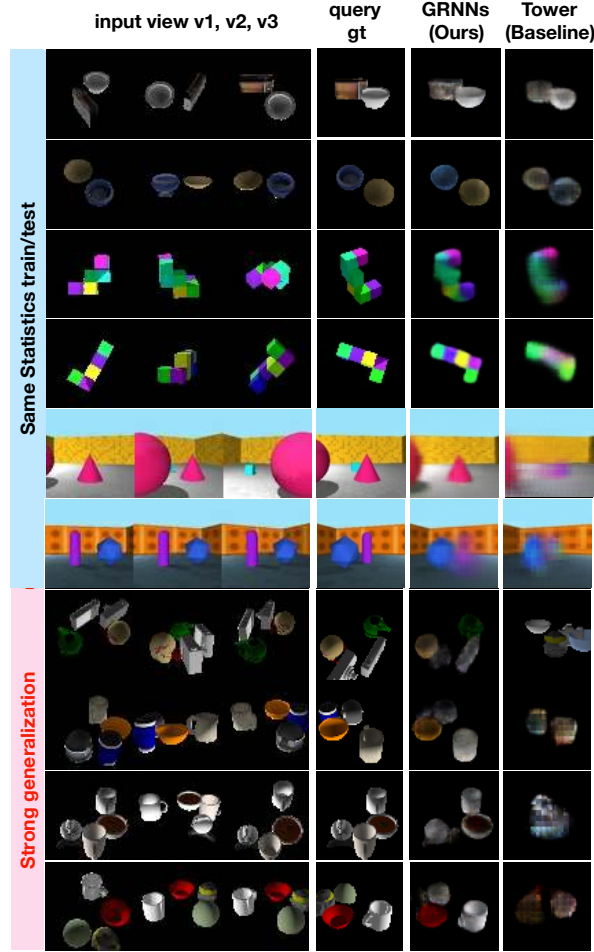


Figure 2.4: **View prediction results** for the proposed GRNNs and the tower model of Eslami et al. [23]. Columns from left to right show the three input views, the groundtruth image from the query viewpoint, the view predictions for GRNNs and for the tower baseline. The first two rows are from the ShapeNet arrangement test set of [11], the next two rows are from the Shepard-Metzler test set of [23], and the following two rows are from the *Rooms-ring-camera* dataset also from [23]. The last four rows show generalization to scenes with four objects from the ShapeNet arrangement dataset, while both models were trained only on scenes with two objects. GRNNs outperform the baseline by a large margin and *strongly* generalize under a varying number of objects.

### 2.2.2 3D object detection and segmentation

We train GRNNs in a supervised manner to predict 3D object bounding boxes and 3D object segmentation masks, using groundtruth 3D object boxes and 3D voxel segmentations from a simulator. We adapt MaskRCNN [38], a state-of-the-art object detector/segmentor, to have 3D input and output, instead of 2D. Specifically, we consider every grid location  $(X, Y, Z)$  in our 3D memory to be a candidate 3D box centroid. At each time step, the 3D feature memory  $\mathbf{M}_t$  is fed to a 3D region proposal network to predict positive anchor centroids, as well as the

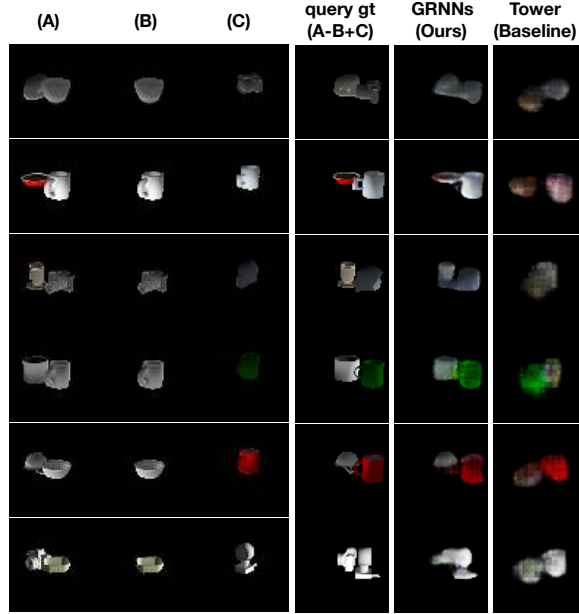


Figure 2.5: **Scene arithmetic** with GRNNs and the model of Eslami et al. [23] (tower). Each row is a separate "equation". We start with the representation of the scene in the leftmost column, then subtract (the representation of) the scene in the second column, and add the (representation of the) scene in the third column. We decode the resulting representation into an image. The groundtruth image is shown in the forth column. It is much more visually similar to the prediction of GRNNs than to the tower baseline.

corresponding adjustment for the box center location and the box dimensions, width, height and depth. Our 3D bounding box encoding is similar to the one proposed in VoxelNet [148]. We filter the proposed boxes using non-max suppression to reject highly overlapping ones. We train with a combination of classification and regression loss, following well established detector training schemes [38, 101]. The proposed 3D bounding boxes that have Intersection of Union (IoU) above a specific threshold with a corresponding groundtruth object box are denoted as Regions of Interest (ROIs) and are used to pool features from their interior to predict 3D object voxel occupancy, as well as a second refinement of the predicted 3D box location and dimensions.

**Object permanence** Even when an object is not visible in the current camera viewpoint, its features are present in the 3D feature memory, and our detector detects and segments it, as we show in the second column of Figure 2.6. In other words, object detections persist through occlusions and changes of the field of view caused by camera motion. Applying the detector on the latent 3D model of the scene as opposed to the 2D visual frame is beneficial. The latent 3D model follows the physical laws of 3D non-intersection and object permanence, while 2D visual observations do not.

We use the ShapeNet arrangement dataset, and the train/test scene split of [11]. We use mean Average Precision (mAP) to score the performance of our model and baselines for 3D object detection and 3D segmentation. Mean average precision measures the area under the

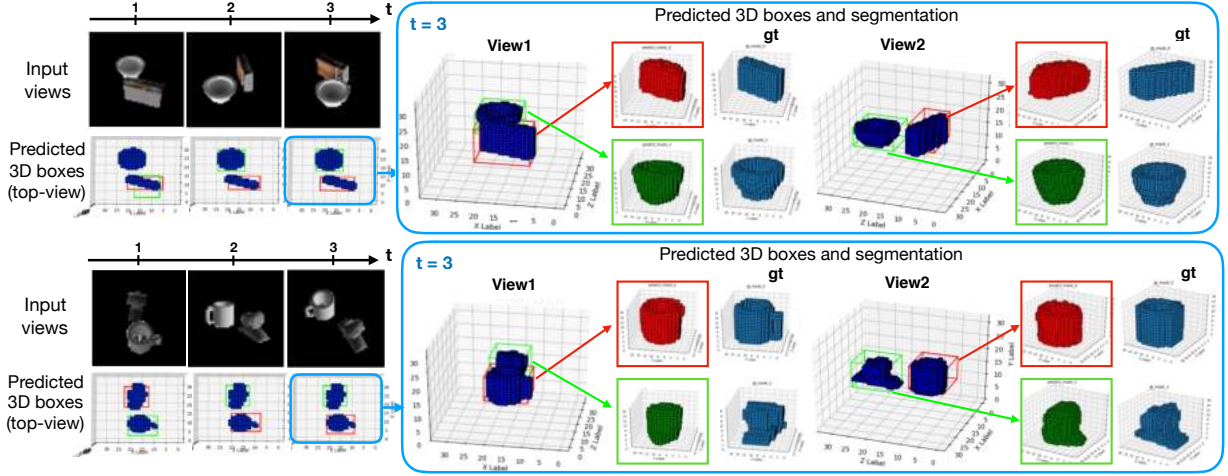


Figure 2.6: **3D object detection and segmentation** with GRNNs. In the first and second row on the left we show the input images over time, and their corresponding object detection results for a top view, respectively. Blue voxels denote groundtruth objects and the predicted bounding boxes are shown in **red** and **green**. On the right, we show segmentation results for the third time step, visualizing the results from two views. Predicted 3D boxes and their corresponding predicted masks are shown in red and green, and we show in blue the corresponding groundtruth. Best seen in color.

precision-recall curve. We vary the cutoff threshold of Intersection over Union (IoU) to be 0.33, 0.5 and 0.75 between our predictions and the groundtruth 3D boxes and masks. We consider four ablations for our model: predicted egomotion (*pego*) versus groundtruth egomotion (*gtego*) used, and predicted depth (*pd*) versus groundtruth depth (*gtd*) used as input. We use suffixes to indicate the model we use.

detection	2DRNN-gtego-gtd	GRNN-gtego-pd	GRNN-gtego-gtd	GRNN-pego-gtd
$mAP_{0.75}^d$	0.364	0.471	<b>0.816</b>	0.549
$mAP_{0.50}^d$	0.964	0.964	<b>0.998</b>	0.983
$mAP_{0.33}^d$	0.998	0.994	<b>0.999</b>	0.999
segmentation	2DRNN-gtego-gtd	GRNN-gtego-pd	GRNN-gtego-gtd	GRNN-pego-gtd
$mAP_{0.75}^m$	0.003	0.024	<b>0.058</b>	0.023
$mAP_{0.50}^m$	0.104	0.246	<b>0.338</b>	0.249
$mAP_{0.33}^m$	0.244	0.429	<b>0.485</b>	0.384

Table 2.2: Mean Average Precision (mAP) for 3D object detection and 3D segmentation for three different thresholds of Intersection over Union (IoU) (0.75,0.5,0.33) on ShapeNet arrangement test set of [11].

We compare against the following 2D baseline model, which we call *2D-RNN*: we remove the unprojection, egomotion estimation and stabilization and projection operations from our model. The baseline takes as input an image and the corresponding depth map, feeds it to a 2D encoder-decoder network with skip connections to obtain a 2D feature tensor. The camera parameters

for this view are concatenated as additional channels to the 2D feature tensor and altogether they are fed to another 2D encoder-decoder network to obtain the 2D feature tensor for a 2D GRU memory update. We then feed the 2D memory feature tensor to an additional 2D encoder-decoder network and reshape the channel dimension of its output into  $d$  feature vector of length 7 (one value for the anchor box prediction, six values for the 3D bounding boxes adjustments) to form a 4D tensor of size  $w \times h \times d \times 7$  as prediction.

We show mean average precision for 3D object detection and 3D segmentation for our model and the baseline in Table 2.2, and visualize predicted 3D bounding boxes and segmentations from GRNNs (GRNN-gtego-gtd) in Figure 2.6. GRNNs significantly outperform the 2D-RNN. Groundtruth depth input significantly helps 3D segmentation. This suggests that inferring depth using a cost volume as in [56] would potentially help depth inference as opposed to relying on a per frame depthnet [21] that does not have access to multiple views to improve its predictions.



# Chapter 3

## Learning to See Moving Objects without 3D Labels

### 3.1 Introduction

In the previous chapter, we have introduced Geometry-Aware Recurrent Networks (GRNNs), neural architectures that construct **end-to-end trainable view-invariant 3D feature representations** for the scenes captured by the embodied agents. We have shown that GRNNs outperform and generalize better than geometry-unaware baselines in 3D object detection and view prediction. Here in this chapter, we want to move a step forward to answer another interesting question: since our machines are embodied agents that can act in the world to collect more data, can they learn or improve through their own collected data? To answer this question, we need to think about what kind of data can mobile agents collect. Assuming these agents are in their initial stage where they have not yet developed their visual perception and cannot even localize objects from the scene, all they can do is move in random directions. Hopeless as it sounds, when the agents move, they can observe a sequence of images capturing from different views. Besides, they will have access to the pose and the displacement of their torques. Using the data, we propose to use *prediction* as a self-supervisory signal in allowing agents to learn better visual representations that will aid their generalization ability.

The first task we propose, which has been introduced in the previous chapter, is view prediction: the agents learn to predict how a scene looks from various camera viewpoints given an input view. Besides RGB images, learning the tasks require the camera positions where the images are taken from, because the models need to know which view to generate. Such information is accessible to the agents since they can calculate the camera poses through forward kinematics using the pose and the displacement of their torques. To be able to predict images from multiple views, the models must learn to complete the features invisible from the input view so they can generate the pixels correctly beyond what can be observed. In this sense, the features can become consistent across views, making them truly view-invariant.

View prediction has been the center of much recent research effort. Most methods test their models in single object scenes, and aim to generate beautiful images for graphics applications [56, 107, 115, 128], as opposed to learning general-purpose visual representations. The work

<sup>0</sup>This chapter is based on the paper published previously at ICLR 2020 [35].

of [23] attempted view prediction in full scenes, yet only experimented with toy data containing a few colored 3D shapes. Their model cannot effectively generalize beyond the training distribution, e.g., cannot generalize across scenes of a variable number of objects.

In the previous chapter, we have shown our models outperform the work of [23] and generalize well in both 3D object detection and view prediction tasks. In this chapter, we show that, with this self-supervised objective, we can further improve our 3D object detection performance. However, we found out training view prediction by regressing the outputs to the target image in the pixel space does not scale to complex scenes like real-world street views. To address the problem, we introduce a novel view-contrastive loss that learns more semantically meaningful features. With the learned features, we can discover objects in 3D from a single camera viewpoint, without any human annotations of object boxes or masks.

## 3.2 Semi-supervised learning of 3D object detection

Can pre-training the models with the view prediction objective help to improve the object detection performance? To answer the question, we pre-train the GRNNs weights with view prediction, and then train a 3D object detector module supervised to map a 3D feature volume  $\mathbf{M}$  to 3D object boxes. We compare the model with a model trained from random weight initialization, i.e., without pre-training. After pre-training, we *freeze* the feature layers after view predictive learning, and only supervise the detector module; for the fully supervised baseline (from random initialization), we train end-to-end.

We train our models in CARLA [18], an open-source photorealistic simulator of urban driving scenes, which permits moving the camera to any desired viewpoint in the scene. We obtain data from the simulator as follows. We generate 1170 autopilot episodes of 50 frames each (at 30 FPS), spanning all weather conditions and all locations in both “towns” in the simulator. We define 36 viewpoints placed regularly along a 20m-radius hemisphere in front of the ego-car. This hemisphere is anchored to the ego-car (i.e., it moves with the car). In each episode, we sample 6 random viewpoints from the 36 and randomly perturb their pose, and then capture each timestep of the episode from these 6 viewpoints. We generate train/test examples from this, by assembling all combinations of viewpoints (e.g.,  $N \leq 5$  viewpoints as input, and 1 unseen viewpoint as the target). We filter out frames that have zero objects within the metric “in bounds” region of the GRNN ( $32m \times 32m \times 4m$ ). This yields 172524 frames (each with multiple views): 124256 in Town1, and 48268 in Town2. We treat the Town1 data as the “training” set, and the Town2 data as the “test” set, so there is no overlap between the train and test images.

We are interested in seeing the benefit of this pre-training across different amounts of label supervision, so we first use the full CARLA train set for view prediction training (without using box labels), and then use a randomly-sampled subset of the CARLA train set for box supervision; we evaluate on the CARLA validation set. We varied the size of the box supervision subset across the following range: 100, 200, 500, 1000, 10000, 80000. We show mean average precision (at an IoU of 0.75) for car detection as a function of the number of annotated 3D bounding box examples in Figure 3.1. As expected, the supervised model performs better with more labelled data. In the low-data regime, pre-training greatly improves results.

However, we observe that the predicted images from this dataset are blurry and miss many

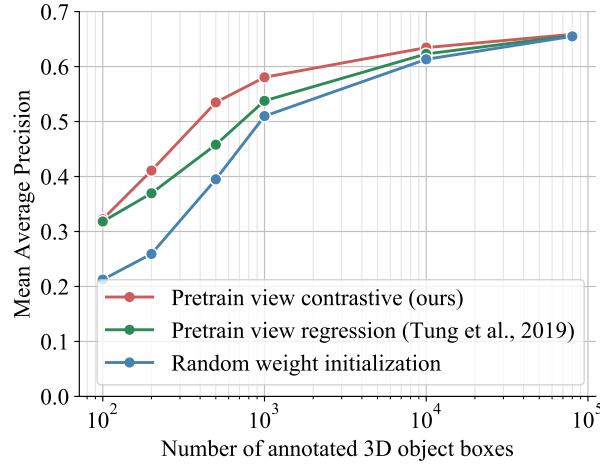


Figure 3.1: **Semi-supervised 3D object detection.** Pre-training with view-contrastive prediction improves results, especially when there are few object 3D bounding box annotations.

vital details, which raises the concern that learning view prediction through RGB regression might be insufficient to obtain semantically meaningful features for complex scenes. To address the problem, we sort to another option which makes predictions in a latent feature space. Recently, [88] used a probabilistic contrastive objective that preserves mutual information between the future bottom-up extracted features and the predicted contextual latent features. By training with the contrastive objective, [88] shows the model can achieve strong performance in diverse domains, including speech, text and image, text, and reinforcement learning in 3D environments. The view-contrastive loss proposed in this work is a non-probabilistic version of their contrastive objective. However, our work focuses on the video domain as opposed to image patches, and uses drastically different architectures for both the contextual and bottom-up representations, using a 3D representation bottleneck. In Figure 3.1, we show by pretraining the model with view-contrastive losses, we can further improve the 3D object detection results when the number of training data is limited. Next, we introduce in detail the view-contrastive loss.

### 3.2.1 View-contrastive rendering

Given a set of input RGBs, pointclouds, and camera poses  $(I^{(1)}, D^{(1)}, V^{(1)}), \dots, (I^{(n)}, D^{(n)}, V^{(n)})$ , we train our model to predict feature abstractions of an unseen input  $(I^{(n+1)}, D^{(n+1)}, V^{(n+1)})$ . We consider two types of representations for the target view: a top-down one,  $\mathcal{T} = f[(I^{(1)}, D^{(1)}, V^{(1)}), \dots, (I^{(n)}, D^{(n)}, V^{(n)}), V^{(n+1)}]$ , and a bottom-up one,  $\mathcal{B} = g[I^{(n+1)}, D^{(n+1)}]$ . Note that the top-down representation has access to the viewpoint  $V^{(n+1)}$  but not to observations from that viewpoint  $(I^{(n+1)}, D^{(n+1)})$ , while the bottom-up representation is only a function of those observations.

We construct 2D and 3D versions of these representation types, using our architecture modules:

- We obtain  $\mathcal{T}^{3D} = \otimes M^{(n)}$  by encoding the set of inputs  $1, \dots, n$ .
  - We obtain  $\mathcal{B}^{3D} = \otimes F^{(n+1)}$  by encoding the single input  $n+1$ .
  - We obtain  $\mathcal{T}^{2D} = M_{\text{view}_{n+1}}^{(n)}$  by rendering  $\otimes M^{(n)}$  from viewpoint  $V^{(n+1)}$ .
  - We obtain  $\mathcal{B}^{2D} = F^{(n+1)}$  by convolving  $I^{(n+1)}$  with a 3-block 2D ResNet [37].
- Finally, the contrastive losses pull corresponding (top-down and bottom-up) features close to-

gether in embedding space, and push non-corresponding ones beyond a margin of distance:

$$\mathcal{L}_{\text{contrast}}^{2\text{D}} = \sum_{i,j,m,n} \max(\mathcal{Y}_{ij,mn}^{2\text{D}}(\|\mathcal{T}_{ij}^{2\text{D}} - \mathcal{B}_{mn}^{2\text{D}}\|_2 - \alpha), 0), \quad (3.1)$$

$$\mathcal{L}_{\text{contrast}}^{3\text{D}} = \sum_{i,j,k,m,n,o} \max(\mathcal{Y}_{ijk,mno}^{3\text{D}}(\|\mathcal{T}_{ijk}^{3\text{D}} - \mathcal{B}_{mno}^{3\text{D}}\|_2 - \alpha), 0), \quad (3.2)$$

where  $\alpha$  is the margin size, and  $\mathcal{Y}$  is 1 at indices where  $\mathcal{T}$  corresponds to  $\mathcal{B}$ , and  $-1$  everywhere else. The losses ask tensors depicting the same scene, but acquired from different viewpoints, to contain the same features. The performance of a metric learning loss depends heavily on the sampling strategy used [111, 119, 120]. We use the distance-weighted sampling strategy proposed by [140] which uniformly samples “easy” and “hard” negatives; we find this outperforms both random sampling and semi-hard [111] sampling.

### 3.2.2 Experiments

#### Sim-to-Real (CARLA-to-KITTI) transfer

We evaluate whether the 3D predictive feature representations learned in the CARLA simulator are useful for learning 3D object detectors in the real world by testing on the real KITTI dataset [29]. Specifically, we use view prediction pre-training in the CARLA train set, and box supervision from the KITTI train set, and evaluate 3D object detection in the KITTI validation set. We use the (single-view) object detection benchmark from the KITTI dataset [29], with the official train/val split: 3712 training frames, and 3769 validation frames. Existing real-world datasets do not provide enough camera viewpoints to support view-predictive learning. Specifically, in KITTI, all the image sequences come from a moving car and thus all viewpoints lie on a near-straight trajectory. Thus, simulation-to-real transferability of features is especially important for view predictive learning.

We show simulation-to-real transfer results in Table 3.1. We compare the proposed view contrastive prediction pre-training, with view regression pre-training, and random weight initialization (no pretraining). In all cases, we train a 3D object detection module supervised using KITTI 3D box annotations. We also compare *freezing* versus

Method	mAP@IOU		
	0.33	0.50	0.75
Random weight initialization	.59	.52	.17
Pretrain view regress., frozen	.64	.54	.15
Pretrain view regress., finetuned	.65	.55	.18
Pretrain view contrast, frozen	.67	.58	.15
Pretrain view contrast, finetuned	<b>.70</b>	<b>.60</b>	<b>.19</b>

Table 3.1: **CARLA-to-KITTI transferability of view-predictive 3D feature representations.** We train a 3D detector module on top of the inferred 3D feature maps  $\mathbf{M}$  using KITTI 3D object box annotations

*finetuning* the weights of the pretrained inverse graphics network. The results are consistent with the CARLA tests: view-contrastive pretraining is best, view regression pretraining is second, and learning from human annotations alone is worst. Note that depth in KITTI is acquired by a real

velodyne LiDAR sensor, and therefore has lower density and more artifacts than CARLA, yet our model generalizes across this distribution shift.

### 3.3 Unsupervised 3D moving object detection

Aside from improving the object detection results, here we propose an algorithm that can discover moving objects in an unsupervised manner using the learned features through view prediction. Upon training, our model learns to map even a *single* RGB-D input to a complete 3D imagination. Given two temporally consecutive **and registered** 3D maps  $\otimes F^{(t)}$ ,  $\otimes F_{\text{reg}}^{(t+1)}$ , predicted independently using inputs  $(I^{(t)}, D^{(t)})$  and  $(I^{(t+1)}, D^{(t+1)})$ , we train a motion estimation module to predict the 3D motion field  $\otimes W^{(t)}$  between them, which we call 3D imagination flow. Since we have accounted for camera motion, **this 3D motion field should only be non-zero for independently moving objects**. We obtain 3D object proposals by clustering the 3D flow vectors, extending classic motion clustering methods [8, 86] to an egomotion-stabilized 3D feature space, as opposed to 2D pixel space.

#### Estimating 3D imagination flow

Our 3D FlowNet is a 3D adaptation of the PWC-Net (2D) optical flow model [124]. Note that our model only needs to estimate motion of the independently-moving part of the scene, since egomotion has been accounted for. It works by iterating across scales in a coarse-to-fine manner. At each scale, we compute a 3D cost volume, convert these costs to 3D displacement vectors, and incrementally warp the two tensors to align them. We train our 3D FlowNet using two tasks: (1) Synthetic transformation of feature maps: We apply random rotations and translations to  $\otimes F^{(t)}$  and ask the model to recover the dense 3D flow field that corresponds to the transformation; (2) Unsupervised 3D temporal feature matching:  $\mathcal{L}_{\text{warp}} = \sum_{i,j,k} \|\otimes F_{i,j,k}^{(t)} - \mathcal{W}(\otimes F_{\text{reg}}^{(t+1)}, \otimes W^{(t)})_{i,j,k}\|$ , where  $\mathcal{W}(\otimes F_{\text{reg}}^{(t+1)}, \otimes W^{(t)})$  back-warps  $\otimes F_{\text{reg}}^{(t+1)}$  to align it with  $\otimes F^{(t)}$ , using the estimated flow  $\otimes W^{(t)}$ . We apply the warp with a differentiable 3D spatial transformer layer, which does trilinear interpolation to resample each voxel. This extends self-supervised 2D optical flow [145] to 3D feature constancy (instead of 2D brightness constancy). We found that both types of supervision are essential for obtaining accurate 3D flow field estimates. Since we are not interested in the 3D motion of empty air voxels, we additionally estimate 3D voxel occupancy, and supervise this using the input pointclouds; we set the 3D motion of all unoccupied voxels to zero. We describe our 3D occupancy estimation in more detail in the appendix.

The proposed 3D imagination flow enjoys significant benefits over 2D optical flow or 3D scene flow. It does not suffer from occlusions and dis-occlusions of image content or projection artifacts [123], which typically transform rigid 3D transformations into non-rigid 2D flow fields. In comparison to 3D scene flow [43], which concerns visible 3D points, **3D imagination flow is computed between visual features that may never have appeared in the field of view, but are rather inpainted by imagination**.

#### 3D moving object segmentation

We obtain 3D object segmentation proposals by thresholding the 3D imagination flow magnitude, and clustering voxels using connected components. We score each component using a 3D

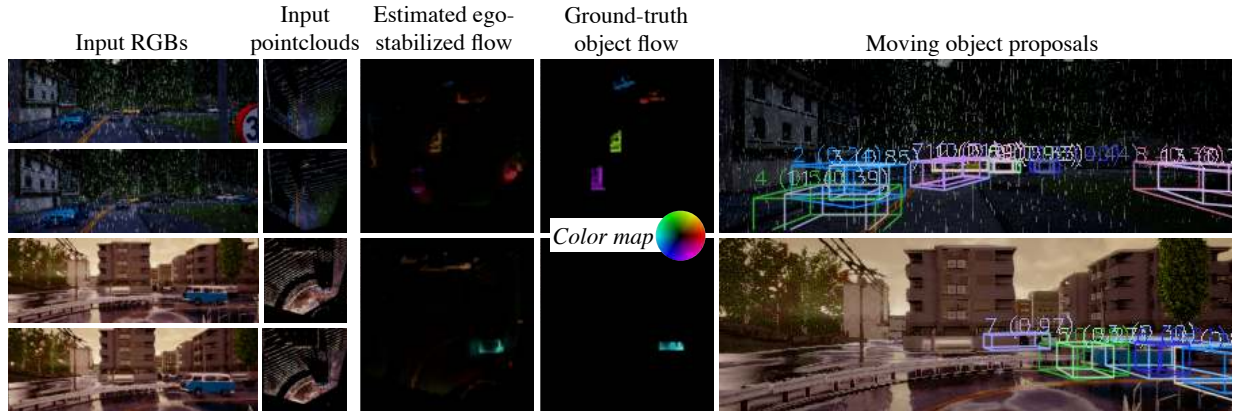


Figure 3.2: **3D feature flow and object proposals, in dynamic scenes.** Given the input frames on the left, our model estimates dense egomotion-stabilized 3D flow fields, and converts these into object proposals. We visualize colorized pointclouds and flow fields in a top-down (bird’s eye) view.

version of a center-surround motion saliency score employed by numerous works for 2D motion saliency detection [28, 73]. This score is high when the 3D box interior has lots of motion but the surrounding shell does not. This results in a set of scored 3D segmentation proposals for each video scene.

### 3.3.1 Experiments

In this section, we test our model’s ability to **detect moving objects in 3D without any 3D object annotations**, simply by clustering 3D motion vectors. We use two-frame video sequences of dynamic scenes from the CARLA data, and we split the validation set into two parts for evaluation: scenes where the camera is stationary, and scenes where the camera is moving. This splitting is based on the observation that moving object detection is made substantially more challenging under a moving camera.

We show precision-recall curves for 3D moving object detection under a *stationary camera* in Figure 3.3. We compare our model against a model trained with RGB view regression and a 2.5D baseline. The 2.5D baseline computes 2D optical flow using PWC-Net [124], then proposes object masks by thresholding and clustering 2D flow magnitudes; these 2D proposals are mapped to 3D boxes by segmenting the input pointcloud according to the proposed masks. Our model outperforms the baselines. Note that even with ground-truth 2D flow, ground-truth depth, and an oracle threshold, a 2.5D baseline can at best only capture the portions of the objects that are in the pointcloud. As a result, 3D proposals from PWC-Net often underestimate the extent of the objects by half or more. Our model imagines the full 3D scene in each frame, so it does not have this issue.

We show precision-recall curves for 3D moving object detection under a *moving camera* in Figure 3.4. We compare our model where egomotion is predicted by our neural egomotion module, against our model with ground-truth egomotion, as well as a 2.5D baseline, and a stabilized 2.5D baseline. The 2.5D baseline uses optical flow estimated from PWC-Net as before. To

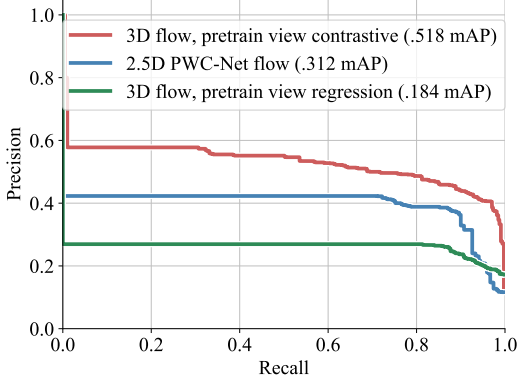


Figure 3.3: **Unsupervised 3D moving object detection with a *stationary* camera.**

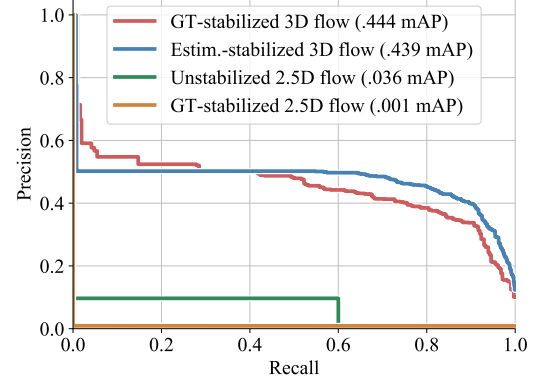


Figure 3.4: **Unsupervised 3D moving object detection with a *moving* camera**

stabilize the 2.5D flow, we subtract the ground-truth scene flow from the optical flow estimate before generating proposals. Our model’s performance is similar to its level in static scenes, suggesting that the egomotion module and stabilization mechanism effectively disentangles camera motion from the 3D feature maps. The 2.5D baseline performs poorly in this setting, as expected. Surprisingly, performance drops further after stabilizing the 2D flows for egomotion. We confirmed this is due to the estimated scene flow being imperfect: subtracting ground-truth scene flow leaves many motion fragments in the background. With ground-truth 2D flow, the baseline performs similar to its static-scene level.

We have attempted to compare against the unsupervised object segmentation methods proposed in [44, 61] by adapting the publicly available code accordingly. These models use an inference network that takes as input the full video frame sequences to predict the locations of 2D object bounding boxes, as well as frame-to-frame displacements, in order to minimize view prediction error in 2D. We were not able to produce meaningful results from their inference networks. The success of [44] may partially depend on carefully selected priors for 2D object bounding box location and object size parameters that match the moving MNIST dataset statistics used in the paper, as suggested by the publicly available code. We do not assume knowledge or existence of such object location or size priors for our CARLA data.





## **Part II**

**Action + Physics: Learning to imagine how objects can move and how to interact with them**



# Chapter 4

## Learning View-invariant Intuitive Physics Models for Manipulation

### 4.1 Introduction

In Part I, we have introduced neural architectures that would allow embodied agents to perceive a stabilized scene while moving. We also introduce how the agents can learn to detect objects in the scene representations. In this chapter, we want to discuss how the agents can develop a deeper understanding of the scene beyond simple object detection. For the agents to intelligently interact with scene or reason about what can happen in the scene, it is critical that they learn how the environment reacts to their actions and interactions [36, 80].

Humans can effortlessly imagine how a scene will change as a result of their interactions with the objects in the scene [15, 27]. What is the representation space of these imaginations? They are not pixel accurate and, interestingly, they are not affected by occlusions. Consider a teaspoon dipping inside a coffee mug. Though it will be occluded from nearly all viewpoints but the bird’s eye view, we have no difficulty keeping it in our mind as present and complete. We can imagine watching it from different viewpoints, increase or decrease its size, predict whether it will fit inside the mug, or even imagine filling the mug with more spoons.

Inspired by humans capability to simulate scene changes in a viewpoint-invariant and occlusion-resistant manner, we present 3D object-factorized environment simulators (3D-OES), an action-conditioned dynamics model that predicts scene changes caused by object and agent interactions in a viewpoint-invariant 3D neural scene representation space, inferred from RGB-D videos. Using Geometry-Aware Recurrent Networks (GRNNs) introduced in Chapter 2, 3D-OES differentially maps an RGB-D image to a 3D neural scene representation, detects objects in it, and forecasts their future 3D motions, conditioned on actions of the agent. A graph neural network operates on the extracted 3D object feature maps and the action input and predicts object 3D translations and rotations. Our model then generates future 3D scenes by simply translating and rotating object 3D feature maps, inferred from the *first time step*, according to cumulative 3D motion predictions. In this way, we avoid distribution shift in object features caused by forward model unrolling, hence minimizing error accumulation.

<sup>0</sup>This chapter is based on the paper published previously at CoRL 2021 [137].

Our main insight is that scene dynamics are simpler to learn and represent in 3D than in 2D, for the following reasons: i) **In 3D, object appearance and object location are disentangled.** This means object appearance (what) does not vary with object locations (where). This what-where disentanglement permits generating scene variations by simply translating and rotating 3D object appearance representations. Scene generation by moving around objects is not possible in a projective 2D image space, since objects change appearance due to camera viewpoint variation, occlusions or out-of-plane object rotations [19]. It is precisely the permanence of object appearance in 3D that permits easy simulation. ii) **In 3D, inferring free space and object collisions is easy.** Given a 3D scene description in terms of object locations and 3D shapes, we can easily predict whether an object will collide with another or will be contained in another. Similar inferences would require many examples to learn directly from 2D images, and would likely have poor generalization. Yet, extracting 3D scene representations from RGB or RGB-D video streams is a challenging open problem in computer vision research [49, 65, 103, 104, 135]. We build upon the recently proposed geometry-aware recurrent neural networks (GRNNs) [33, 135] to infer 3D scene feature maps from RGB-D images in a differentiable manner, optimized end-to-end for our object dynamics prediction task.

We evaluate 3D-OES in single-step and multi-step object motion prediction for object pushing and falling, and apply it for planning to push objects to desired locations. We test its generalization while varying the number and appearance of objects in the scene, and the camera viewpoint. We compare against existing learning-based 2D image-centric or object-centric models of dynamics [32, 144] as well as graph-based dynamics learned over engineered 3D representations of object locations [108]. Our model outperforms them by a large margin. In addition, we empirically show that training the 2D baselines under varying viewpoints causes them to dramatically underfit on the training data, and be highly inaccurate in the validation set. This suggests that different architectures are necessary to handle viewpoint variations in dynamics learning and 3D-OES is one step in that direction.

In summary, the main contribution of this work is a graph neural network over 3D object feature maps extracted from convolutional end-to-end differentiable 3D neural scene representations for forecasting 3D object motion. Graph networks are widely used in 2D object motion interaction predictions [5, 51, 58, 144]. We show that by porting such relational reasoning in an 3D object-factorized space, object motion prediction can generalize across camera viewpoints, lifting a major limitation of previous works on 2D object dynamics. Moreover, future and counterfactual scenes can be easily generated by translating and rotating 3D object feature representations. In comparison to recent 3D particle graph networks [69, 83, 109], our work can operate over input RGB-D images and not ground-truth particle graphs. In comparison to recent scene-specific image-to-3D particle graph encoders [70], our image to 3D scene encoder can generalize across environments with novel objects, novel number of objects, and novel camera viewpoints. Moreover, our model presents effective sim-to-real transfer to a real-world robotic setup.

## 4.2 Object-Factorized Environment Simulators (3D-OES)

The architecture of 3D-OES is depicted in Figure 6.2. At each time step, our model takes as input a single or a set of RGB-D images of the scene along with the corresponding camera views

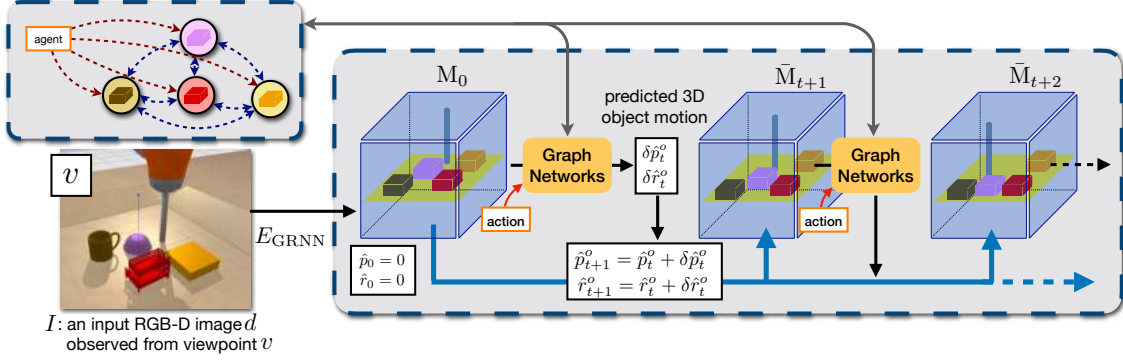


Figure 4.1: **3D-OES** predict 3D object motion under agent-object and object-object interactions, using a graph neural network over 3D feature maps of detected objects. Node features capture the appearance of an object node and its immediate context, and edge features capture relative 3D locations between two nodes, so the model is translational invariant. After message passing between nodes, the node and edge features are decoded to future 3D rotations and translations for each object.

to capture them, and encodes these inputs into a 3D scene feature representation using Geometry-Aware Recurrent Networks (GRNNs) introduced in Chapter 2. Then, it detects 3D object boxes in the inferred 3D scene representation, and crops the scene representation to obtain a set of object-centered 3D feature maps (see Section 2.2.2 for details of the object detector). A graph neural network over the object nodes will take as inputs object appearances and the agent actions and predict the future 3D rotation and translation for each object (Section 4.2.1). We will assume for now rigid objects, and we discuss in Chapter 9 how to extend our framework to deformable and articulated objects. Our model generates future scenes by warping object-centric 3D feature maps with the predicted cumulative 3D object motion. These synthesized future 3D scene feature maps, though not directly interpretable, can be decoded to RGB images from any desired camera viewpoints via a neural renderer to aid interpretability. We use long-term simulations of 3D-OES to generate action plans for pushing objects to desired locations in cluttered environments using model predictive control (Section 4.2.2). We apply our model to learn dynamics of objects pushed around on a table surface and objects falling on top of others. At training time, we assume access to 3D object bounding boxes to train our 3D object detector.

**Differentiable 2D-to-3D lifting with Geometry-Aware Recurrent Networks (GRNNs)** Our model use Geometry-Aware Recurrent Networks (GRNNs) introduced in Chapter 2 to infer 3D scene feature representations from RGB-D images. We will denote the 3D scene feature as  $\mathbf{M} \in \mathbb{R}^{w \times h \times d \times c}$  where  $w, h, d, c$  denote width, height, depth and number of channels, respectively. Given an input video, GRNNs estimate the relative camera poses between frames, and transform the inferred 3D features map  $\mathbf{M}_t$  to a world coordinate frame to cancel the camera egomotion, before accumulating it with 3D feature maps across time steps. In this way, information from 2D pixels that correspond to the same 3D physical point end up nearby in the 3D neural map. We use such cross-view registration in case we have access to concurrent multiple camera views for the first timestep of our simulations. Upon training, GRNNs map RGB-D images or a single RGB-D image to a *complete* 3D feature map of the scene they depict, i.e., the model learns to *imagine*

the missing or occluded information from the input view. We denote this 2D-to-3D mapping as  $\mathbf{M} = \text{GRNN}(I_1, \dots, I_t)$ , where  $\mathbf{M} \in \mathbb{R}^{w \times h \times d \times c}$  and  $I_t = \{d_t, v_t\}$  denotes the RGB-D image  $d_t$  and the corresponding camera pose  $v_t$  at time step  $t$ . Note that the input can be a single RGB-D view, in which case  $\mathbf{M} = \text{GRNN}(I)$ . For further details on GRNNs, please refer to Chapter 2.

**View prediction for visualizing latent 3D neural simulations** We train GRNNs end-to-end for RGB view regression in videos of static scenes and moving cameras as proposed in Chapter 2 Section 2.2.1, by neurally projecting the 3D scene feature maps and mapping them to 2D images. Our decoder involves a differentiable 3D-to-2D projection module that projects the 3D scene feature representation after orienting it to the query camera viewpoint. The projected features are then decoded into images through a learned decoder. In this way, the trained projection and decoding module can be used to interpret and visualize the 3D latent feature space with view-specific 2D images, given any desired camera viewpoint.

**3D object detection** Our model uses a 3D object detector to map the 3D scene neural map  $\mathbf{M}$  to a variable number of object axis-aligned 3D boxes and corresponding 3D segmentation masks, i.e., binary 3D voxel occupancies:  $\mathcal{O} = \text{Det}(\mathbf{M})$ ,  $\mathcal{O} = \{\hat{b}^o = (p_x^o, p_y^o, p_z^o, w^o, h^o, d^o) \in \mathbb{R}^6, m^o \in \{0, 1\}^{w^o \times h^o \times d^o}, o = 1 \dots |\mathcal{O}|\}$ , where  $p_x^o, p_y^o, p_z^o$  stands for the 3D box centroid and  $w^o, h^o, d^o$  stands for 3d box size. Its architecture is similar to Mask R-CNN [38] but uses 3D input and output instead of 2D. Given an object 3D centroid  $p_x^o, p_y^o, p_z^o$ , we crop the 3D scene feature map  $\mathbf{M}$  using a corresponding fixed-size axis-aligned 3D bounding box to obtain corresponding object-centric feature maps  $\mathbf{M}^o, o = 1 \dots |\mathcal{O}|$  for all objects in the scene. Please refer to Chapter 2 Section 2.2.2 for more details about the object detector.

### 4.2.1 3D Object Graph Neural Networks for Motion Forecasting

Objects are the recipient of forces exercised by active agents; meanwhile, objects themselves carry momentum and cause other objects to move. How can we model cross-object dynamic relationships in a way that generalizes with varying number of objects and arbitrary chains of interactions?

We consider a graph interaction network [5] over the graph comprised of the detected objects and the agent’s end-effector. Inputs to the network are the object-centric feature maps, one per object node, the objects’ velocities, the agent’s action represented as a 3D translation, as well as edge features, which incorporate the relative 3D displacements between the nodes. The outputs of the network are the 3D translations  $\delta \hat{p}$  and 3D relative rotations  $\delta \hat{r}$  of the object nodes at the next time step. During message passing in the constructed graph, edge and node features are encoded and concatenated, and messages from neighboring nodes are aggregated via summation. Our graph network is trained supervised to minimize a standard regression loss for the next time step.

**Forward unrolling with object appearance permanence** To predict long term results of actions, as well as results of action sequences, the model needs to be unrolled forward in time as commonly done in related works [5, 6, 51, 58, 144]. Different from previous works though, 3D-OES can synthesize 3D neural scenes of future timesteps by warping (translating and rotating) object feature maps obtained from the first timestep—as opposed to the ones obtained from the predicted scene of the previous timestep—according to cumulative 3D motion predictions.

Specifically, given predicted 3D object motions  $(\delta\hat{p}_t, \delta\hat{r}_t)$  at an unrolling step  $t$ , we estimate the **cumulative** 3D rotation and translation of the object with respect to the first timestep:

$$\hat{p}_t = \hat{p}_{t-1} + \delta\hat{p}_t, \quad \hat{r}_t = \hat{r}_{t-1} + \delta\hat{r}_t, \quad t = 1 \cdots T, \quad \hat{p}_0 = 0 \quad \hat{r}_0 = 0. \quad (4.1)$$

where  $T$  denotes the number of unrolling steps thus far. Then, given 3D object segmentation masks  $m^o$  and object-centric 3D feature maps  $\mathbf{M}^o$  obtained by the 3D object detector from the input RGB-D image, we rotate and translate the object masks and 3D feature maps using the cumulative 3D rotation  $\hat{r}_t$  and 3D translation  $\hat{p}_t$  using 3D spatial transformers. We synthesize a new 3D scene feature map  $\bar{\mathbf{M}}_t$  by placing each transformed object-centric 3D feature map at its predicted 3D location:  $\bar{\mathbf{M}}_t = \sum_{o=1}^{|\mathcal{O}|} \text{Dec}(\text{Rot}(m^o, \hat{r}_t^o) \odot \text{Rot}(\mathbf{M}^o, \hat{r}_t^o), \hat{p}_t^o)$ , where superscript  $o$  denotes the object identity,  $\text{Rot}(\cdot, r)$  denotes 3D rotation by angle  $r$ ,  $\odot$  denotes voxel-wise multiplication, and  $\text{Dec}(\mathbf{M}, p)$  denotes adding a feature tensor  $\mathbf{M}$  at a 3D location  $p$ . This synthesized scene map is used for neural rendering to help interpret the predicted scene at  $t$ . To obtain the inputs for our graph neural network at the next time step, we can potentially crop the synthesized 3D scene map  $\bar{\mathbf{M}}_t$  at the predicted 3D location. However, we find that directly using object features obtained in the first time step and including accumulative relative object pose as part of the object state works better in practice.

Our graph neural motion forecaster is trained through forward unrolling. Error of each time step is back-propagated through time. More implementation details are included in Appendix Section C.1.

## 4.2.2 Model Predictive Control with 3D-OES

Action-conditioned dynamics models, such as 3D-OES, simulate the results of an agent’s actions and permit successful control in zero-shot setups: achieving a specific goal in a novel scene without previous practice. We apply our model for pushing objects to desired locations in cluttered environments with model predictive control. Given an input RGB-D image  $I$  that contains multiple objects, a goal configuration is given in terms of the desired 3D location of an object  $\mathbf{x}_{goal}^o$ . 3D-OES infer the scene 3D feature map  $\mathbf{M} = \text{GRNN}(I)$  and detects the objects present in the scene. We then unroll the model forward in time using randomly sampled action sequences, as described in Section 4.2.1. We evaluate each action sequence based on the Euclidean distance from the goal to the predicted location  $\hat{x}_T^o$  (after  $T$  time steps) for the designated object. We execute the first action of the best action sequence and repeat [126]. Our model combines 3D perception and planning using learned object dynamics in the inferred 3D scene feature map. While most previous works choose bird’s eye viewpoints to minimize cross-object or robot-object occlusions [24], our control framework **can use any camera viewpoint**, thanks to its ability to map input 2.5D images to complete, viewpoint-invariant 3D scene feature maps. We empirically validate this claim in our experimental section.

## 4.3 Experiments

We evaluate our model on its prediction accuracy for single- and multi-step object motion forecasting under multi-object interactions, as well as on its performance in model predictive control

for pushing objects to desired locations on a table surface in the presence of obstacles. We ablate generalization of our model under varying camera viewpoints and varying number of object and varying object appearance. Our model is trained to predict 3D object motion during robot **pushing** and **falling** in the Bullet Physics Simulator. For **pushing**, we have objects pushed by a Kuka robotic arm and record RGB-D video streams from multiple viewpoints. We create scenes using 31 different 3D object meshes, including 11 objects from the MIT Push dataset [146] and 20 objects randomly selected from *camera*, *mug*, *bowl*, and *bed* object categories of the ShapeNet dataset [9]. At training time, each scene contains at most *two* objects. We test with varying number of objects. For **falling**, we use 3D meshes of the objects introduced in Janner et al. [51], including a variety of shapes. We randomly select 1-3 objects and randomly place them on a table surface, and let one object fall from a height. We train our model with three camera views, and use either three or one randomly selected views as input during test time.

We compare 3D-OES against a set of baselines designed to cover representative models in the object dynamics literature: (1) *graph-XYZ*, a model that mimics Interaction Networks [4, 5, 141]. It is a graph neural network in which object features are the 3D object centroid locations and their velocities, and edge features are their relative 3D locations. (2) *graph-XYZ-image*, a model using graph neural network over 3D object centroid locations and object-centric 2D image CNN feature embeddings, similar to Ye et al. [144]. The model further combines camera pose information with the node features. (3) Visual Foresight (*VF*) [20], a model that uses the current frame and the action of the agent to predict future 2D frames by “moving pixels” based on predicted 2D pixel flow fields. (4) *PlaNet* [32], a model that learns a scene-level embedding by predicting future frames and the reward given the current frame.

We compare our model against baselines *graph-XYZ* and *graph-XYZ-image* on both motion forecasting and model predictive control. Since *VF* and *PlaNet* forecast 2D pixel motion and do not predict explicit 3D object motion, we compare against them on the pushing task with model predictive control.

**Implementation Details for Baselines** Here we describe the baselines discussed in Section 4 in detail.

1. *graph-XYZ*, a model that uses the 3D object centroid ( $X, Y, Z$ ) as object state, and incorporate cross-object interactions for forecasting 3D translation using graph convolutions over a object graph, similar to Andrychowicz et al. [4] and Wu et al. [141]. Since the canonical pose of an object is undefined, object orientation is not included in the object state. This model neglects object shape and appearance. The graph networks used in all baselines follow the exact design as the one we use in our model (4-layer MLPs for both the node and edge encoder). The only difference is that its inputs do not contain any object appearance features.
2. *graph-XYZ-image*, a model that uses the 3D object centroid ( $X, Y, Z$ ) and object-centric 2D image feature embeddings for forecasting 3D translation. This baseline model extracts 2D CNN features from each image, concatenates the features with the camera viewpoint, and transforms the combined features into an object appearance feature vector. The feature vector is concatenated with the 3D object centroid and fed into a graph network (identical to the one used in *graph-XYZ*) to predict future object 3D translation. When taking multiple



views as inputs, the model takes the average of the appearance feature vectors across views.

3. Visual Foresight (*VF*) [20], a model that uses the current frame and the action of the agent to predict future 2D frames by “moving pixels” based on predicted 2D pixel flow fields. It is based on the publicly available code of Ebert et al. [20] that uses such frame predictive model to infer an action trajectory that brings an object pixel to the desired (2D) location in the image space.
4. *PlaNet*[32], a model that learns a scene-level embedding by predicting future frames and the reward given the current frame. *PlaNet* only deals with single-goal tasks and does not apply to our multi-goal pushing task. We extend it to our setting by appending the goal state to the observation. In practice, we augment the latent state vector produced from its state encoders first fully connected layer with a randomly selected goal, and provide the model with reward computed correspondingly. The reward at each timestep is the computed as the negative of the distance-to-goal.

Note that both *VF* and *PlaNet* are self-supervised models that do not require ground-truth object states during training. However, we believe that since such supervision is readily accessible in simulation, we should leverage them to push the performance of the learned dynamics model. Self-supervised models are more favored when trained directly in the real world, where strong supervisions are not available, but as we showed in our experiments, our model trained solely in simulation can transfer reasonably well to the real world without any fine-tuning. As a result, we believe including the comparison with such self-supervised baselines is arguably fair and reasonable.

### 4.3.1 Data collection details

**Pushing** Our training data contains RGB-D video streams where the robot pushes objects which in turn can collide and push other objects on the table. We create scenes using 31 different 3D object meshes, including 11 objects from the MIT Push dataset [146] and 20 objects selected from four categories (*camera*, *mug*, *bowl* and *bed*) in the ShapeNet Dataset [9]. We split our dataset so that 24 objects are used during training. At test time, we evaluate the prediction error on the remaining 7 objects. At training time each scene contains at most two (potentially interacting) objects. At test time, we vary the number of objects from one up to five. We randomize the textures of the objects during training to improve transferability to the real world [130]. We consider a simulated Kuka robotic arm equipped with a single rod (as shown in Figure 3 of the main paper). The objects can move on a planar table surface of size  $0.6m \times 0.6m$  when pushed by the arm, or by other objects. We collect training interaction trajectories by instantiating the gripper nearby a (known) 3D object segmentation mask. We sample random pushing action sequences with length of 5 timesteps, where each action is a horizontal displacement of the robot’s end-effector ranging from  $3cm$  to  $6cm$ , and each timestep is defined to be 200ms. We record objects displacement 1 sec after the push. We place cameras at 27 nominal different views including 9 different azimuth angels ranging from the left side of the agent to the right side of the agent combining with 3 different elevation angles from 20, 40, 60 degrees. All cameras are looking at the 0.1m above the center of the table, and are 1 meter away from the look-at point. At each timestep, all cameras are perturbed randomly around their nominal viewpoints, and we

record all 27 views. At training time, our model consumes three randomly selected concurrent camera viewpoints as input. At test time, we use the 3D object detector to predict the 3D object segmentation mask, and our model is tested with either three or a single view as input, all randomly selected. All images are  $128 \times 128$ . There are 5000 pushing trajectories in the training data, and 200 pushing trajectories in the test data.

**Falling** We use the 3D meshes of the block objects introduced in Janner et al. [51], which includes cones, cylinders, rectangles, tetrahedrons, and triangles with a variety of shapes. We randomly select 1-3 objects and initialize their position by placing them on the table surface, and let one object falls freely from the air. One timestep is defined to be 40ms. All other settings are identical to the settings for pushing.

### 4.3.2 Action-Conditioned 3D Object Motion Forecasting

We evaluate the performance of our model and the baselines in single- and multi-step 3D motion forecasting for **pushing** and **falling** on novel objects in Tables 4.1 and 4.2 in terms of translation and rotation error. We evaluate the following ablations: i) using 1 or 3 camera views at the first time step, ii) using groundtruth 3D object boxes (gt-bbox) or 3D boxes predicted by our 3D detector, iii) varying camera viewpoints (random) versus keeping a single fixed camera viewpoint at train and test time. Our model outperforms the baselines both in translation and rotation prediction accuracy. When tested with object boxes predicted by the 3D object detector as opposed to ground-truth 3D boxes, our model is the least affected. *graph-XYZ-image* performs on par with or even worse than *graph-XYZ*, indicating that it does not gain from having access to additional appearance information. We hypothesize this is due to the way appearance and camera pose information are integrated in this baseline: the model simply treats camera pose information as additional input, as opposed to our model, which leverages geometry-aware representations that retain the geometric structure of the scene.

**Multi-step forward unrolling** The *graph-XYZ* baseline can be easily unrolled forward in time without much error accumulation since it does not use any appearance features. Still, as seen in Tables 4.1 and 4.2, our model outperforms it. *graph-XYZ* is oblivious to the appearance of the object and thus cannot effectively adapt its predictions to different object shapes.

**Varying number of camera views** Our model accepts a variable number of views as input, and improves when more views available; yet, it can accurately predict future motion even from a single RGB-D view. The prediction error of our single view model is only slightly higher than the model using three random views as input. As shown in Table 4.1, the *graph-XYZ-image* baseline performs the worst and does not improve with more views are available. We believe this is due to the geometry-unaware way of combining multiview information by concatenation, though the model does have access to camera poses of the input images.

**Varying camera viewpoint versus fixed camera viewpoint** We show in Table 4.1 (last 2 rows) that *graph-XYZ-image* can achieve much better performance when trained and tested on a single fixed camera viewpoint. This is a setting widely used in the recently popular learning-based visual-motor control literature [20, 26, 90, 143], which restricts the corresponding models to work only under carefully controlled environments with a fixed camera viewpoint, while ours performs competitively to these model but also handles arbitrary camera viewpoints.

Table 4.1: **3D object motion prediction test error during object pushing in scenes with two objects** for 1,3, and 5 timestep prediction horizon.

Experiment Setting	Model		T=1	T=3	T=5
3 views (random, novel) + gt-bbox	<i>graph-XYZ</i> [5]	translation(mm)	4.6	32.1	66.3
		rotation(degree)	2.8	16.7	26.4
	<i>graph-XYZ-image</i> [144]	translation(mm)	6.0	39.3	69.7
		rotation(degree)	3.4	29.8	30.7
	Ours	translation(mm)	<b>3.6</b>	<b>22.5</b>	<b>43.4</b>
		rotation(degree)	<b>2.5</b>	<b>12.0</b>	<b>20.6</b>
1 view (random, novel) + gt-bbox	<i>graph-XYZ-image</i> [144]	translation(mm)	6.0	39.3	69.7
		rotation(degree)	3.4	29.8	30.7
	Ours	translation(mm)	<b>4.1</b>	<b>23.6</b>	<b>43.8</b>
		rotation(degree)	<b>3.1</b>	<b>12.2</b>	<b>20.3</b>
1 view (random, novel) + predicted-bbox	<i>graph-XYZ</i> [5]	translation(mm)	6.7	35.4	68.2
		rotation(degree)	3.0	20.1	30.32
	<i>graph-XYZ-image</i> [144]	translation(mm)	6.6	43.1	71.2
		rotation(degree)	3.6	31.8	32.4
	Ours	translation(mm)	<b>4.3</b>	<b>25.2</b>	<b>47.0</b>
		rotation(degree)	<b>2.7</b>	<b>12.1</b>	<b>19.7</b>
1 view ( <b>fixed, same as train</b> ) + predicted-bbox	<i>graph-XYZ-image</i> [144]	translation(mm)	5.1	29.6	54.5
		rotation(degree)	2.6	11.0	16.9

Table 4.2: **3D object motion prediction test error during object falling in scenes with three to four objects** for 1,3, and 5 timestep prediction horizon.

Experiment Setting	Model		T=1	T=3	T=5
1views (random, novel) + predicted-bbox	<i>graph-XYZ</i> [5]	translation(mm)	5.2	<b>11.7</b>	278.6
		rotation(degree)	<b>5.7</b>	<b>10.4</b>	43.28
	<i>graph-XYZ-image</i> [144]	translation(mm)	8.4	17.0	620.2
		rotation(degree)	9.2	16.6	117.9
	Ours	translation(mm)	<b>5.0</b>	13.1	<b>16.4</b>
		rotation(degree)	6.1	12.6	<b>18.7</b>

### 4.3.3 Visualization of the 3D motion predictions

In Figure 4.2, we show qualitative comparison on long term motion prediction results produced by unrolling our model and the baseline model forward in time. Our model generalizes to novel objects and scenes with varying number of objects, though trained only on 2 object scenes. We show in Figure 4.3 rendered physics simulation videos using the proposed model. The latent 3D feature map of the proposed model is interpretable in the sense that we can render human-interpretable RGB images from the feature map using the learned neural image decoder. More

importantly, we can render such simulation videos from any arbitrary view, and the videos captured from different views are consistent with each other.

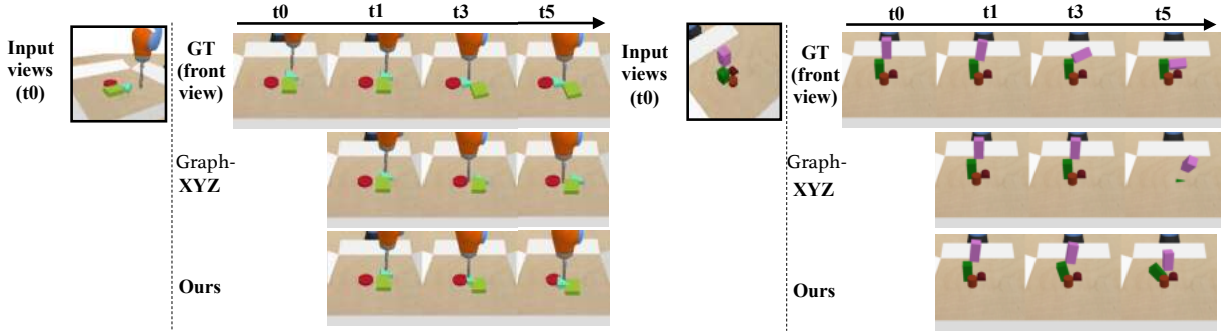


Figure 4.2: **Forward unrolling of our dynamics model and the *graph-XYZ* baseline.** Left: pushing. Right: falling. In the top row, we show (randomly sampled) camera views that we use as input to our model. The second row shows the ground-truth motion of the object from the front view. Rows 3, 4 show the predicted object motion from our model and the *graph-XYZ* baseline from the same front camera viewpoint. Our model better matches the ground-truth object motion than the *graph-XYZ* baseline. The latter does not capture object appearance in any way.

#### 4.3.4 Neural rendering and counterfactual simulations

3D-OES not only can simulate the future state of the scene, it also provides us a way to interpret the latent 3D representation and a space to run counterfactual experiments. We visualize the latent 3D feature map by neurally projecting it from a camera viewpoint to an image through a learned neural decoder, and show the resulting images in Figure 4.4. We also show that our 3D representation allows us to alter the observed scene and run counterfactual simulations in multiple ways.

#### 4.3.5 Pushing with Model Predictive Control (MPC)

We test 3D-OES on pushing objects to desired locations using MPC and report the results in Table 4.3. For our model and *graph-XYZ-image*, we use a single randomly sampled input view. For *VF* and *PlaNet*, we use a fixed top-down view for both training and testing as we found they only work reasonably well with a fixed viewpoint. Our model outperforms all baselines by a large margin. Videos of pushing object to desired locations in the presence of multiple obstacles are available on our project website: <https://zhouxian.github.io/3d-oes/>.

**Implementation Details Pushing without obstacle** We test the performance of our model with MPC to push objects to desired locations. We run 50 experiments in the Bullet simulator. For each testing sample, we place either 1 or 2 objects in the  $0.6m \times 0.6m$  workspace randomly,

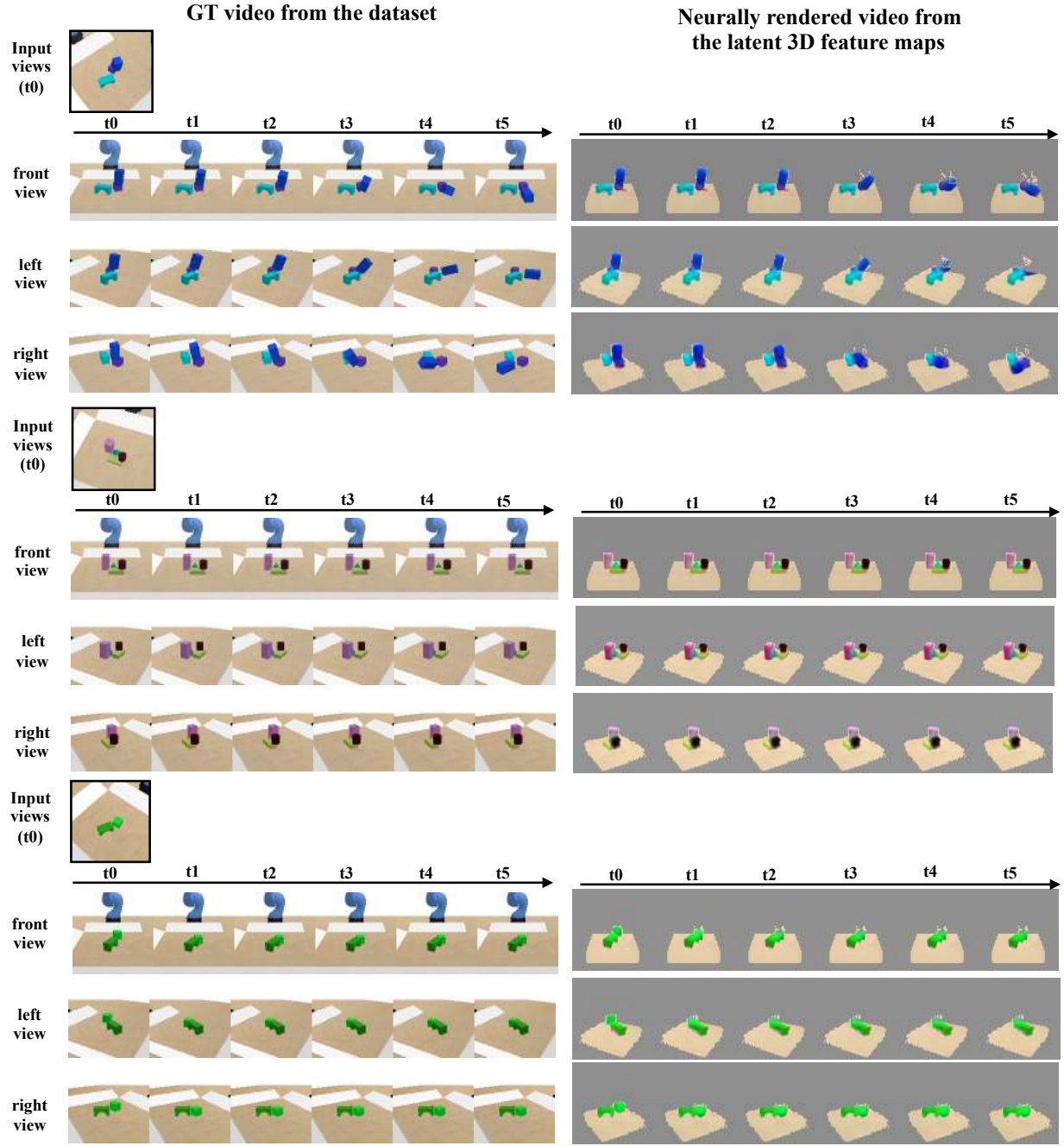


Figure 4.3: **Neurally rendered simulation videos from three different views** Left: groundtruth simulation videos from the dataset. The simulation is generated by the Bullet Physics Simulation. Right: neurally rendered simulation video from the proposed model. Our model forecasts the future latent feature by explicitly warping the latent 3D feature maps, and we pass these warped latent 3D feature maps through the learned 3D-to-2D image decoder to decode them into human interpretable images. We can render the images from any arbitrary views and the images are consistent across views.



Figure 4.4: **Neurally rendered simulation videos of counterfactual experiments.** The first row shows the ground truth simulation video from the dataset. Only the first frame in this video is used as input to our model to produce the predicted simulations. The second row shows the ground truth simulation from a query view. Note that our model can render images from any arbitrary view. We choose this particular view for better visualization. The third row shows the future prediction from our model given the input image. The following rows show the simulation after manipulating an objects (in the blue box) according the instruction on the left most column.

and sample a random goal for each object. The maximum distance of the goal to the initial position for each object is capped at  $0.25m$ . For our model and *graph-XYZ-image*, we use a single randomly sampled view. For *VF* and *PlaNet*, we use a fixed top-down view for both training and testing. We set the maximum number of steps for each action sequence to be 10, and evaluate 30 random action sequences before taking an action. We use planning horizon of 1 since greedy action selection suffices for this task. The results are reported in Table 3 in the main paper. Note that we also train and test variants of *VF* and *PlaNet* to take observations from varying camera viewpoints, together with camera pose information. However, they both fail completely on this task.

**Collision-free pushing** In order to test our models multi-step prediction performance, we evaluate our model on pushing in scenes with randomly sampled obstacles, and the robot is required to push an object to desired goal without colliding into any obstacle. For quantitative evaluation, we randomly place an object of interest and a goal position in the planar workspace. One obstacle object is placed between them with a small perturbation, so that there exists no straight collision-free path to reach the goal. The distance from the object to its goal is uniformly sampled from

Table 4.3: Success rate for pushing objects to target locations.

<i>graph-XYZ</i> [5]	<i>graph-XYZ-image</i> [144]	<i>VF</i> [20]	<i>PlaNet</i> [32]	Ours	Ours-Real
0.76	0.70	0.32	0.16	<b>0.86</b>	0.78

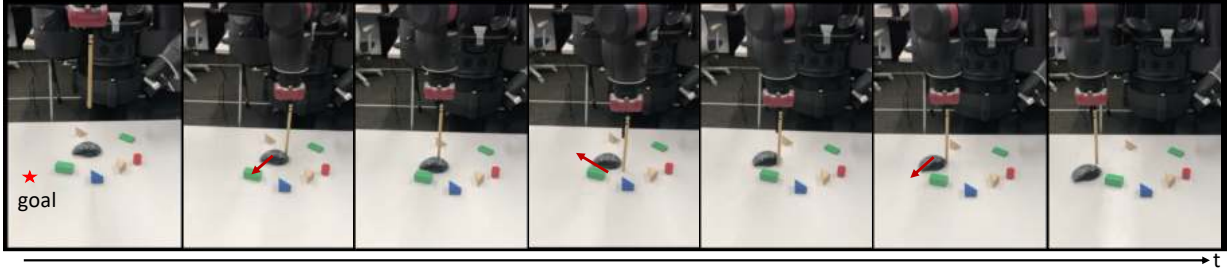


Figure 4.5: **Collision-free pushing on a real-world setup.** The task is to push a mouse to a target location without colliding into any obstacles. Our robot can successfully complete the task with 3 push attempts.

the range  $[0.24m, 0.40m]$ . Similarly, we run 50 examples, and use only one randomly selected camera view as input to our model. We evaluate 60 randomly sampled action sequences with length of 25 steps, and use a planning horizon of 10 steps. We achieve a success rate of **0.68** for this task.

We randomly place *multiple* obstacles in the scene for quantitative evaluation while ensuring existence of collision-free path is non-trivial. For both with- and without- obstacle pushing, it is considered a successful pushing sequence if all objects end up within 4cm (about half of the average object size) from the target positions on average.

### 4.3.6 Sim-to-Real Transfer

We train our model solely in simulation and test it on object pushing control tasks on a real Baxter platform equipped with a rod-shaped end-effector, similar to the setting in the Bullet simulation (Figure 4.5). We attached a Intel RealSense D435 RGB-D camera to the robot’s left hand, and use only one RGB-D view as input for this experiment. The pose of the camera is different from those seen during training. Please refer to Appendix (Section C.3) for details of our real-world setup, objects selection, and 3D detector training. We report the success rate of real-world pushing in Table 4.3 (Ours-Real). Our model achieves similar success rates for pushing in simulation and in the real world. Since geometry information is shared by simulation and the real world by a large extent, and our model combines the viewpoint-invariant property of the geometry-aware representation and an object-factorized structure, it presents good sim-to-real transferrability. In Figure 4.5, we qualitatively show pushing objects along collision-free trajectories in complex scenes in the real-world setup. More results are available on our project website: <https://zhouxian.github.io/3d-oes/>.

**Implementation Details** We use a Baxter robot equipped with a rod-shaped end-effector attached to its right hand, similar to the setting in the Bullet simulation. One Intel RealSense D435



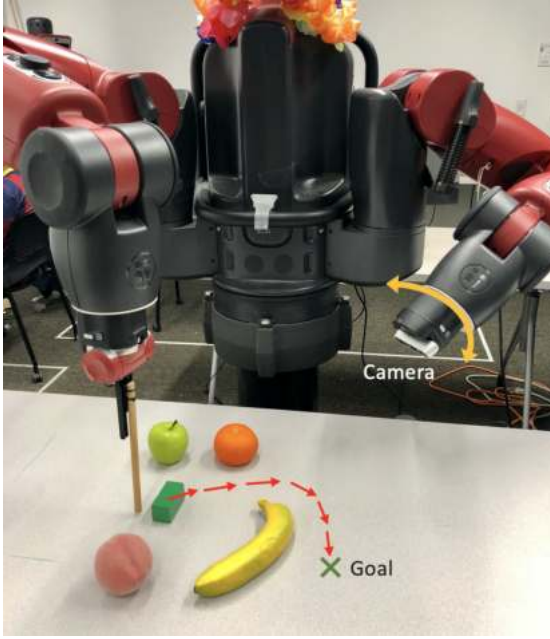


Figure 4.6: Real-world setup with Baxter



Figure 4.7: Objects for real-world experiments

RGB-D camera is attached to the robot’s left hand, and we use only one view for our experiment, as shown in Figure 4.6.

Due to reachability considerations, we down-scaled the size of the planar workspace by twice from the one in simulation, resulting a workspace of  $0.3m \times 0.3m$ . For a fair comparison, we also down-scaled with the same factor the object-to-goal distance, length of horizontal movement per action step, and size of the tolerance for determining success/failure. We pick 20 objects with size of 5 to 10cm, which are commonly seen in a office setting, including fruits, wooden blocks, and stationery, and evaluate 5 pushing samples for each of them. Some of objects selected are shown in Figure 4.7.

For object detection in the real-world, we train our 3D detector using simulated data, and fine-tune it using a small set of real data (100 images capturing 25 distinct object configurations) collected using 4 cameras. The ground truth bounding-boxes and segmentation masks are obtained via background subtraction.



# Chapter 5

## Visually-Grounded Library of Behaviors for Generalizing Manipulation Across Objects, Configurations and Views

### 5.1 Introduction

In Chapter 4, we have talked about how we can do model-based control by learning a general 3D object dynamics model that is robust to camera viewpoints. However, learning these dynamics models requires supervised training on ground truth 3D object trajectories, which are hard to obtain in many real world scenarios. Can our agent learn to manipulate objects even without explicitly modeling the low-level object dynamics? In this Chapter, we study how the visual representations can be used in a model-free manipulation setup.

When robots make their way out from factories into peoples houses, they are expected to be able to manipulate a diverse set of objects that might appear in a house. Such a goal is challenging since it is hard to predict what a robot might encounter during execution. Possibly, the robot would need to grasp previously unseen objects placed in arbitrary locations, and it might be difficult to observe an object in its most familiar views due to obstacles that might block certain viewpoints. In this paper, we explore how we can best **incorporate visual information** to learn robot behaviors **that generalize to novel objects, configurations and views**.

Current policy learning research focuses on *discovering* new behaviors rather than *transferring* behaviours across diverse objects and camera views. Existing works in reinforcement learning (RL) mostly considers the same objects at training and test time and uses state representations that retain 3D object locations and velocities [31, 89] but abstract away object shape, color or texture, which might provide critical information regarding what types of friction and contact models to use. Although such a state abstraction makes the state-to-action mapping easier to learn, it does not allow the policy to adapt across different objects. Methods that do attempt to generalize across objects learn a mapping from images-to-action [67], depth-to-action [74] or pointcloud-to-action [82, 96]. They have been successful in object grasping [74, 91] and object pushing from a fixed camera view [2, 67]. Nevertheless, these methods have shown to be hard to generalize across different camera viewpoints [14] and objects. Works that attempt to transfer

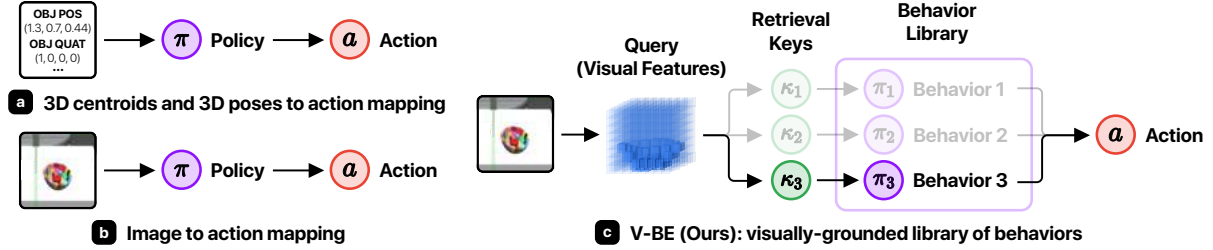


Figure 5.1: We propose a novel policy representation that generalizes to unseen objects and camera views. In contrast to prevalent approaches that learn *state-to-action* or *image-to-action* mappings, our proposed model decomposes a policy into a behavior selection module that uses visual observations and a library of behaviors to select from that uses abstract state representations as inputs.

visuomotor policies learned in simulation to the real-world often require identical placement of the camera in the real world [50, 89].

Lack of generalization in existing visuomotor approaches is attributed to: (a) lack of state abstraction and (b) occlusions, foreshortening, and other artifacts caused by camera projection in images. For a policy to generalize to a new object, it is often useful to find a state abstraction, a minimal subset of environment features useful for performing a task [62]. Abstractions facilitate transfer by explicitly ignoring parts of the environment that are irrelevant to a particular skill. For example, when opening a bottle, the size of the cap is a relevant feature, but the color of the bottle itself is irrelevant. Policies that attempt to learn what features matter automatically often require large amounts of training examples. Aside from a lack of state abstraction, existing visuomotor policies are sensitive to changes in the pixel values in the image space. Appearance and location of the object and the gripper and their relative arrangement, which are constant in 3D, varies with the viewpoint in 2D images, which causes these policies to fail under camera viewpoint changes. Changing the inputs to 3D point clouds can potentially reduce the effect from projection artifacts, yet the quality and completeness of the point cloud still depends on the number of available views and the viewing angles.

To improve generalization, we propose Visually-grounded library of BEhaviors (V-BEs), a policy representation that learns to perform manipulation tasks with varying objects, initial and goal configurations, and under varying camera viewpoints. In place of prevalent approaches of learning a *flat* image-to-action or object-to-action mapping (see Figure 5.1(a)(b)) which operates on a single representation, our model learns to disentangle and operate on two separate representations in a *hierarchical* setup (see Figure 5.1(c)). At the higher level of the hierarchy, a selector selects an appropriate behavior to execute among a library of given behaviors, part of the lower level of the hierarchy. This selector learns a 3D object feature representation that captures the **static** object properties, such as object shape and affordances, via a combination of self-supervised view-prediction and object interaction prediction. The former makes the representation robust to changing camera views and occlusions by learning to complete the missing information from current views, while the latter helps the representation encode affordance information (i.e., how objects change by applying a behavior). At the lower level of the hierarchy, there are a library of distinct behaviors, each of which uses an abstracted state representation that captures **dynamic** properties of entities in the environment such as object and gripper positions

at the current time step. Different behaviors can be redundant, use different state representations, and be learned or manually engineered with different algorithms [3, 25, 71].

We test the proposed model in grasping and pushing a large pool of diverse objects. We show it can generalize to unseen objects at test time with varying object starting positions, initial poses, goal locations, and camera views and outperforms existing single-policy image-to-action mapping [67] or object-to-action mapping that uses only the fast changing 3D object locations [89]. We further show our model trained in simulator can be directly transferred to a real robotic platform, and can significantly improve its performance by fine-tuning only on a handful of interaction labels. In the real robot setup, the robot can use camera views that are totally different from those used at training time in the simulator and still get good task performance.

## 5.2 Method

Our policy  $\Pi$  is a visually-grounded library of behaviors, as shown in Figure 5.2. V-BEs are comprised of multiple behaviors  $\{\pi_i \mid i = 1, 2, \dots, K\}$  and a behavior selector  $G$ . Each behavior  $\pi_i$  is either an open loop trajectory generator, or a closed-loop policy (feedback controller) that tracks objects over time. Since most behaviors are developed using abstract state representations, namely, 3D object locations and 3D poses and do not take into account object shape or appearance, each of them can handle only a subset of objects and their 3D orientations. To be able to handle a diverse range of objects, we need to decide which behavior to use under which circumstance. To achieve this, we propose a behavior selector which is a gating network that, given the sensory observations  $\phi_t$ , predicts the behavior that can best manipulate the object under consideration to its desired configuration. Our sensory input to action mapping then, at each time-step reads:

$$\Pi(a_t|o_t, g) = \sum_{i=1}^K G(o_t)_i \pi_i(a_t|o_t, g), \quad (5.1)$$

where  $g$  is the goal configuration to achieve,  $o_t = \{\mathbf{I}_{\mathbf{v}_t}, \mathbf{v}_t\}$  is the sensory observation in the form of one or more RGB-D images  $\mathbf{I}_{\mathbf{v}} \in \mathbb{R}^{\tilde{H} \times \tilde{W} \times 4}$  captured from arbitrary but known camera poses  $\mathbf{v} \in \mathbb{R}^7$ , and  $G(o_t)_i \in \{0, 1\}$  denotes the selector’s output for  $\pi_i$  at time step  $t$ .

Our main contribution is the factorization of the visual properties of the object presented in the sensory observation  $o_t$ , so that, the behaviors  $\{\pi_i \mid i = 1, 2, \dots, K\}$  and the selector  $G$  operate on separate object visual properties. The behaviors use as input *dynamic* visual properties that change throughout the manipulation episode, such as object or part 3D positions, and abstract away object appearance information. Let  $f^{\pi_i}(o_t)$  denote the state abstraction behavior  $i$  uses. In contrast, the selector  $G$  takes as input the *static* object visual properties. It takes the raw sensory observation  $o_t$  and transforms it into a location invariant visual representation of the object  $f^G(o_t)$ , which captures object appearance and 3D size, which remain constant throughout the manipulation. With such a distinction on the representation used by the two modules, we can rewrite Equation equation 5.1 into:

$$\Pi(a_t|o_t, g) = \sum_{i=1}^K G(f^G(o_t))_i \pi_i(a_t|f^{\pi_i}(o_t), g). \quad (5.2)$$

One advantage of having this decomposition is that each behavior  $\pi_i$  can then use its own state abstraction  $f^{\pi_i}(o_t)$ , such as object 6D-poses, axis-aligned bounding boxes (as used in this paper), part-based 3D boxes, or object keypoints locations [97]. This framework supports integrating a wide range of existing models, representations, and behaviors as selectable behaviors. This architecture is also recursive, in that any behavior can consist of the same architecture replicated at a lower level. We consider here single object manipulation skills but our formulation can generalize to multiple objects.

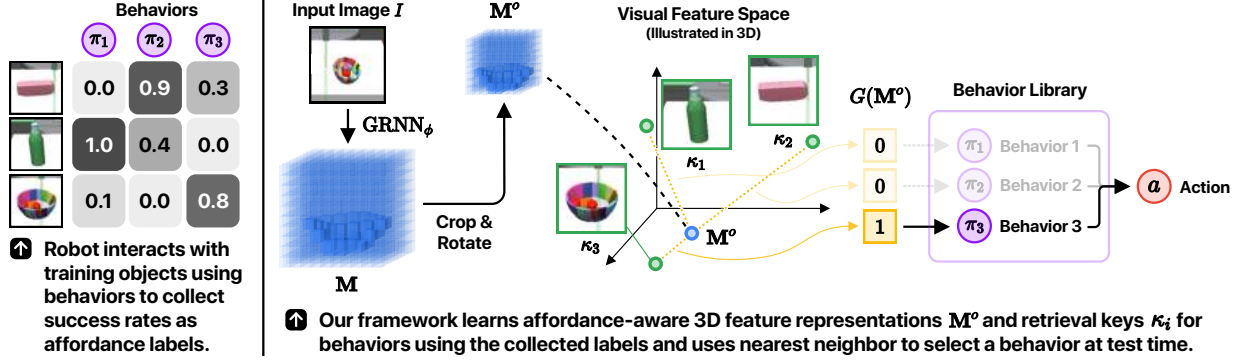


Figure 5.2: Overview of the proposed framework. Our model consists of (a) a behavior selector  $G$  that learns to map RGB-D images  $I$  to an affordance-aware, view-invariant 3D feature space that reflects how objects change by applying a behavior, and (b) a library of behaviors, where each behavior  $\pi_i$  can either be a controller or a policy learned via RL.

In the rest of this section, we first explain the architecture and training details of the behavior selector in section 5.2.1, and then explain how we acquire a library of behaviors for the two robot manipulation tasks we evaluate our framework on in section 5.2.2.

### 5.2.1 Visually-Grounded Behavior Selector

The behavior selector  $G$  is a nearest neighbor classifier that uses the visual feature representation of the object to manipulate  $f^G(o_t)$  as a query to retrieve the behavior that has the highest propabability to manipulate the object successfully. Each behavior  $\pi_i$  is associated with a learned retrieval key  $\kappa_i$ .

**View-invariant Feature Learning Self-supervised by View prediction:** The behaviour selector should ideally be viewpoint invariant as behaviors should not depend on camera placement in the environment in most cases. To achieve this, we design the selector  $G$  to operate on a 3D viewpoint-invariant object-centric feature representation  $f^G(o_t) = M^o \in \mathcal{M} = \mathbb{R}^{H \times W \times D \times C}$ , which is a set of 3-dimensional feature maps centered around the object of interest. The retrieval key  $\kappa \in \mathcal{M}$  is also learned in the same representation space. Both  $M^o$  and  $\kappa$  are with the size of  $64 \times 64 \times 64 \times 32$  in the experiments. We build upon recent advances in 3D perception and use geometry-aware networks (GRNNs) [135] to map a single RGB-D image or a set of multi-view images to a complete 3D feature representation of the scene the image(s) depict. GRNNs learn to complete the missing information from a single view by optimizing end-to-end for view prediction using multi-view data collected by the robot [135]: an input RGB-D image is un-projected

into an (incomplete) 3D feature grid, missing information is “inpainted” via a series of 3D convolutions, and the completed 3D feature grid is projected to a novel viewpoint and decoded to the corresponding RGB-D image. The view prediction loss reads:

$$\mathcal{L}_{\text{view-pred}}(\phi) = \sum_{n=1}^N \|P_{\phi}(\text{GRNN}_{\phi}(\mathbf{I}_{\mathbf{v}^n}, \mathbf{v}^n), q) - I_q^n\|_2, \quad (5.3)$$

where  $\text{GRNN}(\mathbf{I}_{\mathbf{v}}, \mathbf{v})$  is a function that lifts RGB-D input images  $\mathbf{I}_{\mathbf{v}}$  capturing from camera poses  $\mathbf{v}$  to a geometry-consistent 3D feature map  $\mathbf{M} = \mathbb{R}^{\bar{W} \times \bar{H} \times \bar{D} \times \bar{C}}$  with  $\bar{W}, \bar{H}, \bar{D}$  denoting the spatial dimension and  $\bar{C}$  denotes the feature dimension of the 3D feature map,  $P(\mathbf{M}, q)$  is a projection function that projects the 3D feature map from a query viewpoint  $q$  and decodes to a target image,  $I_q^n$  is the target image to predict, and  $\phi$  is the neural network weights of GRNN. From the scene map  $\mathbf{M}$ , we obtain object-centric feature representation  $\mathbf{M}^o$  by cropping the scene map using a fixed-size axis-aligned box, centered around the object we wish to manipulate.

Our selector  $G$ , given a 3D object feature representation  $\mathbf{M}^o$  of the object to be manipulated and a retrieval key  $\kappa_i$  for behavior  $\pi_i$ , computes the probability  $\hat{G}_i$  that  $\pi_i$  can successfully manipulate the object to its desired goal location:

$$\hat{G}(\mathbf{M}^o)_i = \sigma(\langle \mathbf{M}^o, \kappa_i \rangle) \in [0, 1], \quad (5.4)$$

where  $\langle \cdot, \cdot \rangle$  is the inner product operation and  $\sigma$  is the sigmoid function. It then selects the behavior with the highest predicted probability:

$$G(f^G(o_t))_i = G(\mathbf{M}^o)_i = \mathbb{1}\{i = \arg\max_{i'} \hat{G}(\mathbf{M}^o)_{i'}\}. \quad (5.5)$$

**Affordance-aware Feature Learning Self-supervised by Interaction:** The 3D feature representations obtained through view prediction capture how objects look, but not how they can be manipulated. Our selector should operate over representations that capture information about how an object can be used, and how it will react to an agent’s actions. We will use the term *affordance*, coined by Gibson in 1966 [30], to denote this information.

We learn affordance-aware 3D feature representations by applying behaviors in the library to the objects in the training set, and update the object representations and the retrieval keys  $\kappa_i, i = 1 \cdots K$  for behaviors  $\pi_i, i = 1 \cdots K$  according to the success rate a behavior achieves on a particular training object, as shown in Figure 5.2. Let  $u_{ij} \in [0, 1]$  be the success rate after  $N_{\text{attempt}}$  trials of behavior  $\pi_i$  on object  $\omega_j$ , under potentially varying goals  $g$ . Here, we treat same objects with different initial orientation as different objects since we assume that canonical poses of objects are not given. Each trial encounters the same object in a potentially different initial location, orientation and desired goal location. Our loss then reads:

$$\begin{aligned} \mathcal{L}_{\text{affordance}}(\kappa_1 \cdots \kappa_K, \phi) = \\ \sum_{i=1}^K \sum_{j=1}^{|\Omega_{\text{train}}|} \sum_{n=1}^{N_{\text{attempt}}} \text{BCE}(\mathbb{1}\{u_{ij} \geq \delta\}, \langle R_{\phi}(I_n^{\omega_j}), \kappa_i \rangle), \end{aligned} \quad (5.6)$$

where  $\text{BCE}(p, l) = -p \cdot \log(\sigma(l)) + (1 - p) \cdot \log(1 - \sigma(l))$  is the standard binary cross entropy loss,  $\delta$  is a hyperparameter for thresholding the success rates,  $I_n^{\omega_j}$  are the initial RGB-D images

drawn from tasks with object  $\omega_j$  in the  $n^{\text{th}}$  trial,  $R(I)$  is the 2D-to-3D visual feature extractor that takes the cropped object-centric 3D feature representations from GRNNs, and  $\phi$  is the GRNNs weights. The final objective for training the affordance-based visual features is

$$\underset{\kappa, \phi}{\text{minimize}} \mathcal{L}(\kappa, \phi) = \mathcal{L}_{\text{view-pred}}(\phi) + \lambda_a \cdot \mathcal{L}_{\text{affordance}}(\kappa, \phi), \quad (5.7)$$

where  $\lambda_a$  is a hyperparameter for balancing the two losses.

## 5.2.2 Building a Library of Behaviors

The visually-grounded behavior selector selects plausible behaviors to manipulate a specific object from a library of behaviors. But how can we obtain this library in the first place? Indeed, any existing behaviors, whether engineered or learned using reinforcement learning or imitation learning, can be included in our library. In this paper, we consider two common manipulation tasks: pushing and grasping, and build appropriate behavioral libraries for each.

For pushing, the behaviors are deterministic goal conditioned policies  $a_t = \pi(s_t, g)$  that map a state of the environment and the robot  $s_t = [s_t^e, s_t^r]$  to an action at time step  $t$ . In our case, the state of the environment is the 3D object centroid  $s_t^e = f(o_t, O)$  and the robot state (gripper 3D location, pose, and whether it is opened or closed). The action includes 3D motion, opening (position control), and closing (force control) of the gripper. We use a total of 25 goal conditioned policies – one is trained from the whole set of objects, while the others are trained on disjoint subsets of object configurations organized based on object category and initial poses. We train all policies using deterministic policy gradients (DDPG) [99] with goal relabelling (HER) [3] while randomizing initial and goal object 3D locations.

For grasping, we design controllers  $\pi(a_t|g; p^{grasp}, q^{grasp})$  which given a 3D grasping point  $p^{grasp} \in \mathbb{R}^3$  relative to the center of the object and a grasping 3D angle  $q^{grasp} \in \mathbb{R}^2$ , move the gripper (open loop) to the grasping 3D point location, close it, and move it to the desired goal location. The grasping angle  $q^{grasp}$  consists of two numbers describing the yaw of the gripper and the elevation angles between the gripper and the table surface. When the elevation angle is smaller than 90 degrees (not top-down grasps), we constraint the gripper to point toward the center of the object on the x-y plane. We manually select 30 different controllers including top-down grasps with different yaw orientations (top-grasps) and grasps from the side with different elevation angles of the gripper (side-grasps). We empirically found that these parameterized controllers are quite stable and can be shared across multiple objects.

## 5.3 Experiments

Our experiments aim to answer the following questions: (1) Does the proposed library-based approach outperform existing methods that use a single combined perception and policy module, either using 2D images, 3D object locations, or 3D scene feature maps as input? (2) Is the proposed view-invariant and affordance-aware 3D feature representation a necessary choice for the selector? (3) How do V-BEs compare to state-of-the-art methods, specifically, in the grasping domain? (4) Does the method work on a real robot? We test our model on grasping and pushing

a wide variety of objects in the MuJoCo simulator [131] and further test the grasping module on a real-world Franka Panda robot arm.

### 5.3.1 Simulation Experiment Setups

Our simulated environment consists of a Fetch Robot equipped with a parallel-jaw gripper. The robot is positioned in front of a table of height  $0.4m$ . To obtain the visual observations, on each episode we choose 3 random cameras from cameras placed at 30 nominal different views including 10 different azimuths ranging from  $0^\circ$  to  $360^\circ$  combined with 3 different elevation angles from  $20^\circ$ ,  $40^\circ$ ,  $60^\circ$ . All cameras are looking at the center of the table top, and are 0.5 meter away from that point. All images have size  $128 \times 128$ .

**Task Descriptions:** In the grasping task, the agent has to grasp an object and move it to a specified target location above the table. We use 274 distinct object meshes from 6 categories in ShapeNet [9] including toy buses, toy cars, cans, bowls, plates, and bottles. We randomly split the dataset into 207 training objects, and 67 testing objects. After augmenting the meshes with random scaling from 0.8 to 1.5 and random rotations around the vertical z-axis, we get a total of 800 distinct object configurations (object instance and pose), 600 for training and 200 for testing. At the start of each episode, an object is placed in an area of  $30cm \times 16cm$  around the center of the table, and a goal is sampled uniformly  $10 \sim 30cm$  away from the gripper’s initial position. An episode is successful if the object centroid is within  $5cm$  of the target at the final timestep.

In the pushing task, the agent has to push an object placed on the table to a specified target location. We use 100 objects from 12 categories in ShapeNet [9]: baskets, bowls, bottles, toy buses, cameras, cans, caps, toy cars, earphones, keyboards, knives, and mugs. After augmentation and splitting to train and test sets, we obtain 615 training object configurations and 200 for testing. The initial and the goal position of the object are both uniformly sampled to be within  $15cm$  of the center of the table along both x-axis and y-axis, although we resample if that location is already in the goal area. An episode is successful if the object centroid is within  $5cm$  of the goal within 50 timesteps.

**Baselines:** We compare our method with various learning and non-learning based methods for object manipulation:

- (a) *Single Behavior w/ Abstract 3D State (Abstract 3D)* [3, 99]: a policy takes as input ground truth 3D bounding box of the object and gripper and outputs actions.
- (b) *Single Behavior w/ Abstract 3D State and 2D Images (Abstract 3D + Image)*: a policy takes both RGB-D images and the ground truth 3D bounding box as inputs and outputs actions. Our architecture resembles that of [147], but we further include ground truth object position as extra inputs to the model. For fair comparisons to other methods, the model only takes as input the current state as opposed to the states in 5 past steps, as in [147].
- (c) *Single Behavior w/ 3D Feature Tensor (Contextual 3D)*: a policy that takes as an input RGB-D images and the ground truth 3D bounding box and outputs actions. Different from (b), the model first transforms the image into a view-invariant 3D feature tensor using GRNNs [135], then converts the 3D feature tensor into a feature vector through three 3D-convolutional layers and a fully connected layer, and finally concatenates it with the rest of the inputs to predict actions.

- (d) *Ours, Library of Behaviors w/ Visual Selector (V-BEs)*: Our model takes the same input as (b) and (c). The 3D bounding boxes are used as input to all the behaviors. The RGB-D images are transformed into 3D affordance-aware visual features and treated as input to the selector.

We train the baselines with different learning methods including behavior cloning [67], DDPG-HER [3, 71] and DAGGER [106]. We report the best performance we got by training the model with these different methods. We also attempt to make all the models to have similar number of parameters so the comparison is fair. However, larger networks are empirically harder to train and do not converge well, so we instead increase the number of parameters in a smaller networks until its performance saturates. For pushing, we found that using DDPG-HER is enough to learn a good *Abstract 3D* policy from scratch. For *abstract 3D + Image*, we found it is critical to use behavior cloning from expert demonstrations to obtain good policies. The expert demonstrations are obtained from trained expert policies on single objects. We follow the architectural design and training of [147] to learn the image-to-action mapping. For *Contextual 3D*, we include DAGGER to enforce behavior cloning during execution. To train the grasping policies, we further include human demonstrations in the replay buffer when training it with DDPG-HER. Both *abstract 3D + Image* and *Contextual 3D* are trained with DAGGER since offline behavior cloning is insufficient.

### 5.3.2 Single Behavior versus a Library of Behaviors

We compare the proposed model with models that do not use a library-based approach, i.e., single behavior approaches. As shown in Table 5.1, our method outperforms all the single behavior baselines. *Abstract 3D* performs well, but since it does not use any visual information, its performance saturates at around 0.8 for pushing and 0.3 for grasping. *Abstract 3D* performs poorly for grasping. The learned behaviors do not transfer well to new objects. Adding a 2D image helps, but not dramatically (see *Abstract 3D + Image* in Table 5.1). Although 3D feature maps obtained from GRNNs are semantically rich and can handle varying viewpoints, the mapping to actions is harder to learn due to the higher dimensionality of the 3D scene map, resulting in under-fitting models. Our model takes advantage of both abstract and semantically rich representation and thus can handle better object variability and transferability. The combinatorial nature of the proposed method allows the model to capture the multi-modality in trajectory generation.

	Single Behavior			library of behaviors
	Abstract 3D [3, 99]	Abstract 3D + Image	Contextual 3D	V-BEs (Ours)
grasping	0.30	0.35	0.20	<b>0.78</b>
pushing	0.83	0.70	0.10	<b>0.88</b>

Table 5.1: Success rates on grasping and pushing unseen objects.



### 5.3.3 The necessity of building the selector with the proposed view-invariant and affordance-aware 3D Representations

Next, we show the importance of using view-invariant 3D visual feature representations and the importance of self-supervised training with interaction labels. We compare our method with two baselines: (a) a model with a selector that operates over 3D visual feature representation learned only with the view and occupancy prediction loss, as suggested in [34, 135], without fine tuning with interaction labels, and (b) learning the visual affordance features using 2D visual features extracted from 2D CNNs. Our method significantly outperforms these two baselines, which shows the importance of fine-tuning on the execution labels as well as the use of 3D feature representations for behaviour selection.

	2D Features	Proposed Method (3D Features)	Proposed Method w/o Fine-tuning on Interaction Labels [135]
Grasping	0.46	<b>0.78</b>	0.31
Pushing	0.81	<b>0.88</b>	0.46

Table 5.2: Success rates on grasping and pushing unseen objects using selector with varying representations.

**Visualizing Behavior Clusters** To understand the learned affordance-aware visual feature representations, we cluster the testing objects in the pushing task based on which behavior is selected to be executed when the selector is presented by an object. We present a sample of behaviors and the corresponding object clusters in Figure 5.3. As seen in the visualization, the learned feature representation in the affordance-aware behavior selector represents objects that are close in affordance in neighboring regions of the feature space. It is also robust to variations of object colors, sizes, and semantic categories. For example, the behavior trained on knives (last row in the figure) is predicted to be able to handle both very thin keyboards and knives; the behavior trained on bottles (fourth row in the figure) is predicted to be able to handle both bottles and small cans.

### 5.3.4 Comparison with other grasping baselines

We further compare the proposed model with current state-of-art methods in grasping: (a) **DexNet [74]**: a state-of-the-art top-grasps method which, given a top-down depth map, generates grasps as the planar position, angle, and depth of a gripper relative to an RGB-D sensor. (b) **6-DOF Grasp-Net (GraspNet) [82]**: a state-of-the-art grasping method that describes the grasp as a full 6-degrees-of-freedom (DoF) pose. For (a)(b), we use the publicly released code and model provided by the authors, finetune it with interactions in our training environments, and evaluate it in our test environments. (c) **Our Model with Only Top-grasps**: an ablated baseline using the proposed methods and include only the top-grasps. (d) **Our Model with object detector**: our model performs with estimated object bounding box. To remove the effect of the detector, all other baselines use ground truth object locations to obtain the corresponding depth images or object locations.

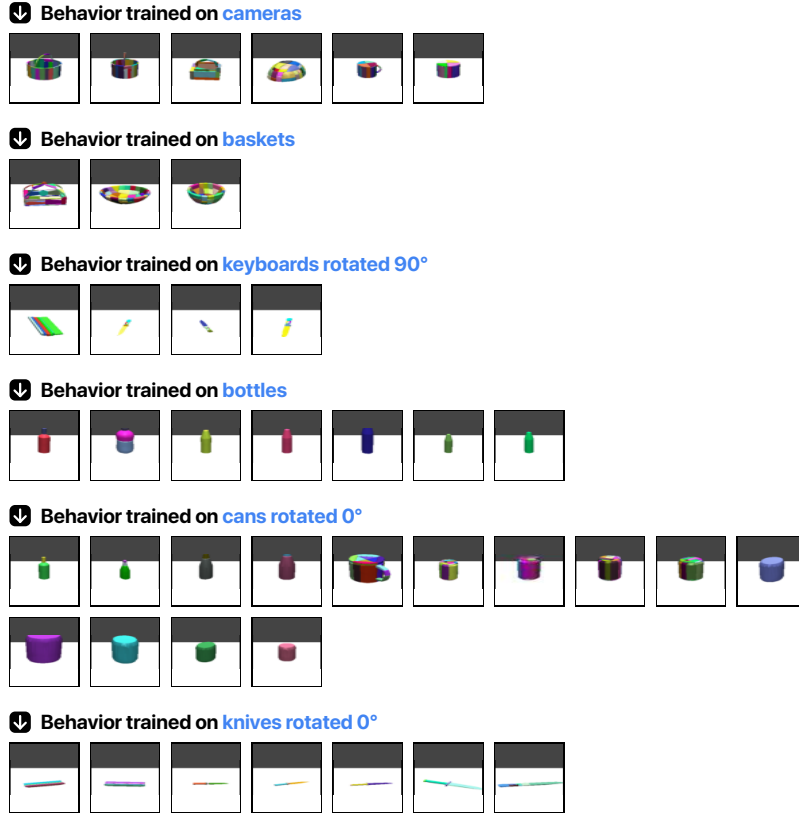


Figure 5.3: Visualization of a sample of behaviors and their corresponding object clusters. In each section of the figure, the behavior described after the arrow is the output of the affordance-aware behavior selector when it takes any of the objects visualized in the section as input.

As shown in Table 5.3, our method outperforms all baselines by a large margin, even with an estimated object bounding box. *Dexnet* performs the best among the baselines, but it can only handle depth maps from a top-down view while all other methods can use any camera view. *GraspNet*, to our surprise, performs poorly, especially when we randomize object initial and goal locations. Many grasps proposed by *GraspNet* often turn out to be unstable when the object is placed close or far away from the robot. Our model handles variances in object locations well because we include in the behavior library general top-down, left-hand, and right-hand grasping behaviors that are more stable across object locations. Our approach outperforms these methods by a large margin even only using very naive controllers in the behavior library.

DexNet [74]	GraspNet [82]	V-BEs w/ Top-Grasps	V-BEs (Ours)	V-BEs w/ detector
0.72	0.36	0.67	<b>0.78</b>	0.73

Table 5.3: Success rates on grasping for unseen objects.

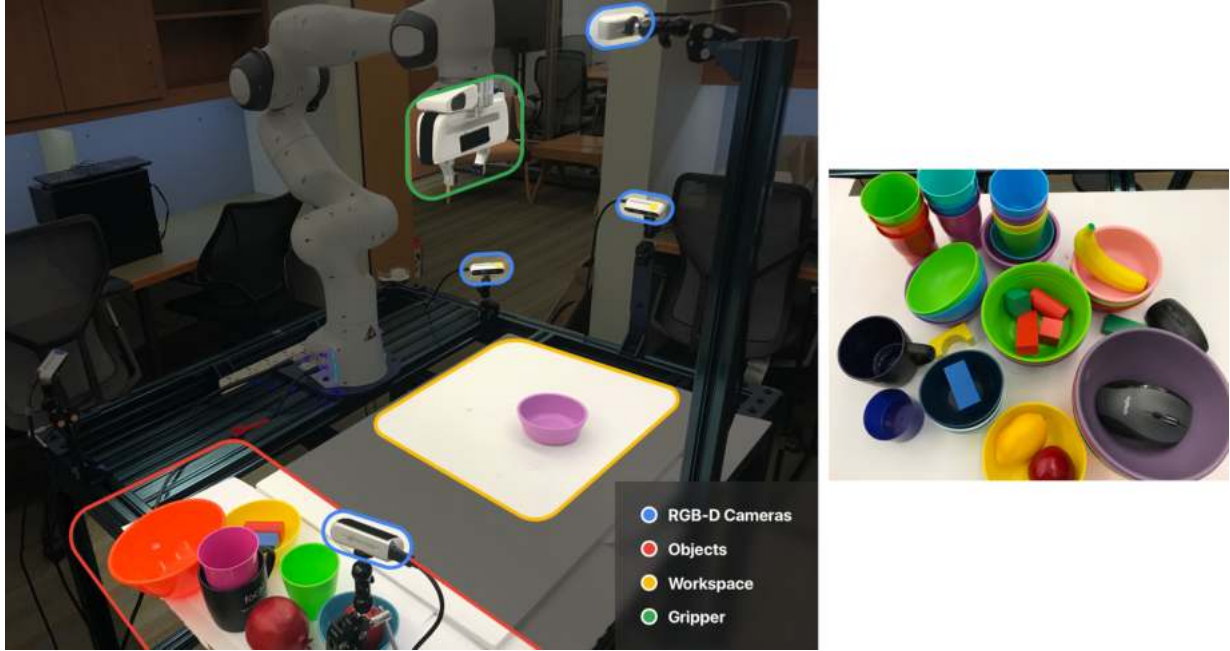


Figure 5.4: Real robot setup (left) and objects used (right).

### 5.3.5 Real robot results

To demonstrate that our V-BEs representation works in a real robot setup, we measure the performance of our model on a real 7-DOF Franka robot arm equipped with a parallel-jaw gripper for the grasping task (see Figure 5.4). We set up 4 realsense RGB-D cameras that have full view of the workspace around the center of the table. In each trial, an object is placed in a  $20\text{cm} \times 10\text{cm}$  region on the table, and a goal is sampled uniformly away from the objects start position from  $-20\text{cm}$  to  $20\text{cm}$  in all  $x, y$ , and  $z$  dimensions. An episode is considered successful if the object centroid is within  $5\text{cm}$  of the goal position.

We use the same set of grasping behaviors as in our simulated experiments. We randomly select 3 views from 4 possible camera views to obtain RGB-D images as inputs to the learned selector. We randomly split 56 objects including computer-mouses, bananas, bowls, mugs and wooden bricks (see Figure 5.4) into a training set and a test set, each with 28 objects. Without further fine-tuning, our model trained in simulated environments achieves success rate of 46.4% on the test set, even though the camera views and objects are unseen at training time. To improve model performance in the real world setup, we collect interaction labels by running the grasping behaviors on the training objects and use the labels to finetune the visual selector with the objective specified in Equation equation 5.7. After fine-tuning, our model can achieve success rate of 82.1% on the test set. Qualitative results are shown in Figure 5.5.

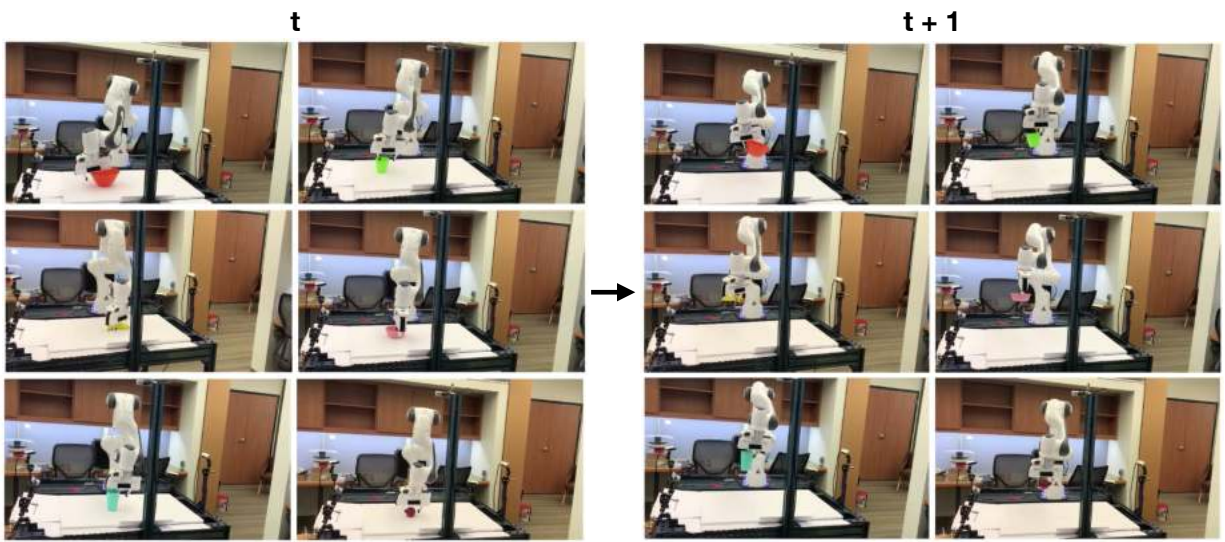


Figure 5.5: **Grasping Results on a real robot.** The robot can successfully pick up different objects and transfer it to a target location in the air.

## **Part III**

**Concept learning: Learning to construct  
memory and associate current observations  
with past memory**



# Chapter 6

## Unsupervised learning of 3D visual Concepts by Corresponding and Quantizing Detected Objects

### 6.1 Introduction

The goal of this chapter is to explore how the unsupervisedly learned 3D feature representations can be used for recognizing unseen objects and infer the objects' 3D poses in 2D images. The ability to recognize objects under varying poses, sizes, lighting conditions, and camera view-points is fundamental for humans and animals to track and interact with diverse objects.

While humans and animals acquire this ability through evolution and interacting with the world under a moving visual sensor—their eyes—, most existing computer vision models still rely on supervised training on massive labelled images to learn to recognize objects and their poses [39, 132]. In robotics, most works assume a closed world of predefined 3D object models, e.g., 3D object meshes, and do not handle objects that cannot be explained by a 3D model [85]. Some works propose to learn a mapping from the images to the parameters of the 3D models, but these models usually need the 3D models of the objects in the first place, to generate a large synthetic dataset with 3D poses and category annotations for learning the mapping [75, 122, 125]. Few-shot object detection methods [60, 118, 138] use a support sample to quickly classify a query sample, but remain in 2D image space and do not infer 3D object orientation, rather object label. Recognizing familiar objects and detecting their 3D locations, poses and scales in images without 3D annotations remains elusive.

Our key intuition in this work is to represent objects in terms of **3D feature representations** inferred from the input RGBD images, and infer alignment between two objects by explicitly rotating and scaling their representations during matching. While current state-of-the-art (SOTA) models for object detection and pose estimation represent an object as a feature vector or 2D feature maps [39, 78, 98], our model represents objects as a 3D feature representation inferred from 2.5D (RGBD) input images, which can be explicitly scaled, rotated and compared in 3D. Different from methods in robotics research that infer explicit 3D geometry of an object in terms

<sup>0</sup>This chapter is based on the paper published previously on CVPR 2020 MVM workshop [93].

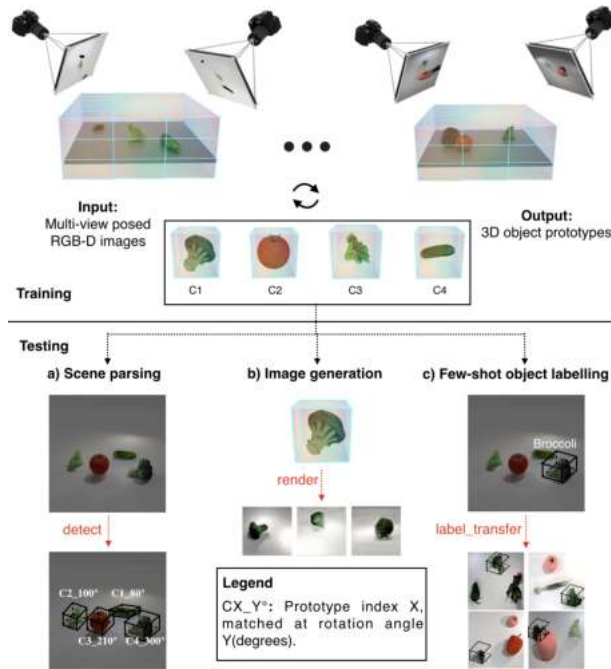


Figure 6.1: **Top: Model overview.** Our model takes as input RGB-D images of scenes, and outputs 3D prototypes of the objects. **Bottom: Evaluation tasks.** (a) Scene parsing: Given a new scene, we match each detected object against the prototypes using a rotation-aware check to infer its identity and pose. (b) Image generation: We visualize prototypes with a pre-trained 3D-to-2D image renderer. (c) Few shot object labelling: Assigning a label to a prototype automatically transfers this label to its assigned instances.

of meshes or pointclouds from multi-view data [85, 92, 129] and depend heavily on a sufficient number of views, our model learns to infer the 3D object feature representation from a single view upon self-training.

We propose 3D quantized-Networks (3DQ-Nets), a model that can detect objects in 3D and that can iteratively establish accurate object correspondences without human labels or 3D annotations. 3DQ-Net first maps 2.5D images to 3D feature maps using Geometry-Aware Recurrent Networks (GRNNs) introduced in Chapter 2. We initialize its feature representations by pre-training on self-supervised view prediction task introduced in section 2.2.1. Since the inferred 3D visual feature map is view-invariant, an object can obtain the same representation when inferred from images captured under different camera distances and angles. 3DQ-Nets further improve the features through automated cross-scene correspondence mining. The step is critical for establishing more accurate correspondence between objects. 3DQ-Nets then cluster objects in a pose-aware manner into several clusters of similar-looking objects. We call the learned cluster centers *prototypes*, since they correspond to aggregates of object instances across 3D poses and scales. Given a scene, our model learns to parse the scene in terms of objects associated to prototype identities and their corresponding 3D poses (see Figure 6.1 (a)). The learned prototypes can be explicitly rotated, and can be rendered into images through a learned neural decoder (see Figure 6.1 (b)). We demonstrate the usefulness of our framework in few-shot learning: our model



can recognize and name objects from one or a few samples (see Figure 6.1 (c)). Once given a labelled instance, the model propagates the label to all the instances in the same cluster.

Whether the model can infer correct correspondence from the object-centric 3D feature representation depends on the quality of two key components: the learned visual features and the 3D object detector. 3DQ-Nets aim to iteratively optimize these two components. The weights of the encoder, decoder, 3D object detector, and prototypes are optimized using a mix of end-to-end backpropagation and expectation-maximization (EM) steps, and we show 3D object detection and prototype learning improve over time and help one another.

We empirically show that individual modules of our model benefit one another and are essential for learning to recognize objects and their 3D orientation without supervision: the 3D object detector benefits from 3D visual prototypes by discarding bounding boxes not matching to prototypes; learning better object detection results in more accurate inference of finding object correspondences; better inferred object correspondences result in better learning of visual feature representations; and better visual feature improves clustering by inferring accurate pose-equivariant alignment of objects to prototypes.

We test our model in diverse environments including photo-realistic simulators and real world videos captured by a Kinect camera. We empirically show our model can effectively learn to name new objects in a few-shot setting by propagating provided labels through the learned clusters. Our model outperforms by a large margin numerous baselines that do not infer a 3D feature space, rather, detect and cluster objects in a 2D feature space using CNN feature representations pretrained on ImageNet and finetuned with the few supplied labels, or do not mine cross-scene correspondences. We ablate each module of the proposed model and quantify its contribution in the performance of our full model.

The main contribution of this work is matching objects in a 3D-aware representation space inferred from images without any 3D annotations. The proposed model learns the representations by unsupervised view prediction and automated correspondence mining. With the learned representations, objects are clustered into 3D prototypes which form then the basis for recognition: prototype identity inference and 3D pose with respect to the prototype’s orientation. To the best of our knowledge, this is the first system that demonstrates that pose-aware 3D object recognition emerges without any 3D annotations in RGB-D images.

## 6.2 3D Quantized-Networks (3DQ-Nets)

We depict the architecture of our model in Fig. 6.2. Given a set of RGB-D images of a static scene and the corresponding camera poses for capturing these images, our model constructs a 3D scene feature representation using geometry-aware recurrent networks (GRNNs) introduced in Chapter 2. Our model detects objects in the inferred 3D scene representation (see Section 2.2.2 for details about the object detector) and matches the 3D object feature tensors against a set of 3D prototypes by searching over 3D rotations (Section 6.2.1). Concurrently, our model uses the detected 3D boxes to improve the 3D visual feature representation by iteratively inferring 3D part correspondences across objects detected in different scenes, and using metric learning to supervise the feature representation to reinforce the inferred correspondences (Section 6.2.2).

Our model iteratively optimizes over weights of the encoder, decoder, 3D detector module

and prototypes, and uses individual modules to bootstrap the learning of the others. We pretrain the weights of the encoder and decoder of GRNNs by view prediction. We detail each module in their respective section and present the learning of the model in Section 6.2.3.

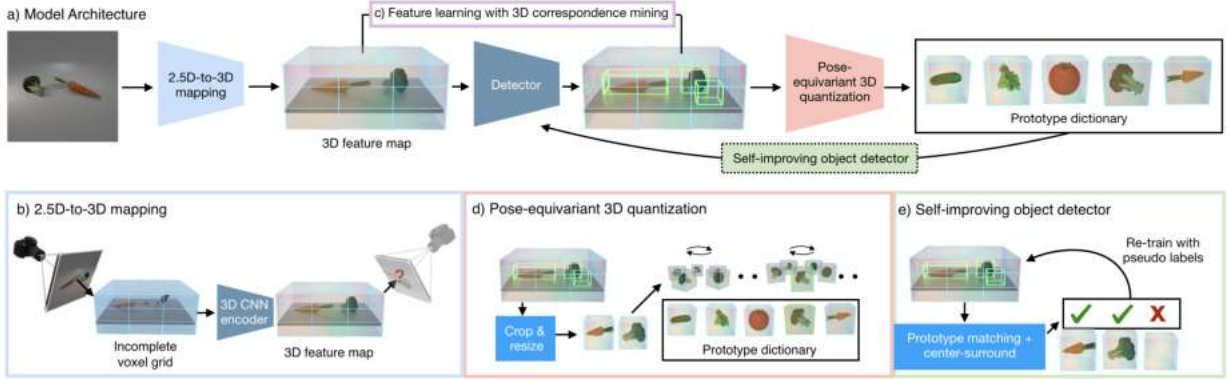


Figure 6.2: Architecture for **3D Quantized-Networks (3DQ-Nets)**. Given multi-view RGB-D images of scenes as input during training, our model learns to map a single RGB-D image to a completed scene 3D feature map at test time, by training for view prediction (b). The model additionally uses cross-scene and cross-object 3D correspondence mining and metric learning, to make the features more discriminative (c). Finally, using these learned features, our model quantizes object instances into a set of pose-canonical 3D prototypes using rotation-aware matching (d). These learned prototypes help improve our object detector by providing confident positive 3D object box labels (e).

**2.5D-to-3D mapping using Geometry-Aware Recurrent Networks (GRNNs)** Our model first use Geometry-aware Inverse Graphics Networks (GRNNs), introduced in Chapter 2 to “lift” RGB-D images of static world scenes to 3D scene feature maps. We will denote the 3D feature map inferred from an input RGB-D image  $I$  as  $\mathbf{M} = \text{GRNN}(I) \in \mathbb{R}^{w \times h \times d \times c}$ , where  $w, h, d, c$  denote the width, height, depth and number of channels, respectively. In this chapter, all our experiments use  $(w, h, d, c) = (72, 72, 72, 32)$ .

**3D feature learning by predicting views** We pre-train the encoder and decoder of GRNNs by predicting views using our multiview RGB-D image set with groundtruth camera poses, as introduced in Section 2.2.1. By training GRNNs to predict a query view given a single view input, we enforces the model to complete the missing or occluded information from the image. Different from the objective used in Section 2.2.1, we further include a occupancy prediction loss. Specifically, to predict a novel view, the scene feature map  $\mathbf{M}$  is oriented to a sampled query viewpoint  $v_q$  and decoded to an RGB image and occupancy grid, and then compared with the ground truth RGB ( $I_q$ ) and occupancy ( $O_q$ ) respectively:

$$\mathcal{L}^v = \|\text{Dec}^{\text{RGB}}(\mathbf{M}, v_q) - I_q\|_1 + \log(1 + \exp(-O_q \cdot \text{Dec}^{\text{occ}}(\mathbf{M}, v_q))), \quad (6.1)$$

The RGB output is trained with a regression loss, and the occupancy is trained with a logistic classification loss. Occupancy labels are computed through raycasting.

**3D object detection** A 3D detector operates on the output of the geometric encoder GRNN and predicts a variable number of object boxes with associated confidences:  $\mathcal{O} = \text{Det}(\mathbf{M}) \in \{(\hat{b}_{loc}^o, c^o) | \hat{b}_{loc}^o \in \mathbb{R}^6, c^o \in [0, 1]\}$ . See Section 2.2.2 for details of the object detector. We provide our detector with a “warm start” by pre-training it with 3D box annotations computed from triangulated 2D category-agnostic proposals from a publicly-available 2D objectness detector [142]. A detector trained with noisy annotations obtained from triangulation is expected to perform poorly, but it is sufficient for our system to start learning something useful. In Sec. 6.2.3 we describe how our model can self-training the detector for it to gradually learn and outperform its initialization.

### 6.2.1 Quantizing objects into prototypes

Our model learns a set of 3-dimensional prototypes  $\mathbf{e}_k \in \mathbb{R}^{w_p \times h_p \times d_p \times c}$ ,  $k \in \mathcal{K} = \{1, \dots, K\}$  by clustering object-centric 3D feature maps. Each prototype represents a set of similar objects. The prototype serves as the cluster center of the set. To learn them, our model clusters objects in the scene in a pose-equivariant and scale-equivariant manner: similar object instances that vary in scale and pose are mapped to the same prototype. We crop the 3D scene feature map  $\mathbf{M}$  given a detected box to obtain object 3D feature tensors, and resize it to match the common size of the 3D prototypes  $\mathbf{M}^o = \text{resize}(\text{crop}(\mathbf{M}, b^o), [w_p, h_p, d_p])$ . Our experiments use  $(w_p, h_p, d_p) = (16, 16, 16)$ . We match detected objects’ 3D feature tensors to prototypes using a rotation-aware feature matching. Specifically, we exhaustively search across rotations  $\mathcal{R}$ , in a parallel manner, considering increments of  $10^\circ$  along the vertical axis:

$$(z_{id}^o, z_R^o) = \arg \min_{k \in \mathcal{K}, R \in \mathcal{R}} \|\mathbf{e}_k - \text{Rot}(\mathbf{M}^o, R)\|_2, \forall o \in \mathcal{O}, \quad (6.2)$$

where  $\text{Rot}(\mathbf{M}, R)$  explicitly rotates the content in feature map  $\mathbf{M}$  with angle  $R$  through trilinear interpolation. Having assigned objects to oriented prototypes, we update our prototypes to minimize their Euclidean distance to the assigned oriented and scaled object tensors:

$$\mathcal{L}^{3DQ}(\mathbf{e}) = \sum_{o=1}^{|\mathcal{O}|} \|\mathbf{e}_{z_{id}^o} - \text{Rot}(\mathbf{M}^o, z_R^o)\|_2 \quad (6.3)$$

We initialize our prototype dictionary with a set of exemplars. To ensure prototype diversity at this initial stage, we build the dictionary incrementally, and only use an exemplar as a prototype if its feature distance to the already-initialized prototypes is higher than a threshold. Equations 6.2 and 6.3 can be seen as expectation maximization steps iterating between exemplars-to-prototypes assignment and prototype updates.

**Implementation details** Each object prototype is a 3D feature tensor of size  $16 \times 16 \times 16 \times 32$ . We initialize these prototypes incrementally and assign an exemplar as a prototype only if its feature distance to the already-initialized prototypes is lower than a cosine distance of 0.8. This ensures diversity of prototypes during initialization. While associating exemplars to a prototype, we check over 36 different rotations along the vertical axis at  $10^\circ$  increments. We keep our prototype dictionary size  $K$  as 50 for all the datasets. Empirically from Figure 3(a)(main paper) we have found that,  $K$  should be large enough to cover the object variability in the dataset.

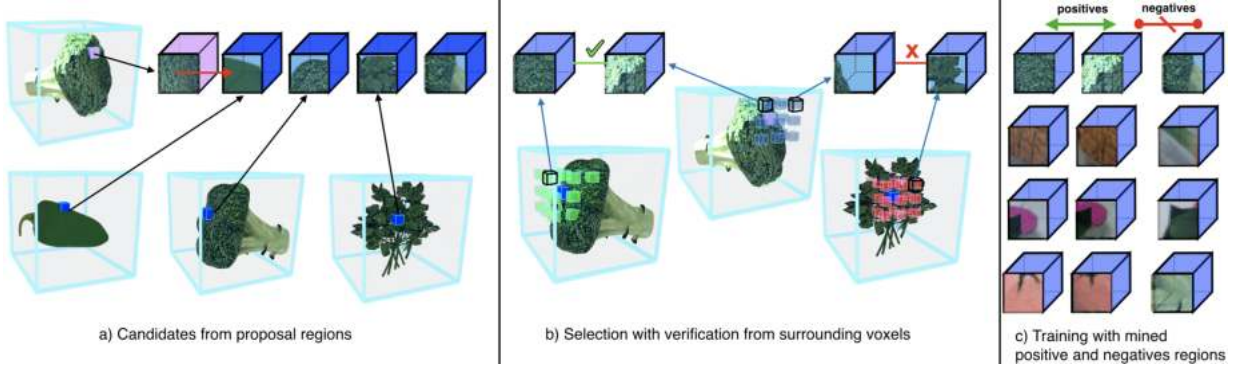


Figure 6.3: **Cross-scene 3D correspondence mining.** (a) We show that our approach relies on part-level correspondences obtained by matching the features of the query region (in pink) to a pool of object-centric 3D features maps. (b) These part-level correspondences are verified based on how well their surrounding voxels match with one another in a spatially consistent manner. (c) Finally we train our 2.5D-to-3D lifting module by doing metric learning using the verified positive regions and randomly sampled negatives.

### 6.2.2 Cross-scene 3D correspondence mining

Whether the model can establish the correct correspondence between objects and learn meaningful clusters relies on the quality of the visual features. To improve the visual features our model exploits visual similarity not only within scenes, but also across scenes. While the view prediction objective of Eq. 6.1 exploits different views of the *same* scene to learn the features, our model further exploits part-based correspondence between objects in *different* scenes to further improve the learned features. We adopt the correspondence mining method of ArtMiner [113] to operate in 3D as opposed to 2D: Part based correspondences are hypothesized within detected objects and are verified by voting of their surrounding context voxels. If the original match is verified, hard-positive matches are then suggested in the surrounding of the match. Using the mined hard positive matches and randomly sampled negatives, we finetune the weights of our encoder GRNN using metric learning. We empirically found that training with such cross-scene part-based correspondences helps improve the features.

**Implementation details** We randomly select 2000 object instances from our training data to create two pools (Query Pool & Target Pool) of size 1000 each. Each pool maintains object-centric 3D features of spatial size  $16 \times 16 \times 16$  extracted from the 3D feature map using the detected boxes. As shown in Figure 6.3, for each training iteration, we randomly select a  $2 \times 2 \times 2$  patch on an object sampled from the Query Pool, and by doing exhaustive search (across 36 different orientations along the vertical axis) and verification in the target pool object features we mine positive patches for metric learning training.

Searching over all the possible patches (we extract 4 patches from each object) for all 1000 objects in the target pool with all the 36 poses is computationally heavy. To reduce computation, we first complete a rough search at the object-level to retrieve objects which are similar to the query object, then we do fine-grained search at the part-level by searching over possible patches

from these objects of interest. We do this by ranking objects based on their cosine distance (we take the maximum cosine distance across 36 rotations) with the query object, and take only the top 30 objects to perform fine-grained search on the patch-level.

For each target object, we extract 4 patches to compare with the query patch. For each patch, we conduct a spatial consistency check similar to the work of [113]: instead of computing inner product between the patches, we compare the surrounding patches of these patches. We take the patches 6 unit Manhattan distance away from the patch center and compute an inner product on these surrounding patches. The summation of the inner product between all the surrounding patches serve as the final matching score for center patches. We take the top 200 patch retrievals based on the score, and take the 8 corners from their surroundings as positives. We create negatives by randomly selecting a pair of patches from the pool. However, training with naively sampled negatives on the fly is unstable. Following the suggestion from the work of [40], we maintain a dictionary of size 100,000 for the negatives examples, and do momentum update on our 2.5D-to-3D lifting module.

### 6.2.3 Iterative learning of object detection, visual features, and clustering

Since the initialized object detector is sub-optimal due to the lack of groundtruth 3D boxes and can affect the rest of the modules, it is critical that we have a mechanism to improve it over time. To achieve this, we iterate our model over the following steps: (i) 3D object detection (Section 6.2). This generates a set of 3D object proposals. (ii) Cross-scene object part correspondence mining and learning (Section 6.2.2). This updates the encoder weights GRNN using metric learning on inferred cross-scene correspondence on the detected objects. (iii) Prototype update (Section 6.2.1). This assigns detected object instances to prototypes and updates the prototypes  $e$  by backpropagating the clustering loss in Eq. 6.3. (iv) Object detector update. We label 3D object proposals as positives or negatives using a combination of 3D center-surround saliency score and matching to prototypes score. After the object detector is updated, we can iterate from step one to improve the rest of the modules.

Specifically, we keep the 3D object proposals that have a good matching score against the learned prototypes and discard the 3D object proposals whose 3D center-surround feature match score is below a threshold. The intuition is trust detection that either detects something that occurs often or has high saliency score. Center-surround saliency heuristic is used by numerous works for 2D and 3D object detection [54, 59]. We then train the 3D object detector module to emulate such labels through standard gradient descent. In Fig. 6.4-(a), we visualize the self annotations and improvement made by our self-improving detector over 4 iterations.

**Implementation details.** For every cropped object tensor, we calculate the cosine distance which is maximum amongst all the prototypes in the dictionary. If this calculated distance for a proposal is greater than 0.8 then we keep it as a valid proposal. We find the invalid proposals using 3D center-surround saliency. Specifically, we calculate the average cosine-distance of the cropped object tensor with its surrounding (top, down, left, right, front, behind) across all 3 axes. If the average cosine-distance is above 0.65 then we consider that proposal as invalid. We use the valid and invalid proposals as pseudo ground truths to further train the detector.

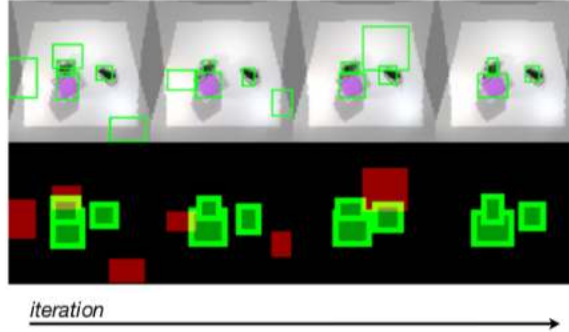


Figure 6.4: **Detection improvement over 4 iterations.** The first row shows the input image and the proposals of the object detector. The second row shows the annotations assigned to the proposals using the 3D prototype distance and 3D center-surround score. We show that our detector improves over time without any ground truth 3D proposals.

## 6.3 Experiments

We test our framework in a variety of simulated environments and real world scenes. In simulation, RGB, depth and egomotion are provided by the simulator, whereas in the real world, RGB and depth are provided by Kinect sensors and egomotion is computed using camera calibration. Our experiments aim to answer the following questions:

1. Do 3DQ-Nets recognize objects better than CNN models pretrained on large labelled image datasets?
2. How does the proposed pose-aware 3D clustering compare against 2.5D pose-aware clustering, 3D pose-unaware clustering, or raw 3D point cloud registration?
3. Does cross-scene 3D correspondence mining improves features over view-predictive training, and how much?
4. In 3DQ-Nets, do feature learning, object clustering to prototypes, and 3D object detection improve over training iterations?

We benchmark our model on three datasets: (i) **CLEVR veggie** dataset: we build upon the CLEVR dataset [53] and add 17 vegetable object models bought from Turbosquid. (ii) **CARLA** dataset: we created scenes using all 26 vehicle categories available in the CARLA simulator of Dosovitskiy *et al.* [18] (iii) **BigBIRD** [114]: a publicly available dataset that contains multiview shots for 125 different objects rotating on a table. We assign the objects to 41 different object categories, combining similar objects into a single category.

We further qualitatively evaluate our model on two datasets: (iv) **Replica** [121] dataset: we render images from the indoor meshes provided by Replica in AI Habitat simulator [110]. The views are selected by moving the agent around randomly selected objects. (v) **Real world desk scenes dataset**: training setup consists of 8 Kinect sensors surrounding the table to capture multiview RGB-D data. During test time, we only use a single Kinect sensor.

Datasets.	ResNetRet	ResNetClass	3DQ-Nets
CARLA	0.27	0.58	<b>0.71</b>
CLEVR	<b>0.80</b>	0.72	0.75
BigBIRD	0.40	0.67	<b>0.82</b>

Table 6.1: **Few shot object category labelling accuracy**

### 6.3.1 Few-shot object category labelling

In this experiment, we use ground-truth 3D bounding boxes during training of our model to isolate errors caused by the 3D object detection module. Our task is to classify object-centric image crops into object categories, when supplied with only two labelled object-image crop per category. This means, that e.g., in the CARLA dataset, we use 52 labelled object image crops. Note, the objects can be at any orientation. We evaluate the ability of our model and baselines to retrieve objects of the same category when supplied with these few labelled examples.

Given an annotated instance, our model finds the prototype that has minimum rotation-aware feature distance to the object instance, and it propagates the label to all the instances that are assigned to the same prototype. If a prototype is matched with more than one label, then the label which has matched the most is assigned to the prototype. Note that the small labelled set is not used to update our features or prototypes. In Table 7.1, we compare 3DQ-Nets against two 2D baseline models using pretrained ResNet-18 on ImageNet as their backbone: (i) Finetuning the top layer of ResNet-18 with our training examples (ResNetClass), (ii) using the top average pool layer activations of ResNet-18 to retrieve and copy the label of the nearest neighbor instance from the training examples (ResNetRet), i.e., not finetuning at all the weights. We show the results in Table 7.1. Our model outperforms both ResNetClass and ResNetRet. Despite the fact the ResNet features are pre-trained on a large set of annotated images, our model can self-adapt in the new domain of each dataset, and thus learn more meaningful object distances, captured in the inferred 3D feature representations. On CLEVR-veggie dataset, ResNetRet performs slightly better than 3DQ-Nets. We suspect this is because the object categories in CLEVR-veggie appear in ImageNet, so the ImageNet pertaining likely provides discriminative features for these objects.

### 6.3.2 Clustering with 3D pose-aware quantization

In this experiment, we evaluate the importance of 3D pose-aware quantization for 3D object clustering. We compare our model against three baselines: (i) 2.5DQ-Nets, a 2D CNN model that takes concatenated RGB and depth as input and quantizes detected 2D image patches into a discrete set of 2D prototypes by optimizing an autoencoding objective. During quantization, the model conducts 2D rotation search. (ii) no-rot-3DQ-Nets, a model similar to ours except that it assigns instances to 3D prototypes without rotation search. (iii) Pointcloud registration [81], a method that uses registered point clouds as prototypes and conducts 3D rotation aware search to infer the identity of the closest 3D pointcloud prototype and the 3D pose of the object instance with respect to the prototype. For our model and baselines we consider ground-truth 3D and 2D object boxes to isolate the error from different detectors.

To evaluate the unsupervised classification accuracy using prototypes, we use LIN-MATCH,



a bipartite graph matching method [63], that finds the permutation of prototype indices that minimizes the classification error. We show these comparisons with varying length of the prototype dictionary in Figure 6.5 (a). We see in Figure 6.5 (a) that models that use 3D representation achieve significantly higher accuracy compared to models using 2D representation. Further adding rotation search in 3D during clustering improves the performance since the operation enforces objects with similar appearance but with different poses to be clustered together. We also show that being able to inpaint objects from a single view during inference helps our model in outperforming the Pointcloud registration baseline that needs to handle incomplete input object-centric pointclouds. In Figure 6.5 (b) we show the scene reconstruction results of our models and the neural baselines after replacing the object in the scene with its best matched prototype under the inferred pose and rendering the 3D feature map through the learned decoder. We further include the unsupervised classification accuracy when testing on the the CLEVR and BigBIRD datasets in Table 6.2. For this experiment we set the number of prototypes (K) as 50.

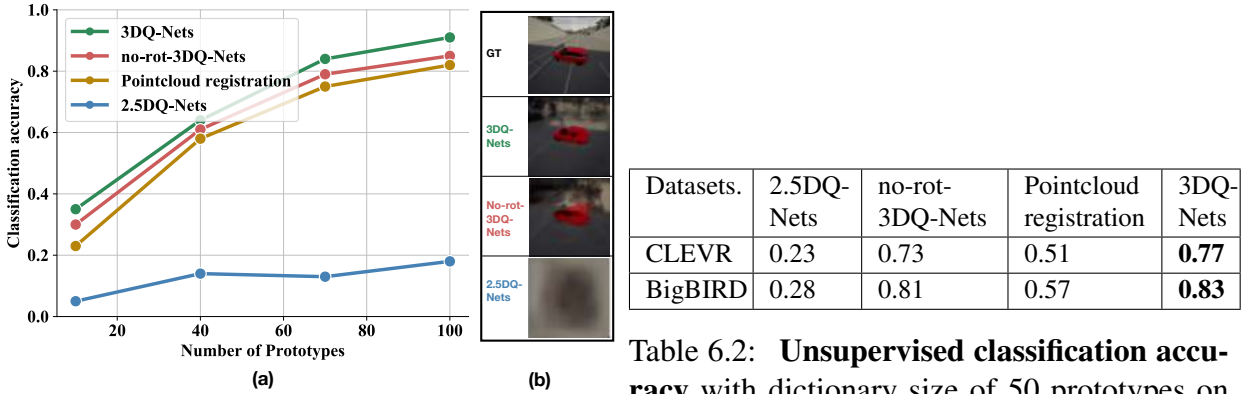


Figure 6.5: (a) **Unsupervised classification accuracy** with varying length of prototype dictionary in CARLA. (b) **Scene reconstruction** results using the learned prototypes from our model and the baselines.

Table 6.2: **Unsupervised classification accuracy** with dictionary size of 50 prototypes on CLEVR and BigBIRD datasets.

### 6.3.3 3D feature learning with 3D correspondence mining

In this experiment, we evaluate the contribution of 3D mining in feature learning, by evaluating the features in object category few shot retrieval. We compare it against the following feature learning methods: (i) Resnet-18 pretrained on Imagenet dataset (ResNet), where we average-pool features within the projected (ground-truth) 2D object boxes to represent the objects. (ii) GRNNs trained with RGB view and occupancy prediction (rgb-occ) of [135]. (iii) GRNNs trained with object-centric view contrastive prediction (rgbocc+VC) of [33]. (iv) We improve (iii) by using the same metric learning loss function [40] as our model (rgbocc+VC\*). (v) GRNNs trained additionally with cross-scene 3D mining (ours). For (ii),(iii),(iv),(v), we use the cropped 3D feature maps from 3D object boxes to represent the objects. We randomly sample 1000 objects and retrieve their nearest neighbors by considering the maximum inner product across 36 rotations against a pool of another 1000 objects. For (i), we consider 2D rotation search as opposed to 3D.



Datasets	ResNet	rgbocc	rgbocc+VC	rgbocc+VC*	ours
CARLA	0.49	0.62	0.55	0.67	<b>0.80</b>
CLEVR	<b>0.87</b>	0.74	0.71	0.74	0.81
BigBIRD	0.47	0.44	0.69	<b>0.77</b>	0.73

Table 6.3: **Retrieval results (precision@10 nearest neighbors)** for different architectures and objectives for 2D and 3D visual representation learning.

We show category-level retrieval precision within the first 10 retrieved nearest neighbors (i.e., precision@10) in Table 6.3.

As shown in Table 6.3, cross-scene correspondence mining improves the retrieval results. In the CLEVR dataset, ResNet outperforms our model. Our model performs the best among the unsupervised methods.

**Object Level Retrieval.** Figure 6.6 shows the qualitative results for object level retrieval. Here, we compare the object retrieval results on object-centric (cropped and resized) 3D features maps which are learned from the proposed method (rgbocc + 3D correspondence mining) and 2 other baselines: rgbocc and rgbocc+vcdict, which are detailed in Section 4.3(main paper). We show the results on 3 datasets: CARLA, BigBIRD, and CLEVR. For each query image, shown in the first column, we show the top 5 retrievals for the three methods mentioned above. The green box signifies that the retrieved image belongs to the same object category as the query, but is in a different viewpoint of the same scene. Blue box depicts retrieval of the same object category from a completely different scene. As can be seen, our method (rgbocc+3D mining) gives much more accurate retrievals (more number of blue and green boxes) compared to the other two baselines across all datasets. We show the object level retrieval results for this method on Replica dataset in Figure 6.7.

**Patch-Level Retrieval.** Figure 6.8 shows the 3D object patch retrieval results using the learnt 3D features from the proposed cross-scene 3D correspondence mining technique. We visualize the top 5 object part retrievals given a query object patch and a pool of target objects. For each query image, we first unproject it in the 3D space, detect objects in the scene, and randomly select a 3D patch on one of the objects. The first column for each dataset represents the query and the next 5 columns show the corresponding top 5 retrieved patches. For each query-prediction row pair, the first row shows the input RGB images and the second row shows bird’s eye view of the same RGB images unprojected in 3D space. The blue patches in the bird’s eye view visualizations (2nd row) show the 2D projection of the query/retrieved 3D patch. We additionally show patch based retrieval results on Replica dataset in Figure 6.9. We show the top 5 retrieved 3D patches that best matched the corresponding query patch using verification from surrounding voxels technique described in Figure 6.3 (b). As can be seen, patch based retrievals seem meaningful when surrounding context is given importance.

**Rotation Matching.** Finding the rotation transformation between two randomly posed RGB images is a crucial step for our model. As mentioned in Section 3.2(main paper), to do pose-

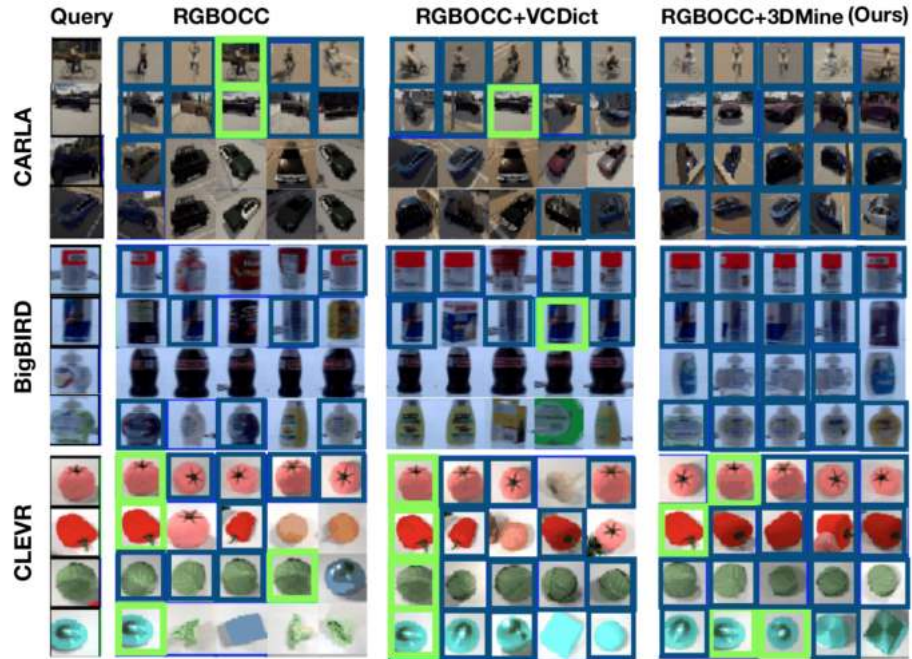


Figure 6.6: **3D object retrieval results** obtained by retrieving image patch using features learned from different feature learning methods, including rgbocc, rgbocc+vcdict, and rgbocc+3D correspondence mining (3DMine) methods. We visualize the retrieval results on CARLA, BigBIRD, and CLEVR datasets. The green boxes indicate that the retrieved image patches belongs to the same object instance as the query, but is in a different viewpoint. The blue boxes indicate instances with the same ground truth object category labels.

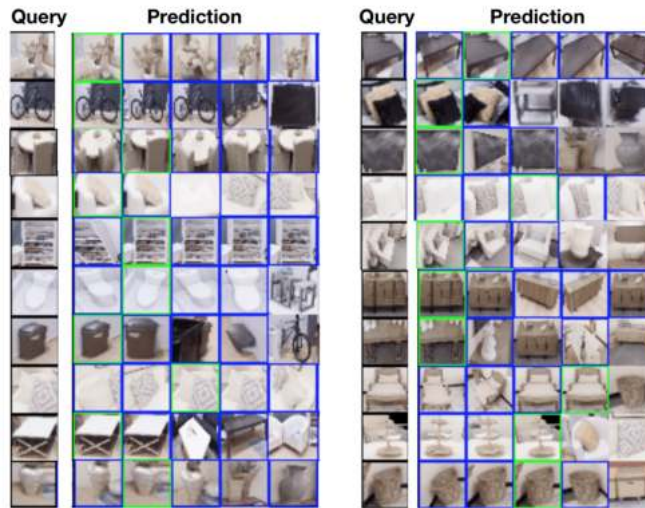


Figure 6.7: 3D object retrieval results obtained by rgbocc+3D correspondence mining on Replica dataset.

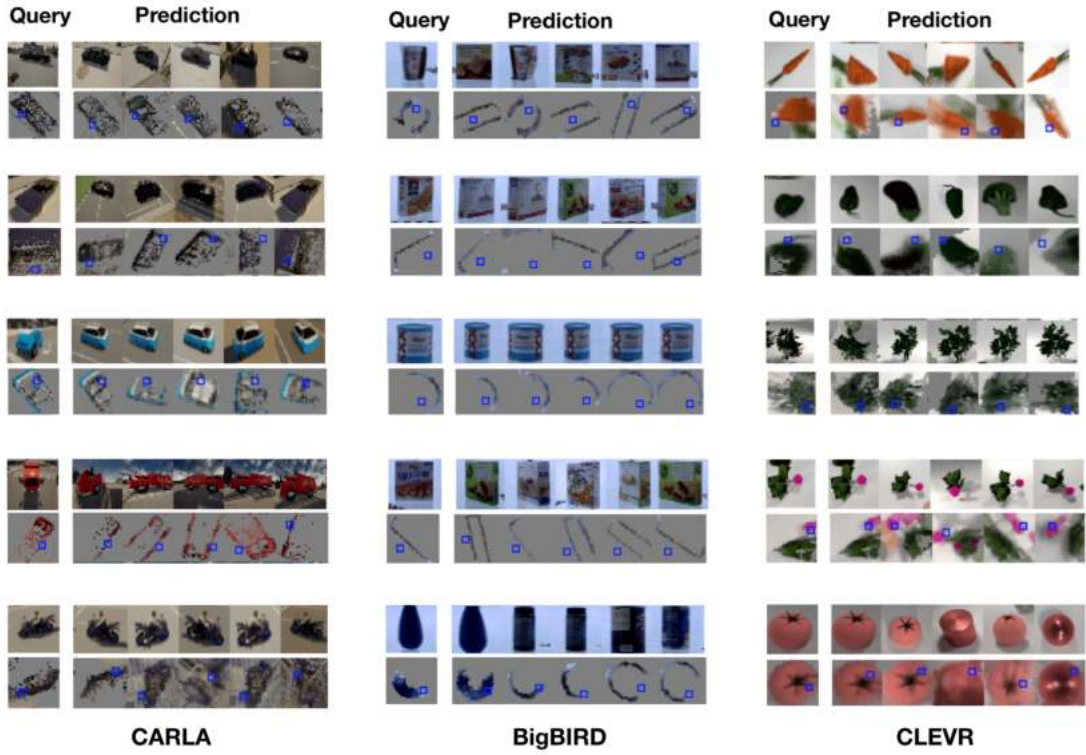


Figure 6.8: **Patch-Level retrieval results** on CARLA, BigBIRD, and CLEVR datasets. For each query-prediction row pair, the first row shows the input RGB images and the second row shows bird’s eye view projection of the RGB-D point cloud. The blue patches in the bird’s eye view visualizations (2nd row) show the 2D projection of the query/retrieved 3D patch.

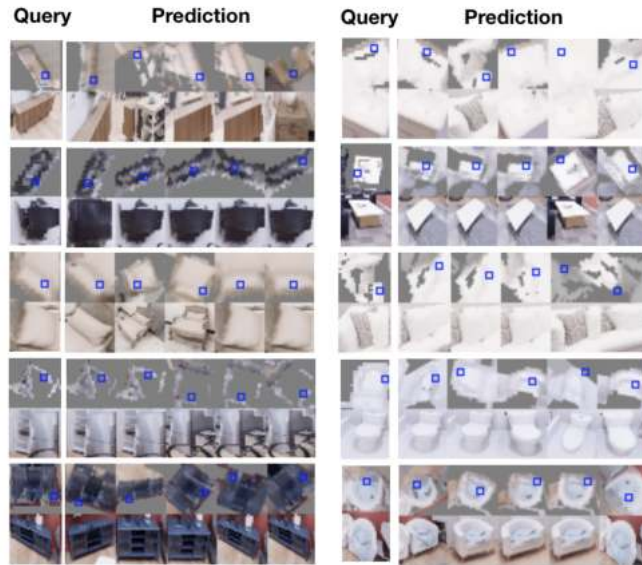


Figure 6.9: Patch based 3D object retrieval results on Replica dataset.



equivariant quantization, we need to first align the input object 3D feature tensors with an object prototype. The quality of our quantization relies on the quality of the features that will yield the correct rotation alignment. We show the qualitative performance of such rotation assignment on CARLA, BigBIRD and CLEVR datasets in Figure 6.10. For each of those  $3 \times 7$  grids, the first row shows the input RGB images of the same object category in different poses, the second row shows the bird’s eye view of the same RGBs unprojected in 3D space, and the third row shows the bird’s eye view of the same unprojected RGBs but warped to the pose that best matches with the object in the first. We conduct this matching on top of our 3D feature space by doing a rotation aware search. As shown in the visualizations, our model can warp the objects in different orientations to an orientation in the vicinity of the pose of the target object.

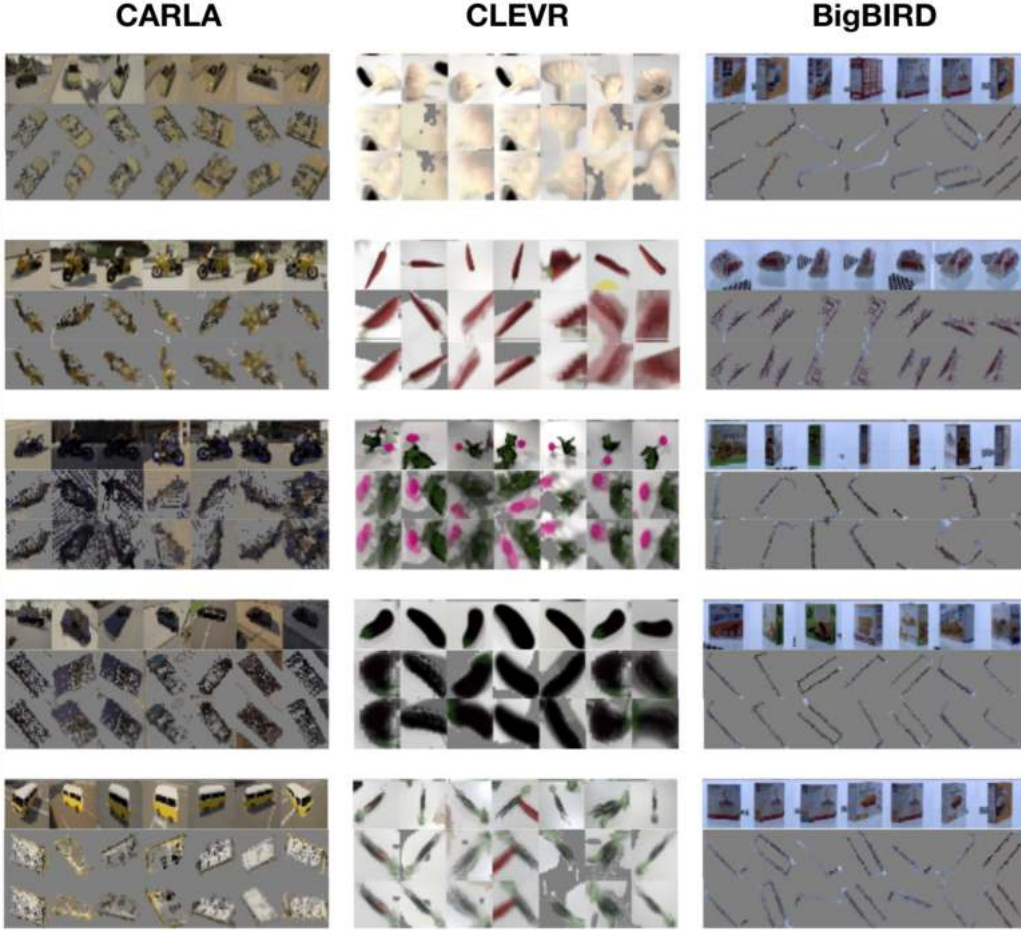


Figure 6.10: **Rotational alignment results** showing relative pose estimation between two randomly posed RGBs of the same object category. For each of the  $3 \times 7$  grids, the first row shows 7 input RGB images of the same object category in different poses. The second row shows the projection of the RGB-D point cloud in a birds eye view. The last row shows the projection of the same RGB-D point but warped to the pose that best matches with the object in the first. Results are shown on CARLA, BigBIRD and CLEVR datasets.

Task \ Iterations	Iter 0	Iter 1	Iter 2	Iter 3
Feature Learning	0.72	0.76	0.79	0.79
Quantization	0.51	0.63	0.65	0.66
Detection	0.43	0.48	0.51	0.52

Table 6.4: Performance across training EM iterations of our model in CLEVR. Feature learning is measured using the same technique as Table 6.3. Object quantization uses the same measurement technique as Fig. 6.5 (a). Detection performance is measured by meanAP at IoU = 0.5.

### 6.3.4 Joint training of 3D object detection, feature learning and clustering

Table 6.4 shows evaluations of our different modules during 4 iterations of EM. We see that the performance of all our modules improves over iterations. To initialize the weights for the modules (Iteration 0), we warm-start the 3D scene features using RGB view and occupancy prediction (rgbocc), and use the 3D object proposals provided by triangulated 2D boxes from 2D objectness detector to train the detector, visual features and prototypes. From Iteration 1 onwards, we use the 3D detected boxes from the trained detector as inputs, and use 3D mining to update the features. We subsequently improve the detector and the rest of the modules iteratively. We show that all modules can bootstrap one another and continually improve over iterations. We further show our detector improvement over time in Figure 6.4. More results are available on our project website: [https://shamitlal.github.io/project\\_pages/3DQ\\_Nets/3dq\\_nets.html](https://shamitlal.github.io/project_pages/3DQ_Nets/3dq_nets.html).

### 6.3.5 Scene parsing using prototypes

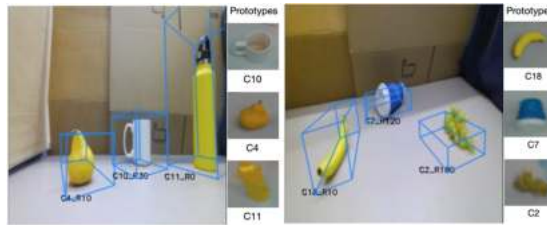


Figure 6.11: Real world scene parsing results.

Our learnt prototypes capture each object instance in its canonical pose. We use these prototypes for task of scene parsing. Given a new scene, we first detect all the objects and extract their features from the scene. Then, we match the object-centric feature maps with all the prototypes using a rotation aware similarity check explained in Section 6.2.1. For each detected object instance in the scene, we visualize the matched prototype number (C) and the respective rotation angle along vertical axis (R) as seen in Figure 6.11, Figure 6.12, Figure 6.13, and Figure 6.14. We also visualize the respective prototypes by neurally rendering them to images.







Figure 6.13: Scene parsing results for CLEVR dataset.



Figure 6.14: Scene parsing results for Replica dataset.



# Chapter 7

## Few-shot Concept learning and VQA with Disentangled 3D concepts

### 7.1 Introduction

In the previous chapter, we show how an agent can learn to correspond the same or similar objects in images without explicit annotations from humans. Since the agent knows to how to correspond objects that it has seen in the past, it becomes possible for the agent to recognition a new object from one or a few samples. In this chapter, we want to push the idea forwards: instead of learning to recognize the whole objects, can our model learns to recognize more general concepts, such as color, shapes, and textures?

Humans can learn diverse concepts from just one or a few samples. Consider the example in Figure 7.1. Assuming there is a person who has no prior knowledge about *blue* and *carrot*, by showing this person an image of a blue carrot and telling him “this is an *carrot* with *blue* color”, the person can easily generalize from this example to (1) recognizing *carrots* of varying colors, 3D poses and viewing conditions and under novel background scenes, (2) recognizing the color *blue* on different objects, (3) combine these two concepts with other concepts to form a novel object coloring he/she has never seen before, e.g., red carrot or blue tomato and (4) using the newly learned concepts to answer questions regarding the visual scene. Motivated by this, we explore computational models that can achieve these four types of generalization for visual concept learning.

We propose disentangling 3D prototypical networks (D3DP-Nets), models that learn to disentangle RGB-D images into objects, their 3D locations, sizes, 3D shapes and styles, and the background scene, as shown in Figure 7.2. Our model can learn to detect objects from a few 3D object bounding box annotations and can further disentangle objects into different attributes through a self-supervised view prediction task. Specifically, D3DP-Nets use Geometry-Aware Recurrent Networks (GRNNs), introduced in Chapter 2 to transform an input RGB-D (2.5D) image to a 3D scene feature map. From the scene feature map, our model learns to detect objects and disentangles each object into a 3D shape code and an 1D style code through a shape/style disentangling autoencoder. We use adaptive instance normalization layers [47] to encourage

<sup>0</sup>This chapter is based on the paper published previously on ICLR 2021 [93].

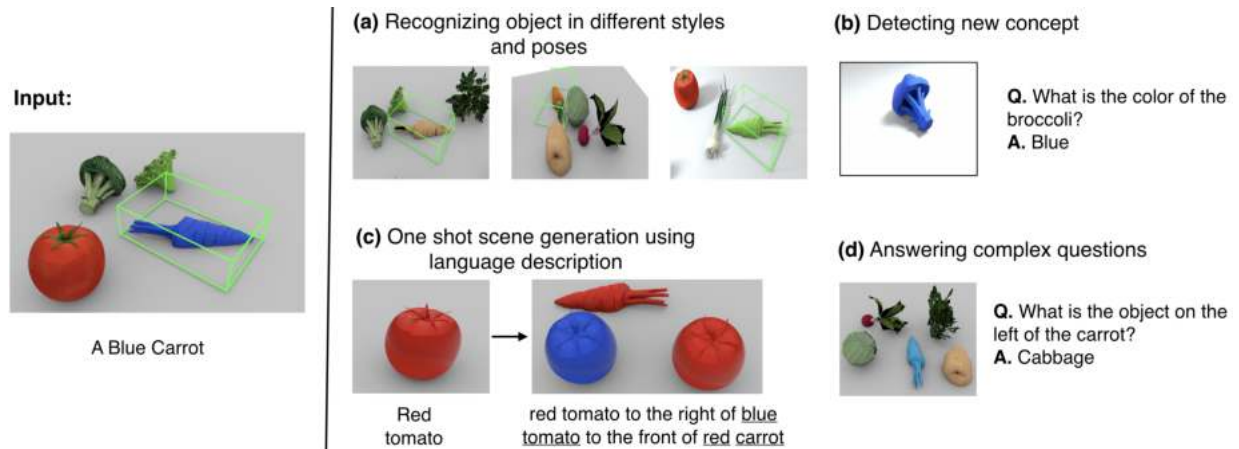


Figure 7.1: Given a single image-language example regarding new concepts (e.g., blue and carrot), our model can parse the object into its shape and style codes and ground them with *Blue* and *Carrot* labels, respectively. On the right, we show tasks the proposed model can achieve using this grounding. (a) It can detect the object under novel style, novel pose, and in novel scene arrangements and viewpoints. (b) It can detect a new concept like *blue broccoli*. (c) It can imagine scenes with the new concepts. (d) It can answer complex questions about the scene.

shape/style disentanglement within each object. Our key intuition is to represent objects and their shapes in terms of **3D feature representations disentangled from style variability** so that the model can correspond objects with similar shape by explicitly rotating and scaling their 3D shape representations during matching.

With the disentangled representations, D3DP-Nets can recognize new concepts regarding object shapes, styles and spatial arrangements from a few human-supplied labels by training concept classifiers only on the relevant feature subspace. Our model learns object shapes on shape codes, object colors and textures on style codes, and object spatial arrangements on object 3D locations. We show how the features relevant for each linguistic concept can be inferred from a few contrasting examples. These concept classifiers learn to attend to the discriminative property of the concept and ignore irrelevant visual features. By attending only to the relevant features, they can generalize with far fewer examples and can recognize novel attribute compositions not present in the training data.

We test D3DP-Nets in few-shot concept learning, visual question answering (VQA) and scene generation. We train concept classifiers for object shapes, object colors/materials, and spatial relationships on our inferred disentangled feature spaces, and show they outperform current state-of-the-art [45, 77], which use 2D representations. We show that a VQA modular network that incorporates our concept classifiers shows improved generalization over the state-of-the-art [77] with dramatically fewer examples. Last, we empirically show that D3DP-Nets generalize their view predictions to scenes with novel number, category and styles of objects, and compare against state-of-the-art view predictive architectures of [23].

The main contribution of this paper is to identify the importance of using disentangled 3D feature representations for few-shot concept learning. We show the disentangled 3D feature representations can be learned using self-supervised view prediction, and they are useful for de-

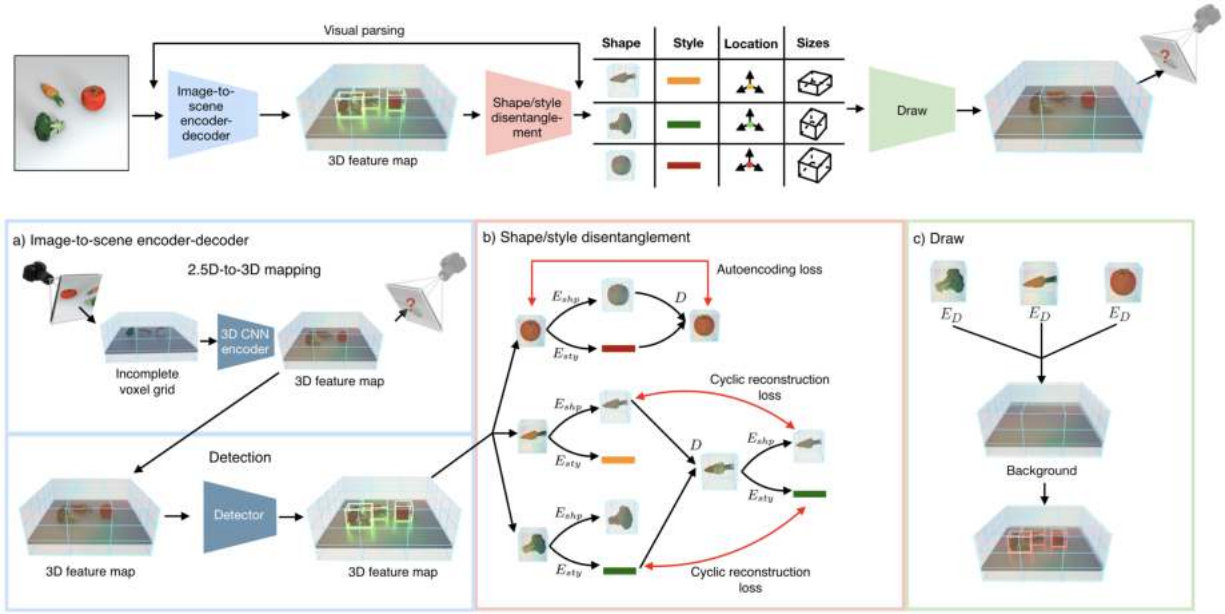


Figure 7.2: **Architecture for disentangling 3D prototypical networks (D3DP-Nets).** (a) Given multi-view posed RGB-D images of scenes as input during training, our model learns to map a single RGB-D image to a completed scene 3D feature map at test time, by training for view prediction. From the completed 3D scene feature map, our model learns to detect objects from the scene. (b) In each 3D object box, we apply a shape-style disentanglement autoencoder that disentangles the object-centric feature map to a 3D (feature) shape code and a 1D style code. (c) Our model can compose the disentangled representations to generate a novel scene 3D feature map. We urge the readers to refer the video in the supplementary material for an intuitive understanding of the architecture

testing and classifying language concepts by training them over the relevant only feature subsets. The proposed model outperforms the current state-of-the-art in VQA in the low data regime and the proposed 3D disentangled representation outperforms similar 2D or 2.5D ones in few-shot concept classification.

## 7.2 Disentangling 3D Prototypical Networks (D3DP-Nets)

The architecture of D3DP-Nets is illustrated in Figure 7.2. D3DP-Nets consists of two main components: (a) an image-to-scene encoder-decoder, and (b) an object shape/style disentanglement encoder-decoder. Next, we describe these components in detail.

**Image-to-scene encoder-decoder** D3DP-Nets use Geometry-Aware Recurrent Networks (GRNNs), introduced in Chapter 2, to map an input RGB-D image  $I$  to a 3D feature map  $\mathbf{M} = \text{GRNN}(I) \in \mathbb{R}^{w \times h \times d \times c}$ , of the scene, where  $w, h, d, c$  denote width, height, depth and number of channels, respectively. To initialize the weights of the encoder, we follow the same unsupervised view prediction and occupancy prediction loss used for pretraining 3DQ-Nets (Chapter 6 Equation 6.1).

Specifically, to predict a novel view and a complete occupancy map, the scene feature map  $\mathbf{M}$  is oriented to a sampled query viewpoint  $v_q$  and decoded to an RGB image and occupancy grid, and then compared with the ground truth RGB ( $I_q$ ) and occupancy ( $O_q$ ) respectively:

$$\mathcal{L}^v = \|\text{Dec}^{\text{RGB}}(\mathbf{M}, v_q) - I_q\|_1 + \log(1 + \exp(-O_q \cdot \text{Dec}^{\text{occ}}(\mathbf{M}, v_q))), \quad (7.1)$$

The RGB output is trained with a regression loss, and the occupancy is trained with a logistic classification loss. Occupancy labels are computed through raycasting. To disentangle attributes of an object, it is critical that the model can detect the object in the first place. To detect objects, we train a 3D object detector that takes as input the scene feature map  $\mathbf{M}$  and predicts 3D axis-aligned bounding boxes using groundtruth 3D bounding boxes. Details for the object detector are provide in Chapter 2 Section 2.2.2.

### 7.2.1 Object shape/style disentanglement

As the style of an image can be understood as a property which is shared across its spatial dimensions, previous works [48, 57] use adaptive instance normalization [47] as an inductive bias to disentangle style and contents in the images, and use the disentangled embeddings to do style transfer between a pair of images. D3DP-Nets uses this same inductive bias in its decoder to disentangle the style and 3D shape of an object. The 3D shape we consider here is not analogous to 3D occupancy. It is a blend of 3D occupancy and texture (spatial arrangement of color intensities).

Given a set of 3D object boxes  $\{b^o | o = 1 \dots |\mathcal{O}|\}$  where  $\mathcal{O}$  is the set of objects in the scene, D3DP-Nets obtain corresponding object feature maps  $\mathbf{M}^o = \text{crop}(\mathbf{M}, b^o)$  by cropping the scene feature map  $\mathbf{M}$  using the 3D bounding box coordinates  $b^o$ . We use ground-truth 3D boxes at training time and detected boxes at test time. Each object feature map is resized to a fixed resolution of  $16 \times 16 \times 16$ , and fed to an object-centric autoencoder whose encoding modules predict a 4D shape code  $z_{\text{shp}}^o = E_{\text{shp}}(\mathbf{M}^o) \in \mathbb{R}^{w \times h \times d \times c}$  and a 1D style code  $z_{\text{sty}}^o = E_{\text{sty}}(\mathbf{M}^o) \in \mathbb{R}^c$ . A decoder  $D$  composes the two using adaptive instance normalization (AIN) layers [47] by adjusting the mean and variance of the 4D shape code based on the 1D style code:  $\text{AIN}(z, \gamma, \beta) = \gamma \left( \frac{z - \mu(z)}{\sigma(z)} \right) + \beta$ , where  $z$  is obtained by a 3D convolution on  $z_{\text{shp}}$ ,  $\mu$  and  $\sigma$  are the channel-wise mean and standard deviation of  $z$ , and  $\beta$  and  $\gamma$  are extracted using single-layer perceptrons from  $z_{\text{sty}}$ . The object encoders and decoders are trained with an autoencoding objective and a cycle-consistency objective which ensure that the shape and style code remain consistent after composing, decoding and encoding again (see Figure 6.2 (b)):

$$\mathcal{L}^{\text{dis}} = \frac{1}{|\mathcal{O}|} \sum_{o=1}^{|\mathcal{O}|} \left( \underbrace{\|\mathbf{M}^o - D(E_{\text{shp}}(\mathbf{M}^o), E_{\text{sty}}(\mathbf{M}^o))\|_2}_{\text{autoencoding loss}} + \underbrace{\sum_{i \in \mathcal{O} \setminus o} \mathcal{L}^{c-\text{shp}}(\mathbf{M}^o, \mathbf{M}^i) + \mathcal{L}^{c-\text{sty}}(\mathbf{M}^o, \mathbf{M}^i)}_{\text{cycle-consistency loss}} \right), \quad (7.2)$$

where  $\mathcal{L}^{c-\text{shp}}(\mathbf{M}^o, \mathbf{M}^i) = \|E_{\text{shp}}(\mathbf{M}^o) - E_{\text{shp}}(D(E_{\text{shp}}(\mathbf{M}^o), E_{\text{sty}}(\mathbf{M}^i)))\|_2$  is the shape consistency loss and  $\mathcal{L}^{c-\text{sty}}(\mathbf{M}^o, \mathbf{M}^i) = \|E_{\text{sty}}(\mathbf{M}^o) - E_{\text{sty}}(D(E_{\text{shp}}(\mathbf{M}^i), E_{\text{sty}}(\mathbf{M}^o)))\|_2$  is the style consistency loss.

We further include a view prediction loss on the synthesized scene feature map  $\bar{\mathbf{M}}$ , which is composed by replacing each object feature map  $\mathbf{M}^o$  with its re-synthesized version  $D(z_{\text{shp}}^o, z_{\text{sty}}^o)$ , resized to the original object size, as shown in Figure 6.2(c). The view prediction reads:  $\mathcal{L}^{\text{view-pred-synth}} = \|\text{Dec}(\text{Rot}(\bar{\mathbf{M}}, v^{t+1})) - I_{t+1}\|_1$ . The total unsupervised optimization loss for D3DP-Nets reads:

$$\mathcal{L}^{\text{uns}} = \mathcal{L}^{\text{view-pred}} + \mathcal{L}^{\text{view-pred-synth}} + \mathcal{L}^{\text{dis}}. \quad (7.3)$$

### 7.2.2 3D disentangled prototype learning

Given a set of human annotations in the form of labels for object attributes (shape, color, material, size), our model computes prototypes for each concept (e.g. "red" or "sphere") in an attribute, using only the relevant feature embeddings. For example, object category prototypes are learned on top of shape codes, and material and color prototypes are learned on top of style codes. In order to classify a new object example, we compute the nearest neighbors between the inferred shape and style embeddings from the D3DP-Nets with the prototypes in the prototype dictionary, as shown in Figure 7.3. This non-parametric classification method allows us to detect objects even from a single example, and also improves when more labels are provided by co-training the underlying feature representation space as in [117].

To compute the distance between an embedding  $x$  and a prototype  $y$ , we define the following rotation-aware distance metric:

$$\langle x, y \rangle_R = \begin{cases} \langle x, y \rangle & \text{if } x, y \text{ are 1D} \\ \max_{r \in \mathcal{R}} \langle \text{Rot}(x, r), y \rangle & \text{if } x, y \text{ are 4D} \end{cases} \quad (7.4)$$

where  $\text{Rot}(x, r)$  explicitly rotates the content in 3D feature map  $x$  with angle  $r$  through trilinear interpolation. We exhaustively search across rotations  $\mathcal{R}$ , in a parallel manner, considering increments of  $10^\circ$  along the vertical axis. This is specifically shown in the bottom right of Figure 7.3 while computing the *Filter Shape* function.

Our model initializes the concept prototypes by averaging the feature codes of the labelled instances. We build color and material concept prototypes, e.g., *red* or *rubber*, by passing the style codes through a color fully connected module and a material fully connected module respectively, and then averaging the outputs. For object category prototypes, we use a rotation-aware averaging over the (4D) object shape embeddings, which are produced by a 3D convolutional neural module over shape codes. Specifically, we find the alignment  $r$  for each shape embedding that is used to calculate  $\langle z_0, z_i \rangle_R$ , and average over the aligned embeddings to create the prototype.

When annotations for concepts are provided, we can jointly finetune our prototypes and neural modules (as well as D3DP-Net weights) using a cross entropy loss, whose logits are inner products between neural embeddings and prototypes. Specifically, given  $P(o_a = c) = \frac{\exp(\langle f_a(z^o), p_c \rangle_R)}{\sum_{d \in \mathcal{C}_a} \exp(\langle f_a(z^o), p_d \rangle_R)}$  where  $\langle \cdot, \cdot \rangle_R$  represents the rotation-aware distance metric,  $f_a$  is the neural module for attribute  $a$ ,  $\mathcal{C}_a$  is the set of concepts for attribute  $a$ , and  $o_a$  is the value of attribute  $a$  for object  $o$ , and  $p_c$  is the prototype for concept  $c$ . The loss used to train prototypes is:

$$\mathcal{L}^{\text{prototype}} = -\frac{1}{|\mathcal{O}|} \sum_{o \in \mathcal{O}} \sum_{a \in \mathcal{A}} \sum_{c \in \mathcal{C}_a} \mathbb{1}_{o_a=c} \log P(o_a = c) + \mathbb{1}_{o_a \neq c} \log P(o_a \neq c) \quad (7.5)$$

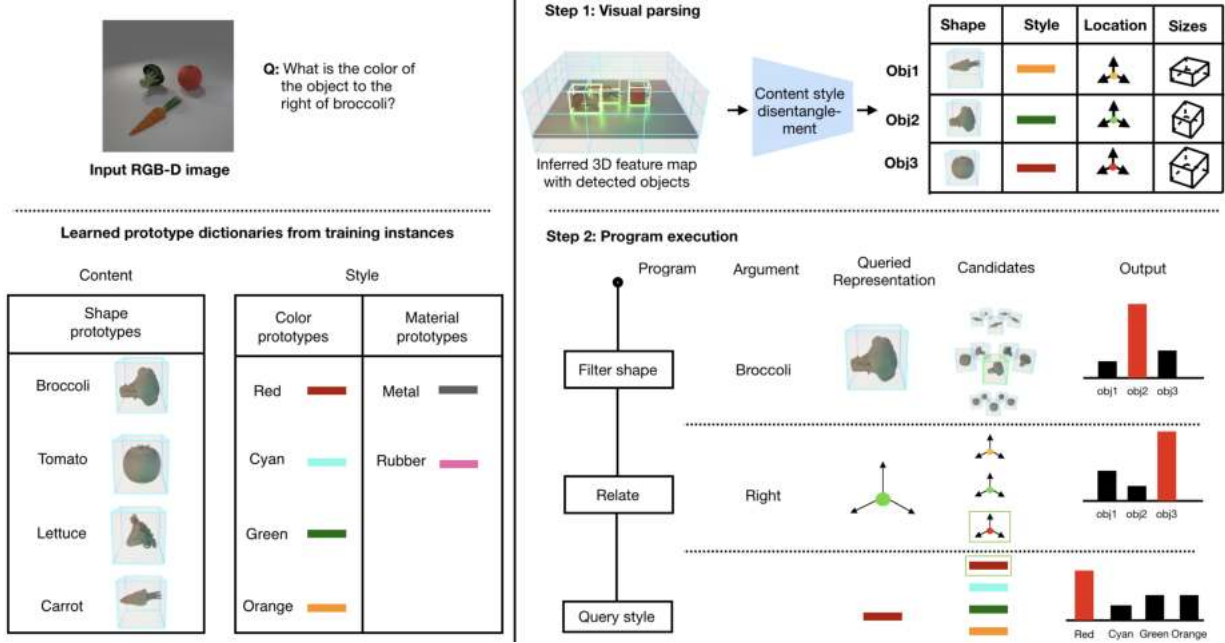


Figure 7.3: **D3DP-VQA Modular Networks.** Given a question-image pair and a list of learned prototype dictionaries (left), D3DP-Nets parse the visual scene to object shapes, styles, locations and sizes codes (top-right), while the semantic language parser converts the question to an executable program. The generated program is executed sequentially to answer the question (bottom-right). Note that in order to associate different poses of the same shape (Filter Shape), our model does a rotation-aware search between the indexed prototype and the candidate objects.

where  $\mathcal{A}$  is the set of attributes.

## 7.3 Experiments

We test D3DP-Nets in few-shot learning of object category, color and material, and compare against state-of-the-art 2D and 2.5D shape-style disentangled CNN representations (Sections 7.3.1). We integrate these concept classifiers in a visual question answering modular system (see Figure 7.3) and show it can answer questions about images more accurately than the state-of-the-art in the few-shot regime (Section 7.3.2). In addition, we test D3DP-Nets on novel 3D scene generation. Furthermore, we show our model can generate a 3D scene (and its 2D image renders) based on a language utterance description (Section 7.3.3).

### 7.3.1 Few-shot object shape and style category learning

We evaluate D3DP-Nets in its ability to classify shape and style concepts from few annotated examples on three datasets: i) CLEVR dataset [53]: it is comprised of cubes, spheres and cylinders of various sizes, colors and materials. We consider every unique combination of color and material categories as a single style category. The dataset has 16 style classes and 3 shape classes



Figure 7.4: **Replica dataset.** On the left, we show two objects in different scenes belonging to the same shape category ‘Plant’. On the right, we show two objects belonging to the same style category ‘Cream’.

	CLEVR				Real Veggie Data				Replica			
	5 shot		1 shot		5 shot		1 shot		5 shot		1 shot	
	Style	Shape	Style	Shape	Style	Shape	Style	Shape	Style	Shape	Style	Shape
D3DP-Net	<b>0.79</b>	0.86	<b>0.61</b>	0.70	<b>0.61</b>	0.52	<b>0.53</b>	0.44	<b>0.48</b>	0.58	<b>0.46</b>	<b>0.51</b>
3DP-Net	0.14	0.64	0.09	0.57	0.38	0.18	0.31	0.19	0.31	0.45	0.27	0.42
2D MUNIT	0.50	0.54	0.41	0.47	0.43	0.48	0.39	0.38	0.30	<b>0.60</b>	0.23	0.42
2.5D MUNIT	0.47	0.58	0.46	0.55	0.41	0.32	0.39	0.33	0.23	0.42	0.20	0.40
GQN	0.09	0.52	0.11	0.45	0.24	0.41	0.22	0.34	0.25	0.31	0.19	0.26
D3D-Net	0.43	0.48	0.26	0.40	0.31	0.28	0.18	0.24	0.23	0.29	0.10	0.14
MB(Supervised)	0.60	<b>0.89</b>	0.36	<b>0.75</b>	0.42	<b>0.71</b>	0.35	<b>0.67</b>	0.33	0.32	0.19	0.24

Table 7.1: Five & one shot classification accuracy for shape and style concepts in CLEVR [53], Real Veggie, and Replica datasets

in total. ii) Real Veggie dataset: it is a real-world scene dataset we collected that contains 800 RGB-D scenes of vegetables placed on a table surface. The dataset has 6 style classes and 7 shape classes in total. iii) Replica dataset [121]: it consists of 18 high quality reconstructions of indoor scenes. We use AI Habitat simulator [76] to render multiview RGB-D data for it. We use the 152 instance-level shape categories provided by Replica. Due to lack of style labels, we manually annotate 16 style categories. Figure 7.4 shows one example for both shape and style category.

We train D3DP-Nets self-supervisedly on posed multiview images in each dataset and learn the prototypes for each concept category. During training, we consider 1 and 5 labeled instances for each shape and style category present in the dataset. During testing, we consider a pool of 1000 object instances.

In this experiment, we use ground-truth bounding boxes to isolate errors caused by different object detection modules. We compare D3DP-Nets with 2D, 2.5D and 3D versions of Prototypical Networks [117] that similarly classify object image crops by comparing object feature embeddings to prototype embeddings. Specifically, we learn prototypical embeddings over the visual representations produced by the following baselines: (i) *2D MUNIT* [48] which disentangles shape and style within each object-centric 2D image RGB patch using the 2D equivalent



of the shape-style disentanglement architecture of our model, and learns using an autoencoding objective (ii) *2.5D MUNIT* an extension of 2D MUNIT which uses concatenated RGB and depth as input. (iii) *3DP-Nets*, a version of D3DP-Nets where object shape-style disentanglement is omitted, this version corresponds to the scene representation learning model of Tung et al. [135], introduced in Chapter 2. (iv) Generative Query Network *GQN* of Eslami et al. [22] which encodes multiview images of a scene and camera poses into a 2D feature map and is trained using cross-view prediction, similar to our model. (v) *D3D-Nets*, a version of D3DP-Nets where prototypical nearest neighbour retrieval is replaced with a linear layer which predicts the class probabilities. (iv) Meta-Baseline *MB* of Chen et al. [10] is the SOTA *supervised* few-shot learning model, pre-trained using ImageNet.

All baselines except *MB* are trained with the same unlabeled multiview image set as our method. All models classify each test image into a shape, and style category.

Few-shot concept classification results are shown in Table 7.1. D3DP-Nets outperforms all unsupervised baselines. Interestingly, D3DP-Nets give better classification accuracy on the 1-shot task than almost all of the unsupervised baselines on the 5-shot task. Figure 7.5 shows a visualization of the style codes produced by D3DP-Nets (left) and 2.5D MUNIT baseline (right) on 2000 randomly sampled object instances from CLEVR using t-SNE [72]. Each color represents a unique CLEVR style class. Indeed, in D3DP-Nets, codes of the same class are placed together, while for the 2.5D MUNIT baseline, this is not the case.

### 7.3.2 Few-shot visual question answering

We integrate concept detectors built on the D3DP-Nets representation into modular neural networks for visual question answering, in which a question about an image is mapped to a computational graph over a small number of reusable neural modules including object category detectors, style detectors and spatial expression detectors. Specifically, we build upon the recent Neuro-Symbolic Concept Learner (NSCL) [77], as shown in Figure 7.3. In NSCL, the input and output of different neural modules are probability distributions over 2D object proposals denoting the probability that the executed subprogram is referring to each object, and their object category, color and material classifiers also use nearest neighbors over learnt prototypes. For example, in the question “*How many yellow objects are there?*”, the model first uses the color classifier to predict for all objects the probability that they are yellow, and then uses the resulting probability map to give an answer. NSCL learns 1D prototypes for object shape, color and material categories and classifies objects to labels using nearest neighbors to these prototypes. In our D3DP-Nets-VQA architecture, we have 3D instead of 2D object proposals, and disentangled 3D shape and 1D color/material and spatial relationship prototypes instead.

We compare D3DP-VQA against the following models: i) *NSCL-2D* (with and without Ima-

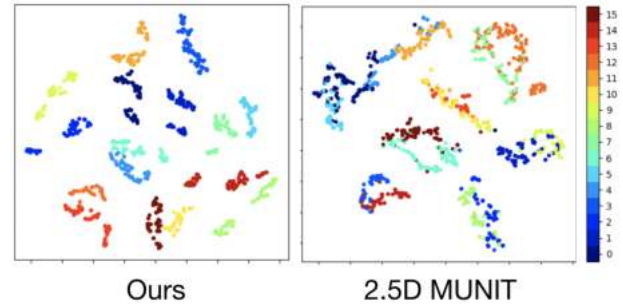


Figure 7.5: t-SNE visualization on styles codes.



VQA Model	In domain test set					One shot test set				
	Number of Training Examples					Number of Training Examples				
	10	25	50	100	250	10	25	50	100	250
Our full model	<b>0.809</b>	0.872	0.902	0.923	0.939	<b>0.775</b>	<b>0.836</b>	<b>0.834</b>	0.828	0.845
without 3D shape prototypes	0.798	0.858	0.538	0.905	0.932	0.410	0.410	0.517	0.745	0.771
without shape/style disentanglement	0.458	0.407	0.616	0.806	0.788	0.457	0.402	0.616	0.807	0.792
without 3D shape prototypes and without shape/style disentanglement	0.718	0.829	0.849	0.868	0.894	0.608	0.681	0.688	0.692	0.701
Entangled disentangled features	0.648	0.565	0.899	0.917	0.928	0.619	0.542	0.812	0.831	0.813
InstanceNorm disentangled features + rotation-aware check	0.606	0.831	0.875	0.894	0.905	0.627	0.775	0.832	<b>0.836</b>	<b>0.861</b>
2D NSCL [77]	0.733	<b>0.927</b>	<b>0.959</b>	<b>0.978</b>	<b>0.990</b>	0.594	0.708	0.703	0.789	0.743
2D NSCL [77] without ImageNet pretraining	0.514	0.624	0.682	0.844	0.931	0.467	0.502	0.553	0.624	0.679
2.5D NSCL [77]	0.594	0.737	0.828	0.881	0.925	0.528	0.633	0.651	0.633	0.633
2.5D NSCL [77] disentangled	0.436	0.486	0.640	0.735	0.842	0.430	0.462	0.517	0.561	0.564

Table 7.2: VQA results with model compared to ablations and 2D baselines in CLEVR [53] dataset.

geNet pretraining) the state of the art model of Mao et al. [77] that uses a ResNet-34 pretrained on ImageNet as input feature representations ii) *NSCL-2.5D*, in which the object visual representations for shape/color/material are computed over RGB and depth concatenated object patches as opposed to RGB alone. This model is pretrained with a view prediction loss using the CLEVR dataset in Sec. 7.3.1 iii) *NSCL-2.5D-disentangle* that uses disentangled object representations generated by our 2.5D MUNIT disentangling model, iv) *D3DP without 3D shape prototypes*, a version of D3DP-Nets that replaces the 3-dimensional shape codes with 1D ones obtained by spatial pooling v) *D3DP without disentanglement*, that learns prototypes for shape, color and material on top of entangled 3D tensors.

We consider the same supervision for our model and baselines in the form of densely annotated scenes with object attributes and 3D object boxes. We use ground-truth neural programs so as to not confound the results with the performance of a learned parser.

VQA performance results are shown in Table 7.2. We evaluate by varying the number of training scenes from 10 to 250. For each training scene we generate 10 questions. The original CLEVR dataset included 70,000 scenes and 700,000 questions, so even when training with 250 scenes, we are training with 0.35% the number of original scenes. Our full model outperforms all of the alternatives, showing the importance of both the 3D feature representations as well as disentanglement of shape and style. To test our model’s one shot generalization ability on questions about object categories it had not seen in the original training set, we introduce a new test set consisting of only novel objects. We generate a test set of 500 scenes in the CLEVR environment with three new objects: “cheese”, “garlic”, and “pepper” and introduce them to our model and baselines using one example image of each, associated with its shape category label. In Figure 7.6, we show example scene/question pairs for the in domain test set and one shot test set. The results described in Table 7.2 indicate that our model is able to maintain its ability to answer questions even when seeing completely novel objects and with very few training

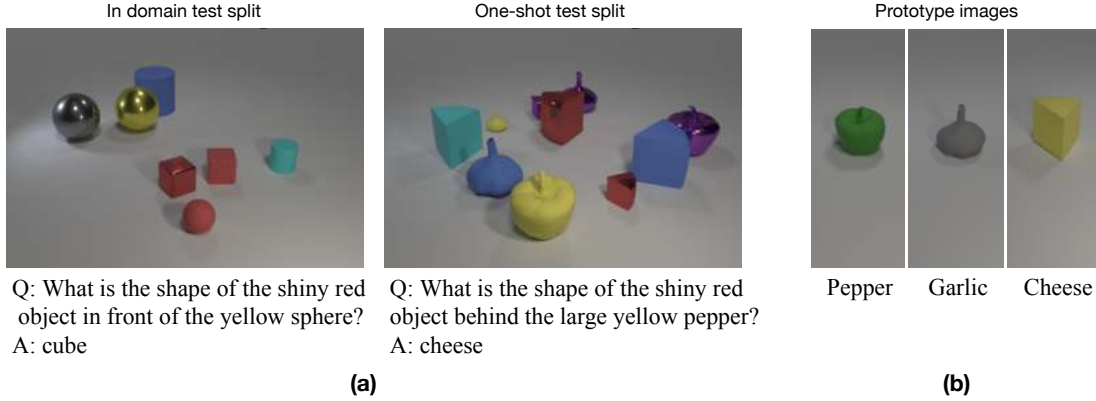


Figure 7.6: **(a)** The left scene/question pair is from the in domain test set, and the right scene/question pair is from the one shot test set. The colors, materials, sizes, and spatial relationships tested in both splits are the same. The only difference is that the one shot test set contains shapes the model did not see while training and was only exposed to one example before the testing phase. **(b)** The prototype images shown to the model before starting the one shot testing phase.

examples. The SOTA 2D model outperforms our model on the in domain test set because it is able to exploit pretraining on ImageNet, which our models are unable to do. However, our model is able to adapt much better than both the 2D and 2.5D baselines when operating at the extremely low data regime or the one shot generalization setting.

### 7.3.3 3D scene generation from language utterances

We test D3DP-Nets on the task of scene generation from language utterances. Given annotations for each prototype, our model can generate 3D scenes that comply with a language utterance, as seen in Figure 7.7 (b). We assume the parse tree of the utterance is given. Our model generates each object’s 3D feature map by combining shape and style prototypes as suggested by the utterance, and placing them iteratively on a background canvas making sure it complies with the spatial constraints in the utterance [94]. Our model can generate shape and style combinations not seen at training time. We neurally render the feature map to generate the RGB image.







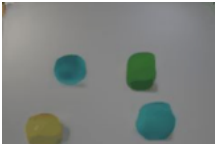









Natural Language Utterance	Neural Renders	
	View 1 (Ref view)	View 2
"Green cube to the left of brown sphere to the front of brown sphere to right front of blue sphere"		
"Blue cube to the left front of cyan cube to the left front of purple sphere to the front of purple cylinder to the right of green cube to the right of yellow cube to the behind of purple cube"		
"Gray cube to the left of blue sphere to the left of cyan cylinder to the top of green sphere to the top of purple sphere to the bottom right of blue cylinder"		
"Green cylinder to the right of cyan sphere to the top of yellow sphere to the left of cyan sphere"		
"Green sphere to the left of cyan cylinder to the top left of yellow cube"		
"Green sphere to the top of red sphere to the bottom of purple cube to the right of red cylinder"		
"Blue cube to the bottom of purple cube to the bottom of yellow sphere to the left of green cylinder to the top of cyan cube to the left of purple sphere to the bottom of purple cylinder"		
"Blue cylinder to the top left of cyan sphere to the top left of brown sphere to the bottom of red cylinder to the right of cyan sphere to the bottom right of brown sphere"		

Figure 7.7: Generating novel scenes using only a single example for each style and content class.



## **Part IV**

### **Language understanding: Learning to interpret language through visual simulation**



# Chapter 8

## Grounding Language in the Learned Visual simulator

### 8.1 Introduction

Consider the utterance “*the tomato is to the left of the pot*”. Humans can answer numerous questions about the situation described, as well as reason through alternatives, such as, “*is the pot larger than the tomato?*”, “*can we move to a viewpoint from which the tomato is completely hidden behind the pot?*”, “*can we have an object that is both to the left of the tomato and to the right of the pot?*”, and so on. How can we learn computational models that would permit a machine to carry out similar types of reasoning that humans are capable of? One possibility is to treat the task as text comprehension [17, 42, 55, 139] and train machine learning models using supervision from utterances accompanied with question / answer pairs. However, information needed for answering the questions is not contained in the utterance itself; training a model to carry out predictions in absence of the relevant information would lead to overfitting. Associating utterances with RGB images that depict the scene described in the utterance, and using both images and utterances for answering questions provides more world context and has been shown to be helpful. Consider though that information about object size, object extent, occlusion relationships, free space and so on, are only indirectly present in an RGB image, while they are readily available given a 3D representation of the scene the image depicts. Though it would take many training examples to learn whether a spoon can be placed in between the tomato and the pot on the table, in 3D such experiment can be mentally carried out easily, simply by considering whether the 3D model of the spoon can fit in the free space between the tomato and the pot. Humans are experts in inverting camera projection and inferring an approximate 3D scene given an RGB image [87]. This paper similarly builds upon inverse graphics neural architectures for providing the 3D visual representations to associate language, with the hope to inject spatial reasoning capabilities into architectures for language understanding.

**We propose associating language utterances to space-aware 3D visual feature representations of the scene they describe.** We infer such 3D scene representations from RGB images of the scene. Though inferring 3D scene representations from RGB images, a.k.a. inverse graphics

<sup>0</sup>This chapter is based on the paper published previously at CVPR 2020 [94].

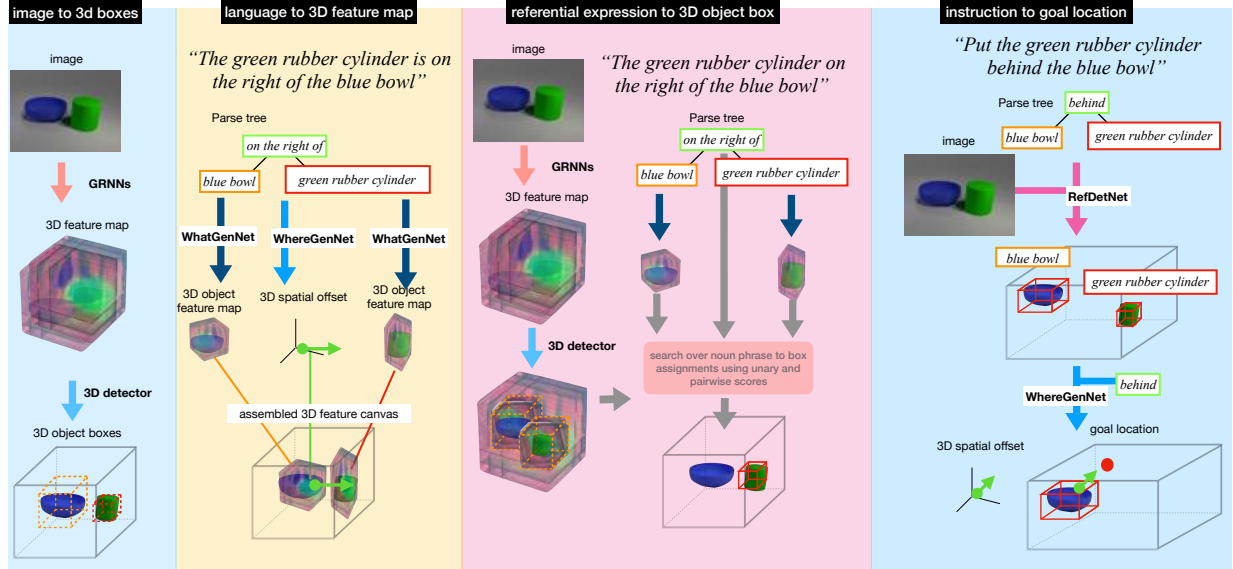


Figure 8.1: **Embodied language grounding with implicit 3D visual feature representations.** Our model associates utterances with 3D scene feature representations obtained from GRNNs. We map RGB images to 3D scene feature representations and 3D object boxes of the objects present. (column 1). We map an utterance and its dependency tree to object-centric 3D feature maps and cross-object relative 3D offsets using stochastic generative networks (column 2). We map a referential expression to the 3D box of the object referent (column 3). Last, given a placement instruction, we 3D localize the referents in the scene and infer the 3D desired location for the object to be manipulated (column 4). We use predicted location to supply rewards for trajectory optimization of placement policies.

is known to be a difficult problem [64, 103, 134], we build upon recent advances in computer vision [135] that consider inferring from images a learnable 3D scene feature representation in place of explicit 3D representations such as meshes, pointclouds or binary voxel occupancies pursued in previous inverse graphics research [64, 103, 134]. Such learnable 3D scene feature map emerges in a self-supervised manner by optimizing for view prediction in neural architectures with geometry-aware 3D representation bottlenecks, as described in previous work of Tung et al. [135]. After training, these architectures learn to map RGB video streams or single RGB images to complete 3D feature maps of the scene they depict, inpainting by imagination occluded or missing details of the 2D image input. **The contribution of our work is to use such 3D feature representations for language understanding and spatial reasoning.** We train modular generative networks that condition on the dependency tree of the utterance and predict a 3D feature map of the scene the utterance describes. They do so by predicting appearance and relative 3D location of objects, and updating a 3D feature workspace, as shown in Figure 8.1, 2nd column. We further train modular discriminative networks that condition on a referential expression and detect the object referred to in the 3D feature map of the input RGB image, by scoring object appearances and cross-object spatial arrangements, respectively, as shown in Figure 8.1, 3rd column. We call our model *embodied* since training the 2D image to 3D feature mapping



requires supervision from a mobile—or more generally embodied—agent that moves around in the 3D world, collects (posed) images and learns to predict visual results of its motion.

We demonstrate the benefits of associating language to 3D visual feature scene representations in three basic language understanding tasks:

**(1) Affordability reasoning** Our model can classify affordable (plausible) and unaffordable (implausible) spatial expressions. For example, “*A to the left of B, B to the left of C, C to the right of A*” describes a plausible configuration, while “*A to the left of B, B to the left of C, C to the left of A*” describes a non-plausible scene configuration, where A, B, C any object mentions. Our model reasons about plausibility of object arrangements in the inferred 3D feature map, where free space and object 3D intersection can easily be learnt/evaluated, as opposed to 2D image space.

**(2) Referential expression detection** Given a referential spatial expression, e.g., “*the blue sphere behind the yellow cube*”, and an RGB image, our model outputs the 3D object bounding box of the referent in the inferred 3D feature map, as shown in Figure 8.1 3rd column. Our 3D referential detector generalizes across camera viewpoints better than existing state-of-the-art 2D referential detectors [46] thanks to the view invariant 3D feature representation.

**(3) Instruction following** Given an object placement instruction, e.g., “*put the cube behind the book*”, our referential 3D object detector identifies the object to be manipulated and our generative network predicts its desired 3D goal location, as shown in Figure 8.1 4rth column. We use such 3D goal location in trajectory optimization of object placement policies. We show our model successfully executes natural language instructions.

In each task we compare against existing state-of-the-art models: the language-to-2D image generation model of [16] and the 2D referential object detector of [46], which we adapt to have same input as our model. Our model outperforms the baselines by a large margin in each of the three tasks. We further show strong generalization of natural language learnt concepts from the simulation to the real world thanks to the what-where decomposition employed in our generative and detection networks, where spatial expression detectors only use 3D spatial information, as opposed to object appearance and generalize to drastically different looking scenes without any further annotations. Our model’s improved performance is attributed to i) its improved generalization across camera placements thanks to the viewpoint invariant 3D feature representations, ii) its improved performance on free-space inference and plausible object placement in 3D over 2D. Many physical properties can be trivially evaluated in 3D while they need to be learned by a large number of training examples in 2D with questionable generalization across camera viewpoints. 3D object intersection is one such important property, very useful for spatial reasoning of plausible object arrangements. Our datasets and code will be made publicly available upon publication to facilitate reproducibility of our work.

## 8.2 Language grounding on 3D visual feature representations

We consider a dataset of 3D static scenes annotated with corresponding language descriptions and their dependency trees, as well as a reference camera viewpoint. We further assume access at training time to 3D object bounding boxes and correspondences between 3D object boxes and noun phrases in the language dependency trees. The language utterances we use describe

object spatial arrangements and are programmatically generated, same as their dependency trees, using the method described in [52]. We infer 3D feature maps of the world scenes from RGB images using Geometry-aware Recurrent Neural Nets (GRNNs) In Section 8.2.1, we describe our proposed generative networks that condition on the dependency tree of a language utterance and generate an object-factorized 3D feature map of the scene the utterance depicts. In Section 8.2.2, we describe discriminative networks that condition on the dependency tree of a language utterance and the inferred 3D feature map from the RGB image and localize in 3D the object being referred to. In Section 8.2.3, we show how our generative and discriminative networks of Sections 8.2.1,8.2.2 can be used for following object placement instructions.

### 8.2.1 Language-conditioned 3D visual imagination

We train generative networks to map language utterances to 3D feature maps of the scene they describe. They do so using a compositional generation process that conditions on the dependency tree of the utterance (assumed given) and generates one object at a time, predicting its appearance and location using two separate stochastic neural modules, *what* and *where*, as shown in Figure 8.2.

The *what* generation module  $G_A(p, z; \phi)$  is an stochastic generative network of object-centric appearance that given a noun phrase  $p$  learns to map the word embeddings of each adjective and noun and a random vector of sampled Gaussian noise  $z \in \mathbb{R}^{50} \sim \mathcal{N}(0, I)$  to a corresponding fixed size 3D feature tensor  $\hat{\mathbf{M}}^o \in \mathbb{R}^{w \times h \times d \times c}$  and a size vector  $s^o \in \mathbb{R}^3$  that describes the width, height, and depth for the tensor. We resize the 3D feature tensor  $\hat{\mathbf{M}}^o$  to have the predicted size  $s^o$  and obtain  $\mathbf{M}^o = \text{Resize}(\hat{\mathbf{M}}^o, s^o)$ . We use a gated mixture of experts [112] layer—a gated version of point-wise multiplication—to aggregate outputs from different adjectives and nouns, as shown in Figure 8.2.

The *where* generation module  $G_S(s, z, \psi)$  is a stochastic generative network of cross-object 3D offsets that learns to map the one-hot encoding of a spatial expression  $s$ , e.g., “*in front of*”, and a random vector of sampled Gaussian noise  $z \in \mathbb{R}^{50} \sim \mathcal{N}(0, I)$  to a relative 3D spatial offset  $d\mathbf{X}^{(i,j)} = (dX, dY, dZ) \in \mathbb{R}^3$  between the corresponding objects. Let  $b_i^o$  denote the 3D spatial coordinates of the corners of a generated object.

Our complete generative network conditions on the dependency parse tree of the utterance and adds one 3D object tensor  $\mathbf{M}_i^o, i = 1 \dots K$  at a time to a 3-dimensional feature canvas according to their predicted 3D locations, where  $K$  is the number of noun phrases in the dependency tree:  $\mathbf{M}^g = \sum_{i=1}^K \text{DRAW}(\mathbf{M}_i^o, \mathbf{X}_i^o)$ , where DRAW denotes the operation of adding a 3D feature tensor to a 3D location. The 3D location  $\mathbf{X}^1$  of the first object is chosen arbitrarily, and the locations of the rest of the object is based the predicted cross-object offsets:  $\mathbf{X}_2^o = \mathbf{X}_1^o + d\mathbf{X}^{(2,1)}$ . If two added objects intersect in 3D, i.e., the intersection over union of the 3D object bounding boxes is above a cross-validated threshold of 0.1,  $\text{IoU}(b_i^o, b_j^o) > 0.1$ , we re-sample object locations until we find a scene configuration where objects do not 3D intersect, or until we reach a maximum number of samples—in which case we infer the utterance is impossible to realize. By exploiting the constraint of non 3D intersection in the 3D feature space, our model can both generalize to longer parse trees than those seen at training time—by re-sampling until all spatial constraints are satisfied—as well as infer plausibility of utterances, as we validate empirically in Section 8.3.2. In 3D, non-physically plausible object intersection is easy to delineate from phys-

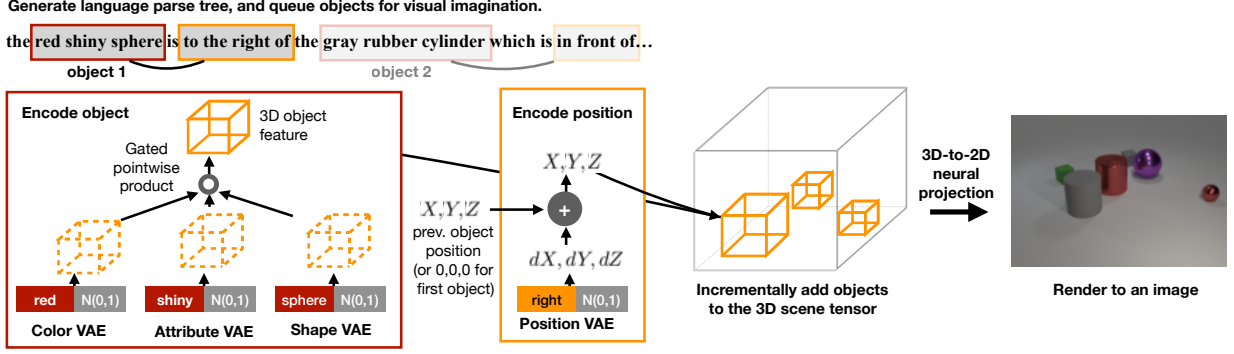


Figure 8.2: Mapping language utterances to object-centric appearance tensors and cross-object 3D spatial offsets using conditional *what-where* generative networks.

ically plausible object occlusion, something that is not easy to infer with 2D object coordinates, as we show empirically in Section 8.3.2, or at least you would need a much much larger number of *annotated* training examples to do so. Given the 3D coordinates of two 3D bounding boxes 3D intersection over union is easy to learn with a classifier, but we simply use thresholding of the computed 3D intersection over union.

We train our stochastic generative networks using conditional variational autoencoders.

### 8.2.2 Detecting referential expressions in 3D

We train discriminative networks to map spatial referential expressions, e.g., “the blue cube to the right of the yellow sphere behind the green cylinder” and related RGB images, to the 3D bounding box of the objects the expressions refer to. They do so using a compositional detection process that conditions on the dependency tree of the referential expression (assumed given) and predicts a 3D appearance detector template for each noun phrase, used to compute an object appearance score, and 3D spatial classifier for each spatial expression, used to compute a spatial compatibility score, as we detail below. Such compositional structure of our detector is necessary to handle referential expressions of arbitrary length. Our detector is comprised of a *what* detection module and a *where* detection module, as shown in Figure 8.3. The *what* module  $D_A(p; \xi)$  is a neural network that given a noun phrase  $p$  learns to map the word embeddings of each adjective and noun to a corresponding fixed-size 3D feature tensor  $f = D_A(p; \xi) \in \mathbb{R}^{w \times h \times d \times c}$ , we used  $w = h = d = 16$  and  $c = 32$ . Our *what* detection module is essentially a deterministic alternative of the *what* generative stochastic network of Section 8.2.1. The object appearance score is obtained by computing inner-product between the detection template  $D_A(p; \xi)$  and the cropped object 3D feature map  $\text{CropAndResize}(\mathbf{M}, b^o)$ , where  $\mathbf{M} = \text{GRNN}(I)$  and  $b^o$  the 3D box of the object. We feed the output of the inner product to a sigmoid activation layer.

The *where* detection module  $D_S(s, b_1^o, b_2^o; \omega)$  takes as input the 3D box coordinates of the hypothesized pair of objects under consideration, and the one-hot encoding of the spatial utterance  $s$  (e.g., “in front of”, “behind”), and predicts a score whether the two-object configuration matches the spatial expression.

We train both the *what* and *where* detection modules in a supervised way. During training,

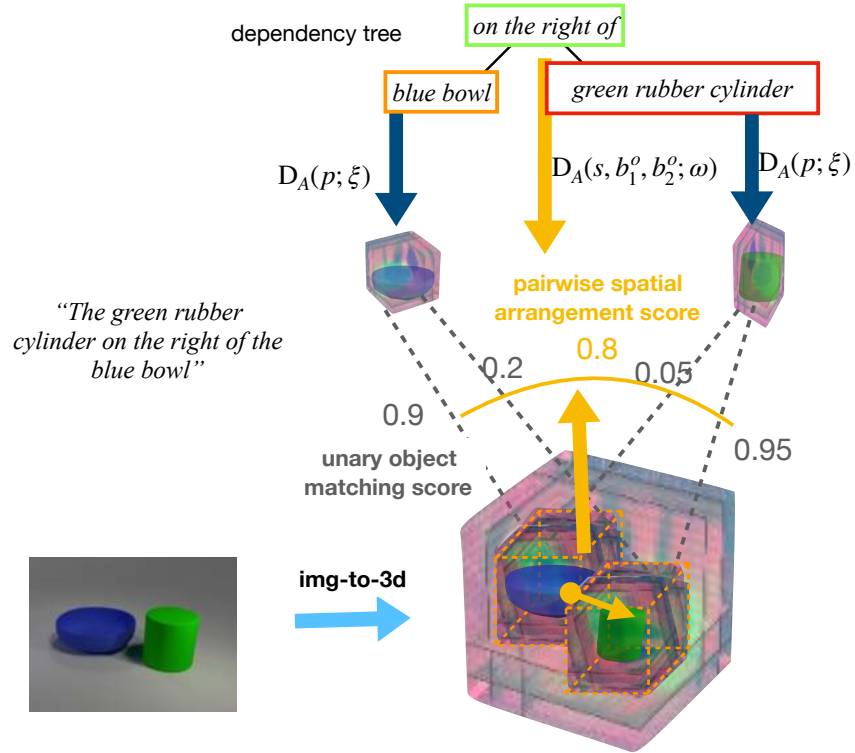


Figure 8.3: **3D referential object detection.** We exhaustively score all possible assignments of noun phrases to detected 3D bounding boxes. Each assignment is scored based on unary appearance scores and pairwise spatial scores, as described in the text.

we use ground-truth associations of noun phrases  $p$  to 3D object boxes in the image for positive examples, and random crops or other objects as negative examples. For cropping, we use ground-truth 3D object boxes at training time and detected 3D object box proposals from the 3D object detector at test time. We use positive examples from our training set and negative examples from competing expressions as well as synthetic 3D object boxes in random locations.

Having trained our *what* and *where* detector modules, and given the dependency parse tree of an utterance and a set of bottom up 3D object proposals, we exhaustively search over assignments of noun phrases to detected 3D objects in the scene. We only keep noun phrase to 3D box assignments if their unary matching score is above a cross-validated threshold of 0.4. Then, we simply pick the assignment of noun phrases to 3D boxes with the highest product of unary and pairwise scores. Our 3D referential detector resembles previous 2D referential detectors [13, 46], but operates in 3D appearance features and spatial arrangements, instead of 2D.

### 8.2.3 Instruction following

Humans use natural language to program fellow humans e.g., “*please, put the orange inside the wooden bowl*”. We would like to be able to program robotic agents in a similar manner. Most current policy learning methods use manually coded reward functions in simulation or instrumented environments to train policies, as opposed to visual detectors of natural language expressions [133]. If visual detectors of “*orange inside the wooden basket*” were available, we would use them to automatically monitor an agent’s progress towards achieving the desired goal and supply rewards accordingly, as opposed to hard-coding them in the environment.

We use the language-conditioned generative and detection models proposed in Section 8.2.1, 8.2.2 to obtain a reliable perceptual reward detector for object placement instructions with the following steps, as shown in Figure 8.1 4th column: **(1)** We localize in 3D all objects mentioned in the instruction using the aforementioned 3D referential detectors. **(2)** We predict the desired 3D goal location for the object to be manipulated  $\mathbf{x}_{goal}^o$  using our stochastic spatial arrangement generative network  $G_S(s, z; \psi)$ . **(3)** We compute per time step costs being proportional to the Euclidean distance of the current 3D location of the object  $\mathbf{x}_t^o$  and end-effector 3D location  $\mathbf{x}_t^e$  assumed known from forward dynamics, and the desired 3D goal object location  $\mathbf{x}_{goal}^o$  and end-effector 3D location  $\mathbf{x}_{goal}^e$ :  $\mathcal{C}_t = \|\mathbf{x}_t - \mathbf{x}_{goal}\|_2^2$ , where  $\mathbf{x}_t = [\mathbf{x}_t^o; \mathbf{x}_t^e]$  is the concatenation of object and end-effector state at time step  $t$  and  $\mathbf{x}_{goal} = [\mathbf{x}_{goal}^o; \mathbf{x}_{goal}^e]$ . We formulate this as a reinforcement learning problem, where at each time step the cost is given by  $c_t = \|\mathbf{x}_t - \mathbf{x}_{goal}\|_2$ . We use i-LQR (iterative Linear Quadratic Regulator) [127] to minimize the cost function  $\sum_{t=1}^T \mathcal{C}_t$ . I-LQR learns a time-dependent policy  $\pi_t(\mathbf{u}|\mathbf{x}; \theta) = \mathcal{N}(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t, \Sigma_t)$ , where the time-dependent control gains are learned by model-based updates, where the dynamical model  $p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{u}_t)$  of the *a priori* unknown dynamics is learned during training time. The actions  $\mathbf{u}$  are defined as the changes in the robot end-effector’s 3D position and orientation about the vertical axis, giving a 4-dimensional action space.

We show in Section 8.3.4 that our method successfully trains multiple language-conditioned policies. In comparison, 2D desired goal locations generated by 2D baselines [133] often fail to do so.

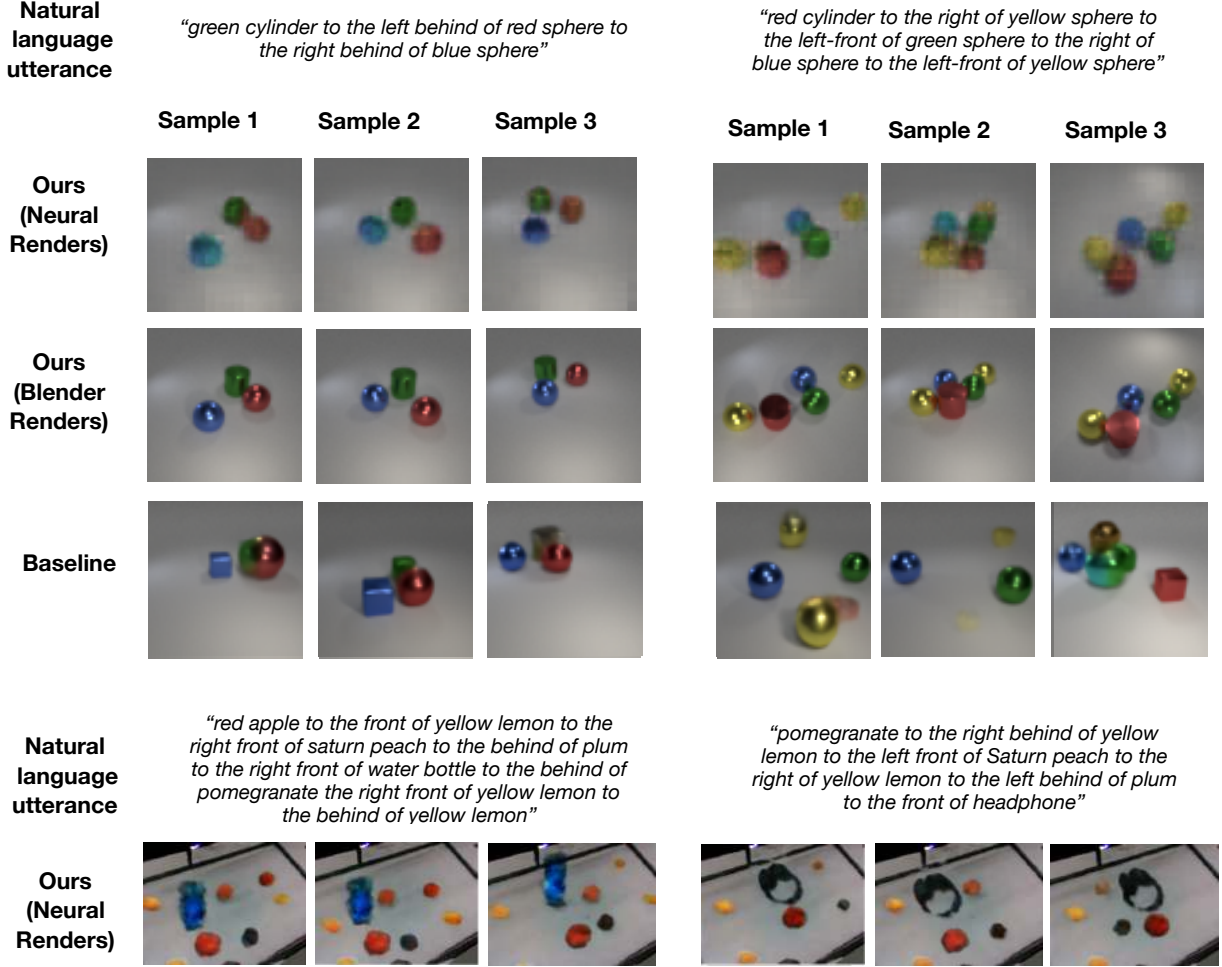


Figure 8.4: **Language to scene generation** (Rows 1,2,4) and **Language to image generation** (Row 3) from our model and the model of Deng et al [16] for utterances longer than those encountered at training time. Both our model and the baseline are stochastic, and we sample three generated scenes/images per utterance.

### 8.3 Experiments

We test the proposed language grounding model in the following tasks: **(i)** Generating scenes based on language utterances **(ii)** classifying utterances based on whether they describe possible or impossible scenes, **(iii)** detecting spatial referential expressions, and, **(iv)** following object placement instructions. We consider two datasets: (i) The CLEVR dataset of Johnson et al. [52] that contains 3D scenes annotated with natural language descriptions, their dependency parse trees, and the object 3D bounding boxes. The dataset contains Blender generated 3D scenes with geometric objects (Figure 8.1). Each object can take a number of colors, materials, shapes and sizes. Each scene is accompanied with a description of the object spatial arrangements, as well as its parse tree. Each scene is rendered from 12 azimuths and 4 elevation angles, namely,  $\{20^\circ, 40^\circ, 60^\circ, 120^\circ\}$ . We train GRNNs for view prediction using the RGB image views in the

training sets. The annotated 3D bounding boxes are used to train our 3D object detector. We generate 800 scenes for training, and 400 for testing. The language is generated randomly with a restriction to have **maximum 2 objects** for the training scenes. (ii) A veggie arrangement dataset we collected in the real world. We built a camera dome comprised of 8 cameras placed in a hemisphere above a table surface. We move vegetables around and collect multiview images. We automatically annotate the scene with 3D object boxes by doing 3D pointcloud subtraction at training time, we use the obtained 3D boxes to train our 3D object detector. At test time, objects are detected from a single view using our trained 3D detector. We further provide category labels for the vegetable present in single object scenes to facilitate association of labels to object 3D bounding boxes. More elaborate multiple instance learning techniques could be used to handle the general case of weakly annotated multi-object scenes [77]. We leave this for future work. We show extensive qualitative results in the veggie arrangement dataset as a proof that our model can effectively generalize to real world data if allowed multiview embodied supervision and weak category object labels.

### 8.3.1 Language conditioned scene generation

We show language-conditioned generated scenes for our model and the baseline model of Deng et al. [16] in Figure 8.4 for utterances longer than those encountered at training time. The model of Deng et al. [16] that generates a 2D RGB image directly (without an intermediate 3D representation) conditioned on a language utterance and its dependency tree. It predicts absolute 2D locations and 2D box sizes for objects and their 2D appearance feature maps, warped in predicted locations, and neurally decoded into an RGB image. We visualize our model’s predictions in two ways: i) **neural renders** are obtained by feeding the generated 3D assembled canvas to the 3D-to-2D neural projection module of GRNNs, ii) **Blender renders** are renderings of Blender scenes that contain object 3D meshes selected by small feature distance to the language generated object 3D feature tensors, and arranged based on the predicted 3D spatial offsets.

Our model re-samples an object location when they detect the 3D intersection-over-union (IoU) computed from the predicted 3D object boxes of the newly added object with existing ones is higher than a cross-validated threshold of 0.1. The model of Deng et al. [16] is trained to handle occluded objects. Notice though that it generates weird configurations as the number of objects increase. We tried imposing constraints of object placement using 2D IoU threshold in our baseline, but ran into the problem that we could not find plausible configurations for strict IoU thresholds, and we would obtain non-sensical configurations for low IoU thresholds. Note that 2D IoU cannot discriminate between physically plausible object occlusions and physically implausible object intersection. Reasoning about 3D object non intersection is indeed much easier in a 3D space.

### 8.3.2 Affordability inference of natural language utterances

We test our model and baselines in their ability to classify language utterances as describing sensible or non-sensible object configurations. We created a test set of 100 NL utterances, 50 of which are affordable, i.e., describe a realizable object arrangement, e.g., “*a red cube in front of a blue cylinder and in front of a red sphere, the blue cylinder is in front of the red sphere.*”, and

50 are unaffordable, i.e., describe a non-realistic object arrangement, e.g., “*a red cube is behind a cyan sphere and in front of a red cylinder, the cyan sphere is left behind the red cylinder*”. In each utterance, an object is mentioned multiple times. The utterance is unaffordable when these mentions are contradictory. Answering correctly requires spatial reasoning over possible object configurations. **Both our model and baselines have been trained only on plausible utterances and scenes. We use our dataset only for evaluation.** This setup is similar to violation of expectation [102]: the model detects violations while it has only been trained on plausible versions of the world.

Our model infers affordability of a language utterance by generating the 3D feature map of the described scene, as detailed in Section 8.2.1. When an object is mentioned multiple times in an utterance, our model uses the first mention to add it in the 3D feature canvas, and uses that pairwise object spatial classifier  $D_S$  of Section 8.2.2 to infer if the predicted configuration also satisfies the later constraints. If not, our model re-samples object arrangements until a configuration is found or a maximum number of samples is reached.

We compare our model against a baseline based on the model of Deng et al. [16]. Similar to our model, when an object is mentioned multiple times, we use the first mention to add it in the 2D image canvas, and use pairwise object spatial classifiers we train over 2D bounding box spatial information—as opposed to 3D—to infer if the predicted configuration also satisfies the later constraints. Note that there are no previous works that attempt this language affordability inference task, and our baseline essentially performs similar operations as our model but in a 2D image generation space.

We consider a sentence to be affordable if the spatial classifier predicts a score above 0.5 for the later constraint. **Our model achieved an affordability classification accuracy of 95% while the baseline 79.3%.** The result suggests an improved ability to reason about affordable and non affordable object configurations in 3D as opposed to 2D image space.

### 8.3.3 Detecting spatial referential expressions

We use the same dataset and train/test split of scenes as in the previous section. For each annotated scene, we consider the first mentioned object as the one being referred to, that needs to be detected. In this task, we compare our model with a variant of the modular 2D referential object detector of Hu et al. [46] that also takes as input the dependency parse tree of the expression. We train the object appearance detector for the baseline same as for our model using positive and negative examples but the inner product is on 2D feature space as opposed to 3D. We also train a pairwise spatial expression classifier to map width, height and x,y coordinates of the two 2D bounding boxes and the one-hot encoding of the spatial expression, e.g., “*in front of*” to a score reflecting whether the two boxes respect the corresponding arrangement. Note that our pairwise spatial expression classifier use 3D box information instead which helps it to generalize across camera placements.

Our referential detectors are upper bounded by the performance of the Region Proposal Networks (RPNs) in 3D for our model and in 2D for the baseline, since we compare language-generated object feature tensors to object features extracted from 2D and 3D bounding box proposals. We compare RPN performance in Table 8.1. An object is successfully detected when the predicted box has an intersection over union (IoU) at least 0.5 with the groundtruth bounding



mAP	ours RGB-D	[100] RGB-D	ours RGB	[100] RGB
2D	<b>0.993</b>	0.903	0.990	0.925
3D	<b>0.973</b>	-	0.969	-

Table 8.1: **Mean average precision for category agnostic region proposals.** Our 3D RPN outperforms the 2D state-of-the-art RPN of Faster R-CNN [100].

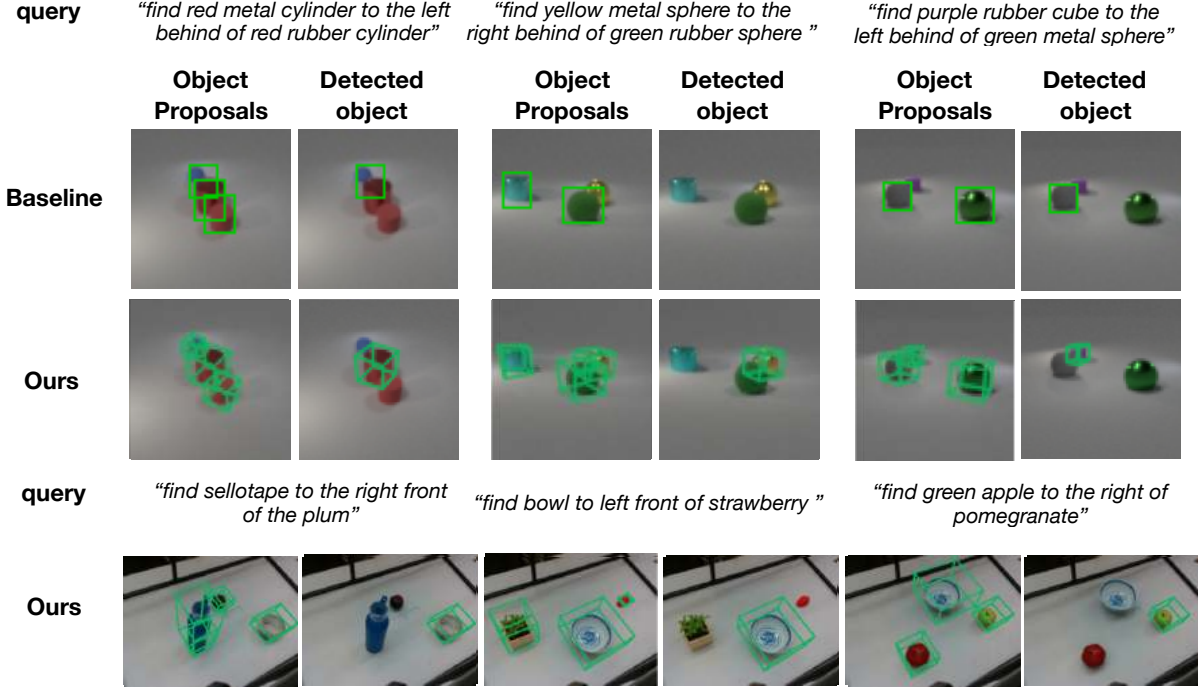


Figure 8.5: **Detecting referential spatial expressions.** Given a scene and a referential expression, our model localizes the object being referred to in 3D, while our baseline in 2D.

box. For our model, we project the detected 3D boxes to 2D and compute 2D mean average precision (mean AP). Both our model and the baseline use a single RGB image as input as well as a corresponding depth map, which our model uses during the 2D-to-3D unprojection operation and the 2D RPN concatenates with the RGB input image. Our 3D RPN that takes the GRNN map  $\mathbf{M}$  as input better delineates the objects under heavy occlusions than the 2D RPN of faster-RCNN [100].

We show quantitative results for referential expression detection in Table 8.2 with groundtruth as well as RPN predicted boxes, and qualitative results in Figure 8.5. In the “*in-domain view*” scenario, we test on camera viewpoints that have been seen at training time, in the “*out-domain view*” scenario, we test on **novel camera viewpoints**. An object is detected successfully when the corresponding detected bounding box has an IoU of 0.5 with the groundtruth box (in 3D for our model and in 2D for the baseline). Our model greatly outperforms the baseline for two reasons: a) it better detects objects in the scene despite heavy occlusions, and, b) even with groundtruth boxes, because the 3D representations of our model do not suffer from projection artifacts, they better generalize across camera viewpoints and object arrangements.

	Ours	[46]	Ours - GT 3D boxes	[46] - GT 2D boxes
<i>in-domain view</i>	0.87	0.70	<b>0.91</b>	0.79
<i>out-domain view</i>	0.79	0.25	<b>0.88</b>	0.64

Table 8.2: **F1-Score for detecting referential expressions.** Our model greatly outperforms the baseline with both groundtruth and predicted region proposals, especially for novel camera views.

### 8.3.4 Manipulation instruction following

We use the PyBullet Physics simulator [1] with similar setup as our CLEVR scenes. We use a simulated KUKA robot arm as our robotic platform. We use a *cube* and a *bowl*, using the same starting configuration for each scene, where the cube is held by the robot right above the bowl. We fix the end-effector to always point downwards, and we assume the object to be in robot’s hand at the beginning of each episode.

We compare our model against the 2D generative baseline of [16] that generates object locations in 2D, and thus supply costs of the form:  $\mathcal{C}^{2D}(\mathbf{x}_t) = \|\mathbf{x}_t^{2D} - \mathbf{x}_{goal}^{2D}\|_2^2$ . We show in Table 8.3 success rates for different spatial expressions, where we define success as placing the object in the set of locations implied by the instruction. Goal locations provided in 2D do much worse in guiding policy search than target object locations in 3D supplied by our model. This is because 2D distances suffer from foreshortening and reflect planning distance worse than 3D ones. This is not surprising: in fact, the robotics control literature almost always considers desired locations of objects to be achieved to be in 3D [66, 68]. In our work, we link language instructions with such 3D inference using inverse graphics computer vision architectures for 2D to 3D lifting in an implicit learnable 3D feature space.

Language Exp.	left	left-behind	left-front	right	right-behind	right-front	inside
Baseline	4/5	1/5	3/5	0/5	2/5	0/5	1/5
Ours	<b>5/5</b>	<b>3/5</b>	<b>5/5</b>	<b>5/5</b>	<b>5/5</b>	<b>3/5</b>	<b>5/5</b>

Table 8.3: **Success rates for executing instructions regarding object placement.** Policies learnt using costs over 3D configurations much outperform those learnt with costs over 2D configurations.

# Chapter 9

## Conclusion and Future Directions

### Overview

In this thesis, we have introduced key modules that would enable embodied agents to work and to learn from the visual data collected from their own body motion. Three key questions we try to answer in the beginning of the thesis are: (1) How should we parse the visual data into structured representations of the physical scene? What should be the representations? (2) How can we use the scene representations for the agents to see, act, reason, and learn language? (3) How can the learning of other modules improve the agent’s visual perception?

We answered the first question by showing that representing scenes as egomotion-stabilized 3D feature representations has several advantages over non-stabilized representations since it can better capture the bias of object permanence. To infer the representations from image observations, we proposed neural architecture and objective function that will allow the model to obtain the representations by learning from self-collected data through moving around. Next, we moved on to answer the second question by showing the learned representations are suitable for learning many downstream tasks regarding seeing, acting, reasoning, and learning language and concepts. Lastly, We answered the third question by demonstrating how the agents can learn and improve their object detection and visual representations by jointly training with these downstream tasks.

The work present in the thesis serves as an initial step towards building embodied agents that can extract information and learn from their first-person experience with the 3D environments. It opens up many questions we can ask as how to deploy more and more intelligent agents.

### General understanding of third-person images/videos

To efficiently learn, humans not only rely on our own personal experience to learn, we also try to learn by watching how other people behave. Fortunately, it is very easy to access such data on the Internet nowadays. We can see people doing all kinds of things on Youtube and social media. While the current trend in Computer vision is to understand these internet images/videos by learning only from passively collected data, we believe that learning visual parsing and perception on these active embodied agents might actually make the problem easier since the agents now can collect rich and dense data around the objects they care about. One key question then

is how can we deploy the visual understanding we have developed on these embodied agents for them to understand these images and videos available on the internet, so the agents can scale up their knowledge acquisition?

Here are some key limitations in the present work that hinders us to directly apply our visual perception module on internet images and videos. One limitation of the current model is that it requires reliable camera pose estimation to aggregate frames across different viewpoints. While camera poses are directly accessible on a physical agent through inverse kinematics, they are unavailable in Internet videos and they need to be estimated. Additionally, in our work, we show having reliable depth sensors can boost the performance of the model, but this again needs to be estimated for an internet video. Having a robust and generic camera pose and depth estimator will be critical to run the model on these Internet videos. Another key limitation is the need of storing and applying computation on a 4D tensor latent space. Exploiting the sparsity of our 4D tensors to save GPU memory is an important direction for scaling up our model to large scenes.

## **Parsing Objects into functional parts**

Being able to use existing knowledge to explore is good, but how effectively we can use existing knowledge depends much on how well we organize them. To acquire new knowledge or concepts more efficiently, we need to improve the way we organize existing knowledge. In the thesis, we discuss how to correspond rigid objects based on their appearance. To push forward, I think we should further factorize the learned whole-object visual concepts into functional parts. By splitting concepts into smaller entities, the model can discover more shared concepts across object instances, and can learn new concepts efficiently by exploiting the combinations between these factorized concepts. By learning about functional properties of the part, the model can organize objects in a way that will aid their learning in intelligently interacting with objects. Learning diverse and discrete concepts further provides the machines the ability to formulate discrete numbers of hypotheses, which enable them to collect informative data in a systematic way.

## **Integrating other sensor modalities**

In this thesis, we only consider how to aggregate and parse information visual inputs, however, there are a lot more sensors that we can quipe on an embodied agent and they all can contribute unique information about the world. For instance, by having a haptic and force sensor, the agents can easily understand invisible physics properties, such as weights and friction; by having audio sensor, the agents can sense critical event that is hard to observe purely from the change in the visual observations, e.g., pushing the button on a toy to make it play music. The 3D feature representation space we proposed is a suitable representation space for us to ground jointly information coming from all senses.

## **Handling deformable and articulated objects**

In both our concept learning and dynamics learning work, we consider rigidly moving objects. Learning object dynamics and concepts of soft bodies, articulated structure and fluids would empower the agents knowledge about how objects actually work and how to act efficiently to them. Learning about these more complex object dynamics will require forecasting dense 3D motion fields, or considering sub-object (part or particle) graphs. For learning concepts and object correspondence, we will require adding not non-rigidly deformation between the category template and the instances. Adding such deformable parametrization would increase the expressiveness of the prototype dictionary.

## **Improving Exploration through Visual Parsing**

To continually and efficiently collect more informative data, the agents need to find better ways to explore. Inspired by the findings in psychology, I propose to use the developed visual cues to guide the exploration in skill learning. When the agents encounter new objects, they should use the learned visual parsing to parse the objects into meaning parts and structure, or use the learned visual features to retrieve relevant acquired skills to aid current skill deployment. Being able to reuse existing skills and use visual cues to guide exploration is a key and interesting venue to efficient skill acquisition.

## **Learning from other research disciplines**

Most of my works are inspired by reading and talking to people outside the region of machine learning and computer vision. I believe to answer those three questions we state upfront, we will need to bring together insights and findings from multiple research disciplines in artificial intelligence including machine learning, computer vision, robotics, linguistic and psychology. Developmental psychology studies what cognitive or motor abilities an intelligent agent (humans) possess, in which order, and how they get developed throughout a persons lifespan. Robotics research tells us how to make things work on physical robots, and what type of visual parsing outputs will enable that. Linguistics explains how humans think in terms of symbols, and how it is related to our actual experience with the world. Notice how related these findings are related to the questions we seek to answer! We already can get many potential answers from the existing works and theories in these research disciplines. I hope to see how learning these disciplines can provide new insights to my research. I also hope to see what we find in my research can provide new insights in these fields.



# Bibliography

- [1] <http://bulletphysics.org/wordpress/>. 8.3.4
- [2] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *CoRR*, abs/1606.07419, 2016. URL <http://arxiv.org/abs/1606.07419>. 5.1
- [3] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *CoRR*, abs/1707.01495, 2017. URL <http://arxiv.org/abs/1707.01495>. 5.1, 5.2.2, (a), 5.3.1, 5.3.2
- [4] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020. 4.3, 1
- [5] Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. *CoRR*, abs/1612.00222, 2016. URL <http://arxiv.org/abs/1612.00222>. 4.1, 4.2.1, 4.3, 4.1, 4.2, 4.3
- [6] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018. 4.2.1
- [7] Anna M. Borghi and Felice Cimatti. Embodied cognition and beyond: Acting and sensing the body. *Neuropsychologia*, 48(3):763–773, 2010. ISSN 0028-3932. doi: <https://doi.org/10.1016/j.neuropsychologia.2009.10.029>. URL <https://www.sciencedirect.com/science/article/pii/S0028393209004369>. The Sense of Body. 1
- [8] Thomas Brox and Jitendra Malik. Object segmentation by long term analysis of point trajectories. In *ECCV*. 2010. 3.3
- [9] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. 2.2.1, 4.3, 4.3.1, 5.3.1
- [10] Yinbo Chen, Xiaolong Wang, Zhuang Liu, Huijuan Xu, and Trevor Darrell. A new meta-

baseline for few-shot learning, 2020. 7.3.1

- [11] Ricson Cheng, Ziyang Wang, and Katerina Fragkiadaki. Geometry-aware recurrent neural networks for active visual recognition. In *NIPS*, 2018. (document), 2.2.1, 2.4, 2.2.2, 2.2
- [12] Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. URL <http://arxiv.org/abs/1406.1078>. 2.2
- [13] Volkan Cirik, Taylor Berg-Kirkpatrick, and Louis-Philippe Morency. Using syntax to ground referring expressions in natural images. In *AAAI*, 2018. 8.2.2
- [14] Sudeep Dasari, Frederik Ebert, Stephen Tian, Suraj Nair, Bernadette Bucher, Karl Schmeckpeper, Siddharth Singh, Sergey Levine, and Chelsea Finn. Robonet: Large-scale multi-robot learning, 2019. 5.1
- [15] Jean Decety and D. H. Ingvar. Brain structures participating in mental simulation of motor behavior: a neuropsychological interpretation. *Acta psychologica*, 73 1:13–34, 1990. 4.1
- [16] Zhiwei Deng, Jiacheng Chen, YIFANG FU, and Greg Mori. Probabilistic neural programmed networks for scene generation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 4028–4038. Curran Associates, Inc., 2018. URL <http://papers.nips.cc/paper/7658-probabilistic-neural-programmed-networks-for-scene-generation.pdf>. (document), 8.1, 8.4, 8.3.1, 8.3.2, 8.3.4
- [17] Bhuwan Dhingra, Hanxiao Liu, William W. Cohen, and Ruslan Salakhutdinov. Gated-attention readers for text comprehension. *CoRR*, abs/1606.01549, 2016. URL <http://arxiv.org/abs/1606.01549>. 8.1
- [18] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *CORL*, pages 1–16, 2017. 3.2, 6.3
- [19] Frederik Ebert, Chelsea Finn, Alex X. Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. *CoRR*, abs/1710.05268, 2017. URL <http://arxiv.org/abs/1710.05268>. 4.1
- [20] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018. 4.3, 3, 4.3.2, 4.3
- [21] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2366–2374. Curran Associates, Inc., 2014. URL <http://papers.nips.cc/paper/5539-depth-map-prediction-from-a-single-image-using-a-multi-scale-deep.pdf>. 2.2.2
- [22] S. M. Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, Koray Kavukcuoglu,



and Geoffrey E. Hinton. Attend, infer, repeat: Fast scene understanding with generative models. *CoRR*, abs/1603.08575, 2016. URL <http://arxiv.org/abs/1603.08575>. 7.3.1

- [23] S. M. Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S. Morcos, Marta Garnelo, Avraham Ruderman, Andrei A. Rusu, Ivo Danihelka, Karol Gregor, David P. Reichert, Lars Buesing, Theophane Weber, Oriol Vinyals, Dan Rosenbaum, Neil Rabinowitz, Helen King, Chloe Hillier, Matt Botvinick, Daan Wierstra, Koray Kavukcuoglu, and Demis Hassabis. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018. ISSN 0036-8075. doi: 10.1126/science.aar6170. URL <http://science.sciencemag.org/content/360/6394/1204>. (document), 2.2, 2.2.1, 2.2.1, 2.4, 2.5, 3.1, 7.1
- [24] Kuan Fang, Yuke Zhu, Animesh Garg, Silvio Savarese, and Li Fei-Fei. Dynamics learning with cascaded variational inference for multi-step manipulation, 2019. 4.2.2
- [25] S. Feng, X. Xinjilefu, C. G. Atkeson, and J. Kim. Optimization based controller design and implementation for the atlas robot in the darpa robotics challenge finals. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 1028–1035, 2015. 5.1
- [26] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. *CoRR*, abs/1610.00696, 2016. URL <http://arxiv.org/abs/1610.00696>. 4.3.2
- [27] Carl Gabbard. The role of mental simulation in embodied cognition. *Early Child Development and Care*, 183(5):643–650, 2013. 4.1
- [28] Dashan Gao, Vijay Mahadevan, and Nuno Vasconcelos. On the plausibility of the discriminant center-surround hypothesis for visual saliency. *Journal of vision*, 8, 2008. 3.3
- [29] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013. 3.2.2
- [30] James J. Gibson. *The Ecological Approach to Visual Perception*. Houghton Mifflin, 1979. 5.2.1
- [31] Shixiang Gu, Ethan Holly, Timothy P. Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation. *CoRR*, abs/1610.00633, 2016. URL <http://arxiv.org/abs/1610.00633>. 5.1
- [32] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565, 2019. 4.1, 4.3, 4, 4.3
- [33] Adam W Harley, Fangyu Li, Shrinidhi K Lakshmikanth, Xian Zhou, Hsiao-Yu Fish Tung, and Katerina Fragkiadaki. Embodied view-contrastive 3d feature learning. *arXiv*, 2019. 4.1, 6.3.3
- [34] Adam W. Harley, Fangyu Li, Shrinidhi K. Lakshmikanth, Xian Zhou, Hsiao-Yu Fish Tung, and Katerina Fragkiadaki. Embodied view-contrastive 3d feature learning. *CoRR*, abs/1906.03764, 2019. URL <http://arxiv.org/abs/1906.03764>. 5.3.3

- [35] Adam W. Harley, Fangyu Li, Shrinidhi K. Lakshmikanth, Xian Zhou, Hsiao-Yu Fish Tung, and Katerina Fragkiadaki. Visual representation learning with 3d view-constrastive inverse graphics networks. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=BJxt60VtPr>. 1.1, 1.5, 0
- [36] Masahiko Haruno, Daniel M Wolpert, and Mitsuo Kawato. Multiple paired forward-inverse models for human motor learning and control. In M. J. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 31–37. MIT Press, 1999. URL <http://papers.nips.cc/paper/1585-multiple-paired-forward-inverse-models-for-human-motor-learning-a.pdf>. 4.1
- [37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL <http://arxiv.org/abs/1512.03385>. 3.2.1
- [38] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017. URL <http://arxiv.org/abs/1703.06870>. 2.1, 2.2.2, 4.2
- [39] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017. URL <http://arxiv.org/abs/1703.06870>. 6.1
- [40] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning, 2019. 6.2.2, 6.3.3
- [41] R. Held and A. Hein. Movement-produced stimulation in the development of visually guided behavior. *Journal of comparative and physiological psychology*, 56:872–6, 1963. 1
- [42] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1693–1701. Curran Associates, Inc., 2015. URL <http://papers.nips.cc/paper/5945-teaching-machines-to-read-and-comprehend.pdf>. 8.1
- [43] Michael Hornacek, Andrew Fitzgibbon, and Carsten Rother. SphereFlow: 6 DoF scene flow from RGB-D pairs. In *CVPR*, 2014. 3.3
- [44] Jun-Ting Hsieh, Bingbin Liu, De-An Huang, Li F Fei-Fei, and Juan Carlos Niebles. Learning to decompose and disentangle representations for video prediction. In *NIPS*, pages 517–526, 2018. 3.3.1
- [45] Ronghang Hu, Marcus Rohrbach, Jacob Andreas, Trevor Darrell, and Kate Saenko. Modeling relationships in referential expressions with compositional modular networks. *CoRR*, abs/1611.09978, 2016. URL <http://arxiv.org/abs/1611.09978>. 7.1
- [46] Ronghang Hu, Marcus Rohrbach, Jacob Andreas, Trevor Darrell, and Kate Saenko. Modeling relationships in referential expressions with compositional modular networks. 11 2016. 8.1, 8.2.2, 8.3.3, 8.3.3

- [47] Xun Huang and Serge J. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. *CoRR*, abs/1703.06868, 2017. URL <http://arxiv.org/abs/1703.06868>. 7.1, 7.2.1
- [48] Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. Multimodal unsupervised image-to-image translation. In *ECCV*, 2018. 7.2.1, 7.3.1
- [49] Hamid Izadinia, Qi Shan, and Steven M. Seitz. IM2CAD. *CoRR*, abs/1608.05137, 2016. 4.1
- [50] Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12627–12637, 2019. 5.1
- [51] Michael Janner, Sergey Levine, William T. Freeman, Joshua B. Tenenbaum, Chelsea Finn, and Jiajun Wu. Reasoning about physical interactions with object-oriented prediction and planning. *CoRR*, abs/1812.10972, 2018. URL <http://arxiv.org/abs/1812.10972>. 4.1, 4.2.1, 4.3, 4.3.1
- [52] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross B. Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. *CoRR*, abs/1612.06890, 2016. URL <http://arxiv.org/abs/1612.06890>. 8.2, 8.3
- [53] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2901–2910, 2017. (document), 6.3, 7.3.1, 7.1, 7.2
- [54] Ran Ju, Yang Liu, Tongwei Ren, Ling Ge, and Gangshan Wu. Depth-aware salient object detection using anisotropic center-surround difference. *Signal Processing: Image Communication*, 38:115–126, 2015. 6.2.3
- [55] Rudolf Kadlec, Martin Schmid, Ondrej Bajgar, and Jan Kleindienst. Text understanding with the attention sum reader network. *CoRR*, abs/1603.01547, 2016. URL <http://arxiv.org/abs/1603.01547>. 8.1
- [56] Abhishek Kar, Christian Häne, and Jitendra Malik. Learning a multi-view stereo machine. *CoRR*, abs/1708.05375, 2017. URL <http://arxiv.org/abs/1708.05375>. 2.2.2, 3.1
- [57] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4401–4410, 2019. 7.2.1
- [58] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems, 2018. 4.1, 4.2.1
- [59] Dominik A Klein and Simone Frintrop. Center-surround divergence of feature statistics for salient object detection. In *2011 International Conference on Computer Vision*, pages

2214–2219. IEEE, 2011. 6.2.3

- [60] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. 2015. 6.1
- [61] Adam R. Kosiosek, Hyunjik Kim, Ingmar Posner, and Yee Whye Teh. Sequential attend, infer, repeat: Generative modelling of moving objects. In *NIPS*, 2018. 3.3.1
- [62] Oliver Kroemer, Scott Niekum, and George Konidaris. A review of robot learning for manipulation: Challenges, representations, and algorithms, 2019. 5.1
- [63] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955. 6.3.2
- [64] Tejas D. Kulkarni, Will Whitney, Pushmeet Kohli, and Joshua B. Tenenbaum. Deep convolutional inverse graphics network. *CoRR*, abs/1503.03167, 2015. URL <http://arxiv.org/abs/1503.03167>. 8.1
- [65] Tejas D Kulkarni, William F Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. In *Advances in neural information processing systems*, pages 2539–2547, 2015. 4.1
- [66] Vikash Kumar, Abhishek Gupta, Emanuel Todorov, and Sergey Levine. Learning dexterous manipulation policies from experience and imitation. *CoRR*, abs/1611.05095, 2016. URL <http://arxiv.org/abs/1611.05095>. 8.3.4
- [67] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *CoRR*, abs/1504.00702, 2015. URL <http://arxiv.org/abs/1504.00702>. 1.2, 5.1, 5.1, 5.3.1
- [68] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.*, 17(1):1334–1373, January 2016. ISSN 1532-4435. URL <http://dl.acm.org/citation.cfm?id=2946645.2946684>. 8.3.4
- [69] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *ICLR*, 2019. 4.1
- [70] Yunzhu Li, Toru Lin, Kexin Yi, Daniel Bear, Daniel L.K. Yamins, Jiajun Wu, Joshua B. Tenenbaum, and Antonio Torralba. Visual grounding of learned physical models. In *International Conference on Machine Learning*, 2020. 4.1
- [71] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Manfred Otto Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2016. 5.1, 5.3.1
- [72] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008. 7.3.1
- [73] Vijay Mahadevan and Nuno Vasconcelos. Spatiotemporal saliency in dynamic scenes. *TPAMI*, 32, 2010. 3.3
- [74] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu,

Juan Aparicio, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. 07 2017. doi: 10.15607/RSS.2017.XIII.058. 5.1, 5.3.4

- [75] Fabian Manhardt, Wadim Kehl, Nassir Navab, and Federico Tombari. Deep model-based 6d pose refinement in rgb. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 800–815, 2018. 6.1
- [76] Manolis Savva\*, Abhishek Kadian\*, Oleksandr Maksymets\*, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A Platform for Embodied AI Research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019. 7.3.1
- [77] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=rJgMlhRctm>. 7.1, 7.3.2, 8.3
- [78] Dushyant Mehta, Oleksandr Sotnychenko, Franziska Mueller, Weipeng Xu, Srinath Sridhar, Gerard Pons-Moll, and Christian Theobalt. Single-shot multi-person 3d pose estimation from monocular rgb. In *2018 International Conference on 3D Vision (3DV)*, pages 120–130. IEEE, 2018. 6.1
- [79] L. Meteyard, Sara Rodríguez Cuadrado, B. Bahrami, and G. Vigliocco. Coming of age: A review of embodiment and the neuroscience of semantics. *Cortex*, 48:788–804, 2012. 1
- [80] R. C. Miall and D. M. Wolpert. Forward models for physiological motor control. *Neural Netw.*, 9(8):1265–1279, November 1996. ISSN 0893-6080. doi: 10.1016/S0893-6080(96)00035-4. URL [http://dx.doi.org/10.1016/S0893-6080\(96\)00035-4](http://dx.doi.org/10.1016/S0893-6080(96)00035-4). 4.1
- [81] Niloy J Mitra, Natasha Gelfand, Helmut Pottmann, and Leonidas Guibas. Registration of point cloud data from a geometric optimization perspective. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 22–31, 2004. 6.3.2
- [82] Arsalan Mousavian, Clemens Eppner, and Dieter Fox. 6-dof graspnet: Variational grasp generation for object manipulation. *CoRR*, abs/1905.10520, 2019. URL <http://arxiv.org/abs/1905.10520>. 5.1, 5.3.4
- [83] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li Fei-Fei, Joshua B Tenenbaum, and Daniel LK Yamins. Flexible neural representation for physics prediction. In *NIPS*, 2018. 4.1
- [84] Montiel J. M. M. Mur-Artal, Raúl and Juan D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015. doi: 10.1109/TRO.2015.2463671. 1.1
- [85] Venkatraman Narayanan and Maxim Likhachev. Deliberative object pose estimation in clutter. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3125–3130. IEEE, 2017. 6.1

- [86] P. Ochs and T. Brox. Object segmentation in video: a hierarchical variational approach for turning point trajectories into dense regions. In *ICCV*, 2011. 3.3
- [87] Bruno A. Olshausen. Perception as an inference problem. 2013. 8.1
- [88] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv:1807.03748*, 2018. 3.2
- [89] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub W. Pachocki, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177, 2018. URL <http://arxiv.org/abs/1808.00177>. 5.1, 5.1
- [90] Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A. Efros, and Trevor Darrell. Zero-shot visual imitation. In *ICLR*, 2018. 4.3.2
- [91] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. *CoRR*, abs/1509.06825, 2015. URL <http://arxiv.org/abs/1509.06825>. 5.1
- [92] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 3406–3413. IEEE, 2016. 6.1
- [93] Mihir Prabhudesai, Shamit Lal, Hsiao-Yu Fish Tung, Adam W Harley, Shubhankar Potdar, and Katerina Fragkiadaki. 3d object recognition by corresponding and quantizing neural 3d scene representations. *CVPR Minds vs. Machine (MVM) workshop*, 2020. 1.3, 1.5, 0
- [94] Mihir Prabhudesai, Hsiao-Yu Fish Tung, Syed Ashar Javed, Maximilian Sieb, Adam W. Harley, and Katerina Fragkiadaki. Embodied language grounding with implicit 3d visual feature representations, 2020. 1.4, 1.5, 7.3.3, 0
- [95] Mihir Prabhudesai, Shamit Lal, Darshan Patil, Hsiao-Yu Tung, Adam W Harley, and Katerina Fragkiadaki. Disentangling 3d prototypical networks for few-shot concept learning. *ICLR*, 2021. 1.3, 1.5
- [96] Yuzhe Qin, Rui Chen, Hao Zhu, Meng Song, Jing Xu, and Hao Su. S4g: Amodal single-view single-shot se(3) grasp detection in cluttered scenes, 2019. 5.1
- [97] Zengyi Qin, Kuan Fang, Yuke Zhu, Li Fei-Fei, and Silvio Savarese. Keto: Learning keypoint representations for tool manipulation. *arXiv preprint arXiv:1910.11977*, 2019. 5.2
- [98] Mahdi Rad, Markus Oberweger, and Vincent Lepetit. Feature mapping for learning fast and accurate 3d pose inference from synthetic images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4663–4672, 2018. 6.1
- [99] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *CoRR*, abs/1709.10087, 2017. URL <http://arxiv.org/abs/1709.10087>. 5.2.2, (a), 5.3.2

- [100] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems* 28, pages 91–99. Curran Associates, Inc., 2015. URL <http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks>. pdf. (document), 8.3.3, 8.1, 8.3.3
- [101] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. URL <http://arxiv.org/abs/1506.01497>. 2.2.2
- [102] Ronan Riochet, Mario Yncente Castro, Mathieu Bernard, Adam Lerer, Rob Fergus, Véronique Izard, and Emmanuel Dupoux. Intphys: A benchmark for visual intuitive physics reasoning. 2019. 8.3.2
- [103] Lukasz Romaszko, Christopher K. I. Williams, Pol Moreno, and Pushmeet Kohli. Vision-as-inverse-graphics: Obtaining a rich 3d explanation of a scene from a single image. In *ICCV Workshops*, Oct 2017. 4.1, 8.1
- [104] 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*, pages 851–859, 2017. 4.1
- [105] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL <http://arxiv.org/abs/1505.04597>. 2.2
- [106] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. No-regret reductions for imitation learning and structured prediction. *CoRR*, abs/1011.0686, 2010. URL <http://arxiv.org/abs/1011.0686>. 5.3.1
- [107] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. PIFu: Pixel-aligned implicit function for high-resolution clothed human digitization. *arXiv:1905.05172*, 2019. 3.1
- [108] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. *arXiv:1806.01242*, 2018. 4.1
- [109] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks, 02 2020. 4.1
- [110] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied AI research. *CoRR*, abs/1904.01201, 2019. URL <http://arxiv.org/abs/1904.01201>. 6.3
- [111] Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. In *CVPR*, 2015. 3.2.1

- [112] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarczyk, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. 2017. URL <https://openreview.net/pdf?id=BlckMDqlg>. 8.2.1
- [113] Xi Shen, Alexei A. Efros, and Mathieu Aubry. Discovering visual patterns in art collections with spatially-consistent feature learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 6.2.2, 6.2.2
- [114] Arjun Singh, James Sha, Karthik S. Narayan, Tudor Achim, and Pieter Abbeel. Bigbird: A large-scale 3d database of object instances. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 509–516, 2014. 6.3
- [115] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhöfer. DeepVoxels: Learning persistent 3D feature embeddings. In *CVPR*, 2019. 3.1
- [116] Linda Smith and Michael Gasser. The development of embodied cognition: Six lessons from babies. *NIPS*, 2017. 1
- [117] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4077–4087. Curran Associates, Inc., 2017. URL <http://papers.nips.cc/paper/6996-prototypical-networks-for-few-shot-learning.pdf>. 7.2.2, 7.3.1
- [118] Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. *CoRR*, abs/1703.05175, 2017. URL <http://arxiv.org/abs/1703.05175>. 6.1
- [119] Kihyuk Sohn. Improved deep metric learning with multi-class N-pair loss objective. In *NIPS*, pages 1857–1865, 2016. 3.2.1
- [120] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep metric learning via lifted structured feature embedding. In *CVPR*, 2016. 3.2.1
- [121] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J. Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, Anton Clarkson, Mingfei Yan, Brian Budge, Yajie Yan, Xiaqing Pan, June Yon, Yuyang Zou, Kimberly Leon, Nigel Carter, Jesus Briales, Tyler Gillingham, Elias Mueggler, Luis Pesqueira, Manolis Savva, Dhruv Batra, Hauke M. Strasdat, Renzo De Nardi, Michael Goesele, Steven Lovegrove, and Richard Newcombe. The Replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019. 6.3, 7.3.1
- [122] Edgar Sucar, Kentaro Wada, and Andrew Davison. Neural object descriptors for multi-view shape reconstruction. *arXiv preprint arXiv:2004.04485*, 2020. 6.1
- [123] D. Sun, S. Roth, and M. J. Black. Secrets of optical flow estimation and their principles. In *CVPR*, 2010. 3.3
- [124] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. PWC-Net: CNNs for optical



- flow using pyramid, warping, and cost volume. In *CVPR*, 2018. 3.3, 3.3.1
- [125] Martin Sundermeyer, Zoltan-Csaba Marton, Maximilian Durner, Manuel Brucker, and Rudolph Triebel. Implicit 3d orientation learning for 6d object detection from rgb images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 699–715, 2018. 6.1
  - [126] Yuval Tassa, Tom Erez, and William D Smart. Receding horizon differential dynamic programming. In *Advances in neural information processing systems*, pages 1465–1472, 2008. 4.2.2
  - [127] Yuval Tassa, Nicolas Mansard, and Emanuel Todorov. Control-limited differential dynamic programming. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1168–1175, 2014. 8.2.3
  - [128] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Single-view to multi-view: Reconstructing unseen views with a convolutional network. *CoRR*, abs/1511.06702, 2015. URL <http://arxiv.org/abs/1511.06702>. 3.1
  - [129] Andreas ten Pas and Robert Platt. Using geometry to detect grasp poses in 3d point clouds. In *Robotics Research*, pages 307–324. Springer, 2018. 6.1
  - [130] Joshua Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *CoRR*, abs/1703.06907, 2017. URL <http://arxiv.org/abs/1703.06907>. 4.3.1
  - [131] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033. IEEE, 2012. ISBN 978-1-4673-1737-5. URL <http://dblp.uni-trier.de/db/conf/iros/iros2012.html#TodorovET12>. 5.3
  - [132] Shubham Tulsiani, Saurabh Gupta, David F. Fouhey, Alexei A. Efros, and Jitendra Malik. Factoring shape, pose, and layout from the 2d image of a 3d scene. *CoRR*, abs/1712.01812, 2017. URL <http://arxiv.org/abs/1712.01812>. 6.1
  - [133] Fish Tung and Katerina Fragkiadaki. Reward learning using natural language. *CVPR*, 2018. 8.2.3
  - [134] Hsiao-Yu Fish Tung, Adam Harley, William Seto, and Katerina Fragkiadaki. Adversarial inverse graphics networks: Learning 2d-to-3d lifting and image-to-image translation with unpaired supervision. *ICCV*, 2017. 8.1
  - [135] Hsiao-Yu Fish Tung, Ricson Cheng, and Katerina Fragkiadaki. Learning spatial common sense with geometry-aware recurrent networks. In *CVPR*, 2019. 4.1, 5.2.1, (c), 5.3.3, 6.3.3, 7.3.1, 8.1
  - [136] Hsiao-Yu Fish Tung, Ricson Cheng, and Katerina Fragkiadaki. Learning spatial common sense with geometry-aware recurrent networks. *CVPR*, 2019. URL <http://arxiv.org/abs/1901.00003>. 1.1, 1.5, 0
  - [137] Hsiao-Yu Fish Tung, Zhou Xian, Mihir Prabhudesai, Shamit Lal, and Katerina Fragkiadaki. 3d-oes: Viewpoint-invariant object-factorized environment simulators. 2021. 1.2,

1.5, 0

- [138] Oriol Vinyals, Charles Blundell, Timothy P. Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. *CoRR*, abs/1606.04080, 2016. URL <http://arxiv.org/abs/1606.04080>. 6.1
- [139] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *CoRR*, abs/1410.3916, 2014. URL <http://arxiv.org/abs/1410.3916>. 8.1
- [140] Chao-Yuan Wu, R Manmatha, Alexander J Smola, and Philipp Krahenbuhl. Sampling matters in deep embedding learning. In *ICCV*, pages 2840–2848, 2017. 3.2.1
- [141] Jiajun Wu, Erika Lu, Pushmeet Kohli, Bill Freeman, and Josh Tenenbaum. Learning to see physics via visual de-animation. In *Advances in Neural Information Processing Systems*, pages 153–164, 2017. 4.3, 1
- [142] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. <https://github.com/facebookresearch/detectron2>, 2019. 6.2
- [143] Yufei Ye, Dhiraj Gandhi, Abhinav Gupta, and Shubham Tulsiani. Object-centric forward modeling for model predictive control, 2019. 4.3.2
- [144] Yufei Ye, Dhiraj Gandhi, Abhinav Gupta, and Shubham Tulsiani. Object-centric forward modeling for model predictive control. *the Conference on Robot Learning (CoRL)*, 2019. 4.1, 4.2.1, 4.3, 4.1, 4.2, 4.3
- [145] Jason J. Yu, Adam W. Harley, and Konstantinos G. Derpanis. Back to basics: Unsupervised learning of optical flow via brightness constancy and motion smoothness. In *ECCV*, 2016. 3.3
- [146] Kuan-Ting Yu, Maria Bauzá, Nima Fazeli, and Alberto Rodriguez. More than a million ways to be pushed: A high-fidelity experimental data set of planar pushing. *CoRR*, abs/1604.04038, 2016. 4.3, 4.3.1
- [147] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018. (b), 5.3.1
- [148] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2.2.2