Towards Generalizable Open-World Robot Manipulation by Training with Off-Domain Data

Yi Li

A dissertation

submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

University of Washington
2025

Reading Committee:
Dieter Fox, Chair
Joshua Smith
Abhishek Gupta

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Towards Generalizable Open-World Robot Manipulation by Training with Off-Domain Data

Yi Li

Chair of the Supervisory Committee:
Dieter Fox
Computer Science and Engineering

Achieving generalization in open-world robotic manipulation is a critical step toward deploying autonomous agents in dynamic, unstructured environments. However, learning manipulation skills that generalize to unseen objects, layouts, and natural language instructions remains challenging, particularly due to the scarcity and narrow coverage of real-world robot data. This dissertation explores how off-domain supervision—ranging from synthetic data to large-scale vision-language pretraining—can be harnessed to build scalable, generalist robot systems.

We present three systems that progressively tackle generalization across different levels of the perception-to-action pipeline. First, we introduce DeepIM, a pose refinement framework based on iterative render-and-compare, which enables accurate 6D pose estimation using only RGB input. DeepIM demonstrates robust generalization to unseen objects and views, providing a foundation for geometry-aware manipulation. Next, we propose STOW, a discrete-frame segmentation and tracking method trained entirely on synthetic data. STOW exhibits strong sim-to-real transfer in cluttered warehouse environments by learning object-centric, temporally consistent representations, enabling robust multi-object scene understanding. Finally, we develop HAMSTER, a hierarchical vision-language-action model that integrates pretrained vision-language models with robot control via an intermediate abstraction of spatial sketch trajectories. HAMSTER enables the interpretation of diverse natural language instructions and their execution across varied semantic, geometric, and visual contexts.

Together, these systems chart a path from model-based to model-free design in robotics, demonstrating that careful use of intermediate representations, modularity, and off-domain learning can bridge the gap between narrow robot deployments and open-world capability. By leveraging world knowledge and pretraining from large-scale datasets, this work contributes toward the long-term vision of scalable, generalist manipulation systems that adapt flexibly to real-world complexity.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my family—especially my mother, Wenhong Sun—who has been a constant source of love, strength, and encouragement throughout my life. She gave me the name Yi with the hope that I would contribute meaningfully to society. Her unwavering belief in me and complete trust empowered me to pursue my own path, beyond the traditional pursuit of stability. During moments of doubt, her boundless patience and our daily conversations gave me courage and clarity. She remains my greatest source of comfort and inspiration.

I am profoundly thankful to my advisor, Dieter Fox, for his exceptional mentorship, patience, and unwavering support. The two difficult years I spent stuck in China during the COVID pandemic were particularly challenging, but Dieter's compassion and encouragement helped me stay focused and resilient in the face of setbacks. His openness, ambition, humility, sense of humor, and remarkable athleticism continue to inspire me. It has been a privilege to learn from and work with such a dedicated and brilliant advisor.

I am also deeply grateful to my mentors—Yu Xiang, Kaichun Mo, and Ankit Goyal. Before beginning my PhD, my background was primarily in computer vision, with limited exposure to robotics. Their generous guidance and belief in me gave me the confidence to explore new directions and pursue ambitious projects, even in uncertain times.

I sincerely appreciate the opportunity to collaborate with the NVIDIA Seattle Robotics Lab and to work alongside such a talented and driven team. Special thanks to Prof. Abhishek Gupta, Anqi Li, Arsalan Mousavian, Caelan Garrett, Fabio Ramos, Jason Lu, Lucas Manuelli, and Wei Yang for their invaluable insights, thoughtful feedback, and continual encouragement.

I would also like to extend heartfelt thanks to Gu Wang, Muru Zhang, Yuquan Deng, Jesse Zhang, Joel Jang, Sanjar Normuradov, Soofiyan Atar, Karthik Desingh, and Markus Grotz. Their support and collaboration lightened the challenges of this journey and made many milestones achievable.

To my labmates—Adam Fishman, Arun Byravan, Chris Xie, Daniel Gordon, Ge Yan, Helen Wang, Jiafei Duan, Junha Roh, Karthik Desingh, Marius Memmel, Mohit Shridhar, Vinitha Ranganeni, Wentao Yuan, Xiangyun Meng, and Zoey Chen—thank you for the insightful discussions, stimulating brainstorming sessions, and the many memorable moments we shared over boba tea and at lab retreats.

Beyond the RSE Lab, I'm grateful to my friends Liyiming Ke, Boling Yang, Yunchuang Zhang, Octi Zhang, Chuning Zhu, and Bernie Zhu for their generous technical support and shared expe-

riences in Seattle.

I'd also like to thank my committee members, Dieter Fox, Joshua Smith, and Abhishek Gupta, for their thoughtful feedback and support during this process. Their questions and perspectives helped me see my work more clearly and pushed me to refine my ideas.

Lastly, I want to express my heartfelt appreciation to my girlfriend for her patience, understanding, and steadfast support. Being apart in different countries for years and enduring a PhD that stretched longer than I had promised was not easy. Thank you for your trust, your unwavering commitment, and your quiet strength.

To each and every one of you—thank you. Your support, encouragement, and companionship have made this journey not only possible but deeply meaningful and unforgettable.

DEDICATION

To my family

Contents

Chapter 1

Introduction

In both industrial and household environments, the ability for robots to *manipulate objects in open-world scenarios*—where tasks, objects, and environments vary in unpredictable ways—is increasingly seen as a core capability for autonomy. As robots are deployed beyond structured factories into homes, warehouses, and public spaces, their success depends on being able to handle *novel objects*, *ambiguous instructions*, and *changing configurations* without requiring exhaustive reprogramming or retraining. Yet today's robotic systems still largely fail to meet this bar.

Much of the current progress in robot manipulation is driven by *in-domain training paradigms*: data is collected in fixed environments using a specific robot and sensor configuration, then used to train a model for tasks in that same domain. This has enabled high performance on benchmarks and controlled tests Goyal et al. [2024]; Gervet et al. [2023]; Shridhar et al. [2023]; Zhao et al. [2023], but such methods break down in *real-world deployments*. The mismatch between tightly controlled training data and the messiness of real life means these models often fail in unseen homes, on new object types, or when given slightly reworded instructions. In short, they are *overfitted to their training domain*, and brittle in the face of change.

To illustrate this challenge more concretely, consider a household task where a user asks the robot to "pick up the silver can and pour it to the cup." A capable system should not only detect the relevant objects but also infer their spatial relation and plan a valid action. Now imagine the instruction is rephrased as "Diet Coke, please" or the visual appearance of the can and cup differ from training, or the table is a different size or shape. Despite these surface-level changes, the core task remains the same. A competent robot must still ground the instruction semantically and visually, identify the relevant objects, and manipulate them accordingly. These small but meaningful variations exemplify the demands of **open-world manipulation**: the ability to generalize across instructions, object instances, and physical layouts, without brittle dependencies on training specifics.

Two key requirements for generalist robot systems are: they must be both **general-purpose** and **generalizable**. A general-purpose robot can perform a wide range of tasks rather than being confined to a narrow domain like bin picking or peg insertion. A generalizable robot, on the other hand, must apply its learned capabilities to novel environments and task variations. These two properties together define the vision of a "foundation model for robotics"—a robotic counterpart

to large pretrained models in language and vision. Like GPT-4 Achiam et al. [2023], which generalizes to diverse prompts and tasks due to its training scale and data diversity, we envision robotic systems that can similarly reason, perceive, and act under a broad distribution of tasks and scenes.

But why pursue generalization rather than perfection in narrow domains? The answer lies in the economics of real-world deployment: **reducing the marginal cost of adaptation**. While a specialist robot might achieve near-perfect accuracy in one kitchen, factory, or lab, the cost of retooling it for a new space is prohibitively high. Every new deployment requires fresh data collection, retraining, and manual adjustment. In contrast, a moderately accurate generalist that performs adequately across many settings is more useful and scalable. In practical robotics, this trade-off is critical. A system that generalizes "well enough" without additional tuning is often more valuable than one that performs optimally in a constrained setup but fails to transfer.

To build such systems, scale becomes necessary. Achieving robust generalization requires exposure to a wide variety of scenes, object types, and task formulations. However, acquiring real-world robotic data at this scale remains challenging. Unlike computer vision or NLP, where massive datasets can be scraped from the internet and labeled with minimal supervision, robotic data collection involves hardware, environment control, and complex annotation Black et al. [2024]. Efforts such as RT-1 Brohan et al. [2022] and BridgeData Walke et al. [2023] have begun to scale robot datasets, but the process is still expensive and typically limited to a single robot platform or lab setting. Even with recent innovations in teleoperation pipelines Fu et al. [2024b]; Khazatsky et al. [2024], the diversity, scalability, and reproducibility of robot data lag significantly behind other modalities.

This motivates our use of **off-domain data**—data that is not collected in the deployment environment or on the deployment robot. Off-domain sources include simulated manipulation environments James et al. [2020], egocentric video datasets like Ego4D Grauman et al. [2022], and large-scale robot demonstration corpora O'Neill et al. [2024]. These sources offer scale and diversity that are difficult to achieve with in-domain robot data, enabling learning from a wide distribution of tasks, viewpoints, and interactions. While such data may not match the test setting exactly, we hypothesize that it can endow models with transferable priors that improve robustness and generalization. In this sense, off-domain data serves a similar role to internet pretraining in vision and language models—providing a broad base of knowledge that can be adapted to embodiment tasks with minimal real-world experience.

Of course, working with off-domain data brings its own challenges. Distributional shifts across embodiment, observation space, and control dynamics can degrade performance. Bridging these gaps requires both the right abstractions and representations that generalize across domains. To address this, we explore how intermediate-level interfaces—such as 6D poses, segmentation masks, or 2D trajectories—can help decouple learning from low-level embodiment. These abstractions act as translation layers between vision-language understanding and embodied robot execution. For example, if a robot learns to follow a 2D trajectory sketched on an image, the model need not reason about joint-level commands or control frequencies; it only needs to learn to translate high-level intent into actionable spatial trajectories..

This philosophy aligns with Sutton's bitter lesson Sutton [2019]: the most effective AI sys-

tems rely not on hand-crafted domain knowledge but on general-purpose methods that scale with data and computation. In this thesis, we observe that detailed object states—like 6D poses or instance masks—are often fragile, unavailable, or unnecessary for open-world tasks. Relying on them hinders scalability, especially when objects are novel or occluded. Instead, we find that abstract motion representations, such as 2D spatial sketches, are more robust and flexible targets for general-purpose manipulation, and are easier to acquire at scale.

This thesis investigates how to bring this approach to robotic manipulation. We present three projects that each address different levels of the manipulation stack—object, scene, and task—and explore how off-domain data can be leveraged to enable open-world behavior.

In Chapter 2, we introduce **DeepIM**, which revisits the problem of 6D object pose estimation. This work targets the object level of manipulation and proposes a render-and-compare framework for iterative pose refinement using only RGB input. The model predicts disentangled rotation and translation representations to have the estimation more effective, enabling accurate tracking even under occlusions and viewpoint shifts. By avoiding explicit reliance on depth or segmentation masks and focusing on visual alignment, DeepIM achieves strong generalization to novel poses and camera configurations. While effective in constrained settings, its reliance on known object geometry and synthetic renderings highlights the limits of generalization when semantic or deformable understanding is needed.

In Chapter 3, we present **STOW**, a system for discrete-frame segmentation and tracking of multiple objects in cluttered scenes. This work moves up to the scene level and focuses on the challenge of recognizing and tracking novel object instances over time. STOW is trained entirely in simulation using synthetic household objects and achieves real-world deployment in a warehouse picking robot. The system decouples the learning of visual recognition from tracking logic, leveraging mask-based representations to improve instance correspondence across frames. By designing the pipeline to generalize from synthetic to real domains, STOW demonstrates that structure-aware training and object-centric representations can reduce the domain gap without requiring expensive real-world robot data. However, the system's reliance on per-frame object masks still constrains its ability to reason semantically or infer high-level task goals.

In Chapter 4, we introduce **HAMSTER**, a hierarchical vision-language-action framework designed to operate at the task level. HAMSTER predicts short 2D keypoint trajectories in image space—"sketches" of what a robot should do—which serve as an intermediate abstraction between high-level vision-language models and low-level closed-loop controllers. These 2D trajectories are expressive enough to capture task-relevant spatial goals, yet simple enough to be predicted from off-domain training sources such as internet-scale vision-language data and human egocentric videos. Crucially, this abstraction allows the system to decouple semantic grounding from motor control: a vision-language model can predict the what and where of the task, while a separate controller learns how to execute the plan with minimal task-specific data. HAMSTER thus enables manipulation from diverse instructions and unseen configurations, bridging pretrained multimodal models with embodied robotic execution.

Together, these projects explore a path toward building generalist robot systems that are *data-scalable*, *cost-efficient*, and *generalizable*. By combining broad off-domain pretraining with an em-

bodied interface, we aim to reduce the marginal cost of real-world deployment and enable robots to operate in the diverse, unstructured environments that real users live and work in. Like in language and vision domains, we believe foundation models for robotics can also emerge through the right combination of architecture, data, and abstractions.

Chapter 2

DeepIM: Deep Iterative Matching for 6D Pose Estimation

Estimating 6D poses of objects from images is an important problem in various applications such as robot manipulation and virtual reality. While direct regression of images to object poses has limited accuracy, matching rendered images of an object against the input image can produce accurate results. In this chapter, we propose a novel deep neural network for 6D pose matching named DeepIM. Given an initial pose estimation, our network is able to iteratively refine the pose by matching the rendered image against the observed image. The network is trained to predict a relative pose transformation using a disentangled representation of 3D location and 3D orientation and an iterative training process. Experiments on commonly used benchmarks for 6D pose estimation demonstrate that DeepIM achieves large improvements over state-of-the-art methods. We further show that it is able to estimate and track the 6D pose of previously unseen objects.

2.1 Overview

Localizing objects in 3D from images is important in many real world applications. For instance, in a robot manipulation task, the ability to recognize the 6D pose of objects, i.e., 3D location and 3D orientation of objects, provides useful information for grasp and motion planning. In a virtual reality application, 6D object pose estimation enables virtual interactions between human and objects. While several recent techniques have used depth cameras for object pose estimation, such cameras have limitations with respect to frame rate, field of view, resolution, and depth range, making it very difficult to detect small, thin, transparent, or fast moving objects. Unfortunately, RGB-only 6D object pose estimation is still a challenging problem, since the appearance of objects in the images changes according to a number of factors, such as lighting, pose variations, and occlusions between objects. Furthermore, a robust 6D pose estimation method needs to handle both textured and textureless objects.

Traditionally, the 6D pose estimation problem has been tackled by matching local features extracted from an image to features in a 3D model of the object [Lowe, 1999; Rothganger et al., 2006;

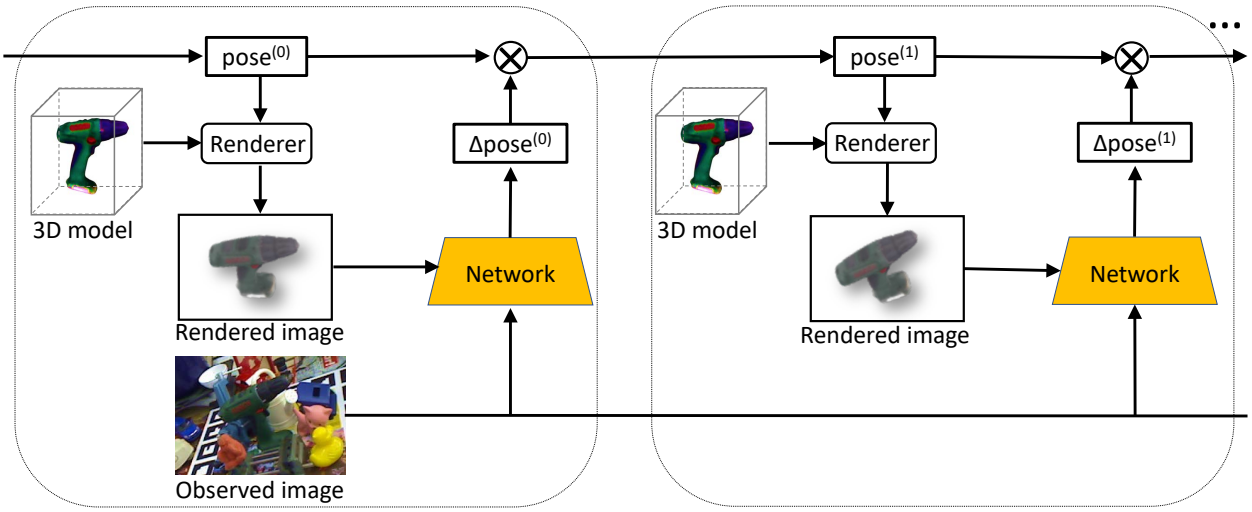


Figure 2.1: We propose DeepIM, a deep iterative matching network for 6D object pose estimation. The network is trained to predict a relative SE(3) transformation that can be applied to an initial pose estimation for iterative pose refinement. Given a 6D pose estimation of an object, which can be the output of other pose estimation methods like PoseCNN [Xiang et al., 2018] (pose$^{(0)}$ in the figure) or the refined pose from previous iteration (pose$^{(1)}$ in the figure), along with the 3D model of the object, we generate the rendered image showing the appearance of the target object under this rough pose estimation. With the image pairs of rendered image and observed image, the network predicts a relative transformation (■pose in the figure) which can be applied to refine the input pose. The refined pose can be used as the input pose of next iteration and therefore the process can be repeated until the refined pose converges or the number of iterations reaches a pre-determined number.

Collet et al., 2011]. By using the 2D-3D correspondences, the 6D pose of the object can be recovered. Unfortunately, such methods cannot handle textureless objects well since only few local features can be extracted for them. To handle textureless objects, two classes of approaches were proposed in the literature. Methods in the first class learn to estimate the 3D model coordinates of pixels or keypoints of the object in the input image. In this way, the 2D-3D correspondences are established for 6D pose estimation [Brachmann et al., 2014; Rad and Lepetit, 2017; Tekin et al., 2017]. Methods in the second class convert the 6D pose estimation problem into a pose classification problem by discretizing the pose space [Hinterstoisser et al., 2012a] or into a pose regression problem [Xiang et al., 2018]. These methods can deal with textureless objects, but they are not able to achieve highly accurate pose estimation, since small errors in the classification or regression stage directly lead to pose mismatches. A common way to improve the pose accuracy is pose refinement: Given an initial pose estimation, a synthetic RGB image can be rendered and used to match against the target input image. Then a new pose is computed to increase the matching score. Existing methods for pose refinement use either hand-crafted image features [Tjaden et al., 2017] or matching score functions [Rad and Lepetit, 2017].

In this chapter, we propose DeepIM, a new refinement technique based on a deep neural network for iterative 6D pose matching. Given an initial 6D pose estimation of an object in a test

image, DeepIM predicts a relative SE(3) transformation that matches a rendered view of the object against the observed image, or in other words, it predicts the relative rotation and translation that can refine the initial 6D pose estimation. By iteratively re-rendering the object based on the improved pose estimates, the two input images to the network become more and more similar, thereby enabling the network to generate more and more accurate pose estimates. Fig. 2.1 illustrates the iterative matching procedure of our network for pose refinement.

This work makes the following main contributions. i) We introduce a deep network for iterative, image-based pose refinement that does not require any hand-crafted image features and automatically learns an internal refinement mechanism. ii) We propose a disentangled representation of the SE(3) transformation between object poses to achieve accurate pose estimates. This representation also enables our approach to refine pose estimates of unseen objects. iii) We have conducted extensive experiments on the LINEMOD [Hinterstoisser et al., 2012a] and the Occlusion LINEMOD [Brachmann et al., 2014] datasets to evaluate the accuracy and various properties of DeepIM. These experiments show that our approach achieves large improvements over state-of-the-art RGB-only methods on both datasets. Furthermore, initial experiments demonstrate that DeepIM is able to accurately match poses for textureless objects (T-LESS [Hodan et al., 2017]) and for unseen objects [Wu et al., 2015]. The rest of the paper is organized as follows. After reviewing related works in Section 2, we describe our approach for pose matching in Section 3. Experiments are presented in Section 4, and Section 5 concludes the paper.

2.2 Related work

We review representative works on 6D pose estimation in the literature.

2.2.1 RGB based 6D Pose Estimation

Traditionally, object pose estimation using RGB images is tackled by matching local features [Lowe, 1999; Rothganger et al., 2006; Collet et al., 2011]. In this paradigm, a 3D model of an object is first reconstructed and local features of the object are attached to the 3D model. Keypoint-based features such as SIFT [Lowe, 1999] or SURF [Bay et al., 2008] are widely used. Given an input image, local features extracted from the image are matched against features on the 3D model. By filtering out incorrect matches using robust estimation techniques such as RANSAC [Nistér, 2005], the 6D pose of the object can be recovered using the 2D-to-3D correspondences between the local features. Local-feature matching based methods can handle partial occlusions between objects as long as the features on the visual part of the object are sufficient to determine the 6D pose. However, these methods cannot handle textureless objects well, since rich texture on the object is required in order to detect these features robustly.

In contrast, template-matching based methods are capable of handling textureless objects [Jurie and Dhome, 2001; Liu et al., 2010; Gu and Ren, 2010; Hinterstoisser et al., 2012b]. In this paradigm, templates of an object are first constructed, where examples of templates are renderings of the object from the 3D object model or Histogram of Oriented Gradients (HOG) [Dalal and Triggs, 2005] templates from different viewpoints. Then these templates are matched against the

input image to determine the location and orientation of the target object in the input image. The drawback of template-matching based methods is that they are not robust to occlusions between objects. When the target object is heavily occluded, the matching score is usually low which may result in incorrect pose estimation.

Recent approaches apply machine learning, especially deep learning, for 6D pose estimation using RGB images [Brachmann et al., 2014; Krull et al., 2015]. Learning techniques are employed to detect object keypoints for matching or learn better feature representations for pose estimation. The state-of-the-art methods [Rad and Lepetit, 2017; Kehl et al., 2017; Tekin et al., 2017; Xiang et al., 2018; Tremblay et al., 2018] augment deep learning based object detection or segmentation methods [Girshick, 2015; Long et al., 2015; Liu et al., 2016; Redmon et al., 2016] for 6D pose estimation. For example, [Rad and Lepetit, 2017; Tjaden et al., 2017; Tremblay et al., 2018] utilize deep neural networks to detect keypoints on the objects, and then compute the 6D pose by solving the PnP problem. [Kehl et al., 2017; Xiang et al., 2018] employ deep neural networks to detect objects in the input image, and then classify or regress the detected object to its pose. A recent work [Sundermeyer et al., 2018] uses an autoencoder to map the object in the image to a vector and search for the most similar vector in a pre-generated codebook for pose estimation. Overall, learning-based methods achieve better performance than traditional methods, largely due to the ability of learning a powerful feature representation for pose estimation.

2.2.2 Depth based 6D Pose Estimation

From another point of view, the 6D pose estimation problem can be tackled using depth images. Given a 3D model of an object and an input depth image, the problem is formulated as aligning the two point clouds computed from the 3D model and the depth image, respectively, which is also known as the geometric registration problem. Roughly speaking, geometric registration methods can be classified as local refinement methods and global registration methods. The most well-known local refinement algorithm is the Iterative Closest Point (ICP) algorithm [Besl and McKay, 1992] and its variants [Rusinkiewicz and Levoy, 2001; Salvi et al., 2007; Tam et al., 2013]. Given an initial pose estimation, the ICP algorithm iterates between finding the correspondences between points and refining the pose estimation using the new correspondences. In general, local refinement algorithms are sensitive to the initial pose. If the initial pose estimation is not close enough, the algorithm may converge to a local mimimum.

Global registration methods [Mellado et al., 2014; Theiler et al., 2015; Zhou et al., 2016; Yang et al., 2016] solve a more challenging problem by not assuming an initial pose estimate. A common strategy is to utilize iterative model fitting frameworks such as RANSAC. In each iteration, a set of point correspondences are sampled, and an alignment is computed and evaluated using the sampled correspondences. The limitation of most global registration methods is that they are computationally expensive. Also, the registration quality heavily depends on the quality of the 3D model and the scanned point cloud. In order to improve the registration performance, features on point clouds are also introduced for matching. These include point pairs [Mian et al., 2006; Hinterstoisser et al., 2016], spin-images [Johnson and Hebert, 1999], and point-pair histograms [Rusu et al., 2009; Tombari et al., 2010]. Similar to the trend in image-based matching, recent

approaches [Wang et al., 2019] propose to learn point features for registration, such as applying deep neural networks to point clouds [Qi et al., 2017].

2.2.3 RGB-D based 6D Pose Estimation

When both RGB images and depth images are available, they can be combined to improve 6D pose estimation. A common strategy is to estimate an initial pose of an object based on the color image, and then refine the pose using depth-based local refinement algorithms such as ICP [Hinterstoisser et al., 2012a; Michel et al., 2017; Zeng et al., 2017].

For example, Hinterstoisser et al. [2012a] renders the 3D model of an object into templates of color images, and then matches these templates against the input image to estimate an initial pose. The final pose estimation is obtained via ICP refinement on the initial pose. Brachmann et al. [2014], Brachmann et al. [2016], Michel et al. [2017] regress each pixel on the object in the input image to the 3D coordinate of that pixel on the 3D model. When depth images are available, the 3D coordinate regression establishes correspondences between 3D scene points and 3D model points, from which the 6D pose can be computed by solving a least-squares problem. PoseCNN [Xiang et al., 2018] introduces an end-to-end neural network for 6D object pose estimation using RGB images only. Given an initial pose from the network, a customized ICP method is applied to refine the pose. A recent work [Wang et al., 2019] introduces a neural network that combines RGB images and depth images for 6D pose estimation, and an iterative pose refinement network using point clouds as input.

2.2.4 RGB vs. RGB-D

Overall, the performance of RGB-based methods is still not comparable to that of the RGB-D based methods. We believe that this performance gap is largely due to the lack of an effective pose refinement procedure using RGB images only. Manhardt et al. [2018] which is published at the same time as ours introduces a method to refine 6D object poses with only RGB images, but there is still a large performance gap between Manhardt et al. [2018] and depth-based methods. Our work is complementary to existing 6D pose estimation methods by providing a novel iterative pose matching network for pose refinement on RGB images.

The approaches most related to ours are the object pose refinement network in Rad and Lepetit [2017] and the iterative hand pose estimation approaches in Carreira et al. [2016]; Oberweger et al. [2015]. Compared to these techniques, our network is designed to directly regress to relative SE(3) transformations. We are able to do this due to our disentangled representation of rotation and translation and the reference frame we used for rotation, which also allows our approach to match unseen objects. As shown in Mousavian et al. [2017], the choice of reference frame is important to achieve good pose estimation results. Our work is also related to recent visual servoing methods based on deep neural networks [Saxena et al., 2017; Costante and Ciarfuglia, 2018] that estimate the relative camera pose between two image frames, while we focus on 6D pose refinement of objects. Recent works [Garon et al., 2016; Garon and Lalonde, 2017] that focus on tracking could predict the transformation of the object pose between previous frame and current frame and have

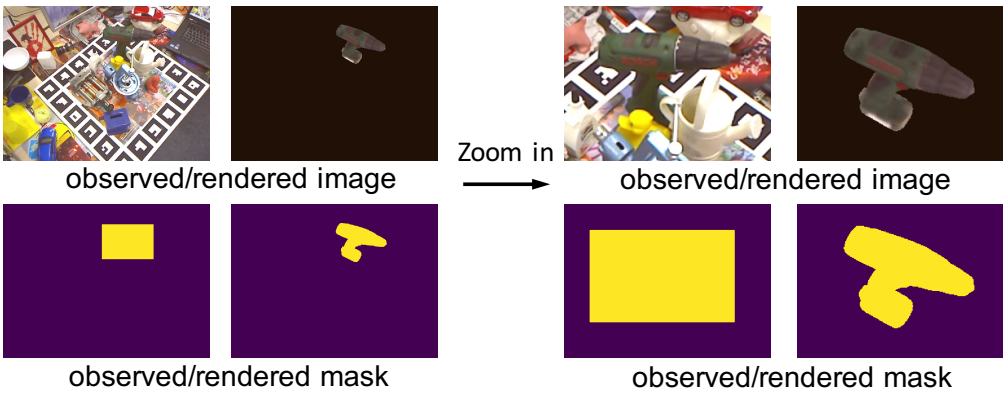


Figure 2.2: DeepIM operates on a zoomed in, up-sampled input image, the rendered image, and the two object masks (480×640 in our case after zooming in). More specifically, we enlarge the bounding box of the object in the rendered image, crop the corresponding patch using the enlarged bounding box in both image pairs and mask pairs and then up-sample them to high resolution. Notice that the aspect ratio is kept during this process to avoid image distortion. See Sec. 2.3.1 for more details.

the potential to be used for pose refinement.

2.3 DeepIM Framework

In this section, we describe our deep iterative matching network for 6D pose estimation. Given an observed image and an initial pose estimate of an object in the image, we design the network to directly output a relative SE(3) transformation that can be applied to the initial pose to improve the estimate. We first present our strategy of zooming in the observed image and the rendered image that are used as inputs of the network. Then we describe our network architecture for pose matching. After that, we introduce a disentangled representation of the relative SE(3) transformation and a new loss function for pose regression. Finally, we describe our procedure for training and testing the network.

2.3.1 High-resolution Zoom In

It can be difficult to extract useful features for matching if objects in the input image are very small. To obtain enough details for pose matching, we zoom in the observed image and the rendered image before feeding them into the network, as shown in Fig. 2.2. Specifically, in the i-th stage of the iterative matching, given a 6D pose estimate $\mathbf{p}^{(i-1)}$ from the previous step, we render a synthetic image using the 3D object model viewed according to $\mathbf{p}^{(i-1)}$.

We additionally generate one foreground mask for the observed image and rendered image. The four images are cropped using an enlarged bounding box according to the observed mask and the rendered mask, where we make sure the enlarged bounding box has the same aspect ratio as the input image and is centered at the 2D projection of the origin of the 3D object model.

In more detail, given the rendered mask $\mathbf{m}_{\text{rend}}$ and the observed mask $\mathbf{m}_{\text{obs}}$, the cropping patch is computed as

$$\begin{aligned} x_{\text{dist}} &= \max(|l_{\text{obs}} - x_c|, |l_{\text{rend}} - x_c|, \\ &\qquad |r_{\text{obs}} - x_c|, |r_{\text{rend}} - x_c|), \\ y_{\text{dist}} &= \max(|u_{\text{obs}} - y_c|, |u_{\text{rend}} - y_c|, \\ &\qquad |d_{\text{obs}} - y_c|, |d_{\text{rend}} - y_c|), \\ \text{width} &= \max(x_{\text{dist}}, y_{\text{dist}} \cdot r) \cdot 2\lambda, \\ \text{height} &= \max(x_{\text{dist}}/r, y_{\text{dist}}) \cdot 2\lambda, \end{aligned} \tag{2.1}$$

where u_*, d_*, l_*, r_* denotes the upper, lower, left, right bound of foreground mask of observed or rendered images, x_c, y_c represent the 2D projection of the center of the object in $\mathbf{img}_{\text{rend}}$, r represent the aspect ratio of the origin image (width/height), λ denotes the expand ratio, which is fixed to 1.4 in the experiment in order to make the expanded patch is roughly twice than the nested one. Then this patch is bilinearly sampled to the size of the original image, which is 480×640 in this paper. By doing so, not only the object is zoomed in without being distorted, but also the network is provided with the information about where the center of the object lies.

2.3.2 Network Structure

Fig. 2.3 illustrates the network architecture of DeepIM. The observed image, the rendered image, and the two masks, are concatenated into an eight-channel tensor input to the network (3 channels for observed/rendered image, 1 channel for each mask). We use the FlowNetSimple architecture from Dosovitskiy et al. [2015] as the backbone network, which is trained to predict optical flow between two images. We tried using the VGG16 image classification network [Simonyan and Zisserman, 2014] as the backbone network, but the results were very poor, confirming the intuition that a representation related to optical flow is very useful for pose matching [Wang et al., 2017].

The pose estimation branch takes the feature map after 10 convolution layers from FlowNetSimple as input. It contains two fully-connected layers each with dimension 256, followed by two additional fully-connected layers for predicting the quaternion of the 3D rotation and the 3D translation, respectively.

During training, we also add two auxiliary branches to regularize the feature representation of the network and increase training stability and performance, see Sec. 2.4.4 and Table. 2.2 for more details. One branch is trained for predicting optical flow between the rendered image and the observed image, and the other branch for predicting the foreground mask of the object in the observed image.

2.3.3 Disentangled Transformation Representation

The representation of the coordinate frames and the relative SE(3) transformation $\mathbf{\Delta p}$ between the current pose estimate and the target pose has important ramifications for the performance of the

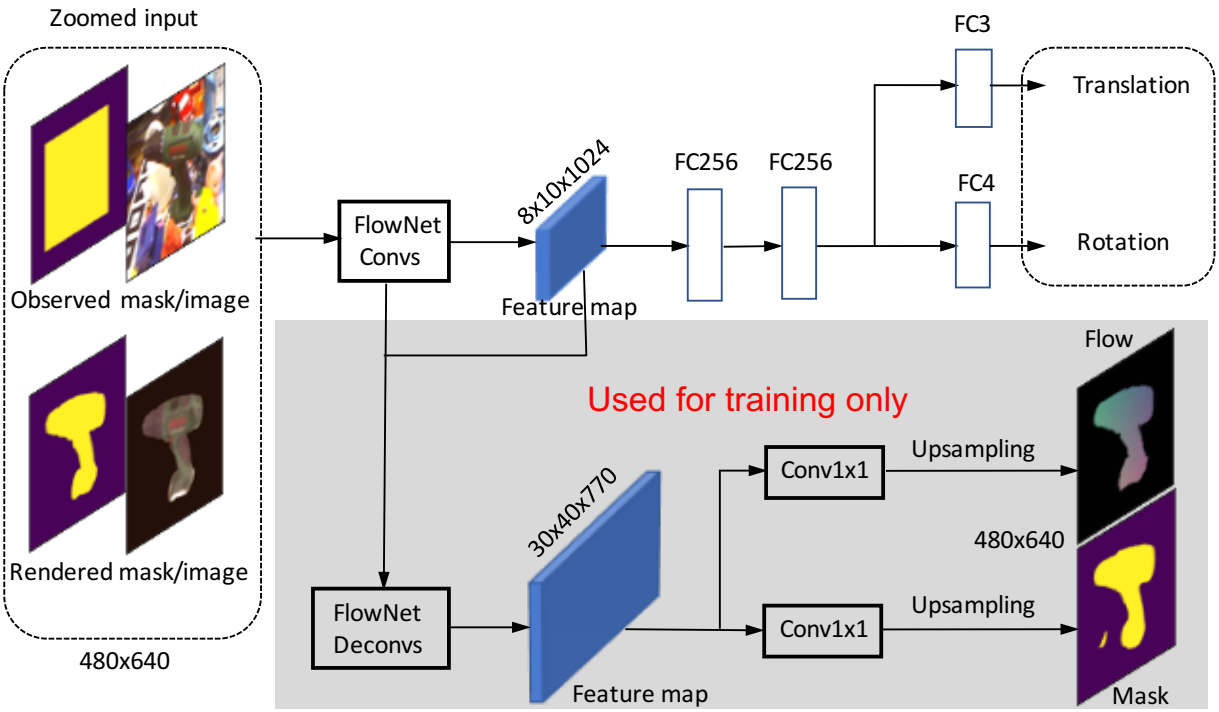


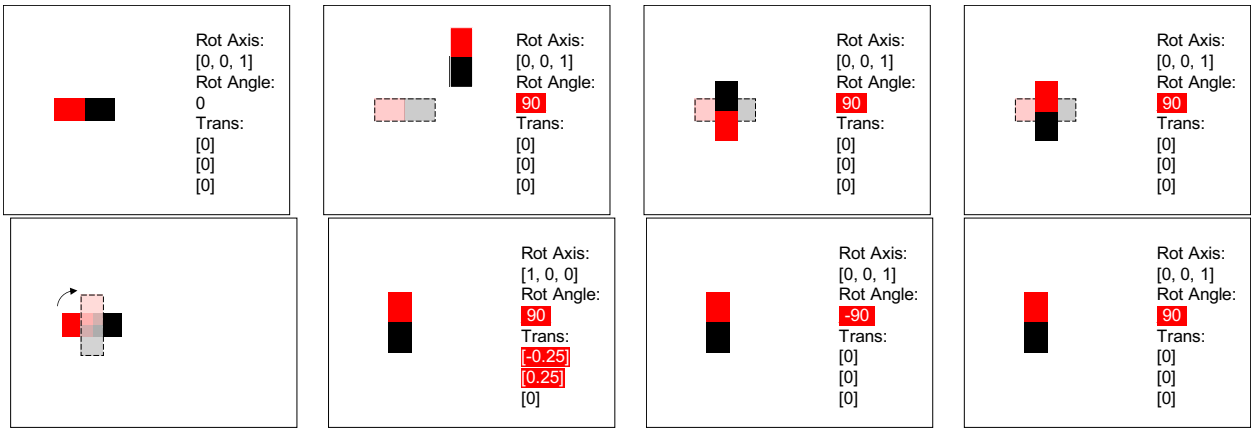
Figure 2.3: DeepIM uses a FlowNetSimple backbone to predict a relative SE(3) transformation to match the observed and rendered image of an object. Taking observed image and rendered image and their corresponding masks as input, the convolution layers output a feature map which then be forwarded through several fully connected layers to predict the translation and rotation. The same feature map, combined with feature maps in the previous layers, will also be used to predict flow and foreground mask during training.

network. Ideally, we would like (1) the individual components of these transformations to be maximally dis-entangled, thereby not requiring the network to learn unnecessarily complex geometric relationships between translations and rotations, and (2) the transformations to be independent of the intrinsic camera parameters and the actual size and coordinate system of an object, thereby enabling the network to reason about changes in object appearance rather than accurate distance estimates.

The most obvious choice are camera coordinates to represent object poses and transformations. Denote the relative rotation and translation as $[\mathbf{R}_\Delta|\mathbf{t}_\Delta]$ (We denote $\mathbf{R}_*$ as rotation and and $\mathbf{t}_*$ as translation in this paper). Given a source object pose $[\mathbf{R}_{\text{src}}|\mathbf{t}_{\text{src}}]$, the transformed target pose would be as follows:

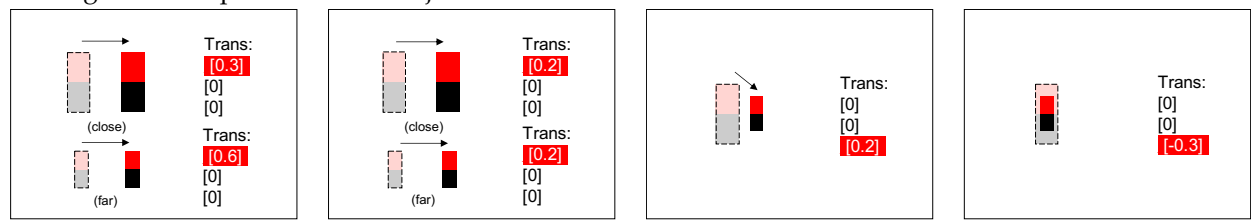
$$\mathbf{R}_{\text{tgt}} = \mathbf{R}_\blacksquare \mathbf{R}_{\text{src}}, \quad \mathbf{t}_{\text{tgt}} = \mathbf{R}_\blacksquare \mathbf{t}_{\text{src}} + \mathbf{t}_\blacksquare, \tag{2.2}$$

where $[\mathbf{R}_{\text{tgt}}|\mathbf{t}_{\text{tgt}}]$ denotes the target pose resulting from the transformation. The $\mathbf{R}_\blacksquare \mathbf{t}_{\text{src}}$ term indicates that a rotation will cause the object not only to rotate, but also translate in the image even if the translation vector $\mathbf{t}_\blacksquare$ equals to zero. Column (b) in Fig. 2.4 illustrates this connection for an



(a) Initial pose **(b)** Camera coord. **(c)** Model coord. **(d)** disentangled coord.

Figure 2.4: Rotations using different coordinate systems. (Upper row) The panels show how a 90 degree rotation in the image plane axis changes the position of the object shown in column (a). In the camera coordinate system, the center of rotation is in the center of the image, thereby causing an undesired translation in addition to the object rotation. In the model coordinate frame, as the frame of the object model can be defined arbitrarily, an object might rotate along any axis given the same rotation vector. Shown here is a CCW rotation, but the same axis might also result in an out of plane rotation for a differently defined object coordinate frame. In our disentangled representation, the center of rotation is in the center of the object and the axes are defined parallel to the camera axes. As a result, a rotation around a specific axis always results in the same object rotation, independent of the object. (Lower row) Rotation vectors a network would have to predict in order to achieve an in-place rotation using the different coordinate systems. Notice the extra translations required to compensate for the translation caused by the rotation using camera coordinates (column b). In model coordinates, the network would have to learn the frame specified for the object model in order to determine the correct rotation axis and angle. In our disentangled representation, rotation axis and angle are independent of the object.



(a) Camera coord. xy-plane translation **(b)** Disentangled coord. xy-plane translation **(c)** Camera coord. z-axis translation **(d)** Disentangled coord. z-axis translation

Figure 2.5: Translations using camera and our disentangled representations. In camera coordinates, translations in the image plane are represented by vectors in 3D space. As a result, the same translation in the 2D image corresponds to different translation vectors depending on whether an object is close or far from the camera. In our disentangled representation, the value of x and y is only related to the 2D vector in the image-plane. Additionally, as shown in column (c), in the camera representation, a translation along the z-axis is not only difficult to infer from the image, but also causes a move relative to the center of the image. In our disentangled translation representation (column (d)), only the change of scale needs to be estimated, making it independent of other translations and the metric size and distance of the object.

object rotating in the image plane. In standard camera coordinates, the translation $\mathbf{t}_{\blacksquare}$ of an object is in the 3D metric space (meter, for instance), which couples object size with distance in the met-

ric space. This would require the network to memorize the actual size of each object in order to transform mis-matches in images to distance offsets. It is obvious that such a representation is not appropriate, particularly for matching unknown objects.

To eliminate these problems, we propose to decouple the estimation of $\mathbf{R}_{\blacksquare}$ and $\mathbf{t}_{\blacksquare}$. First, we move the center of rotation from the origin of the camera to the center of the object in the camera frame, given by the current pose estimate. In this representation, a rotation does not change the translation of the object in the camera frame. The remaining question is how to choose the directions of the rotational axes of the coordinate frame. One way is to use the axes as specified in the 3D object model. However, as illustrated in column (c) of Fig. 2.4, such a representation would require the network to learn and memorize the coordinate frames of each object, which makes training more difficult and cannot be generalized to pose matching of unseen objects. Thus, we propose to use axes parallel to the axes of the camera frame when computing the relative rotation. By doing so, the network can be trained to estimate the relative rotation independently of the coordinate frame of the 3D object model, as illustrated in column (d) in Fig. 2.4.

In order to estimate the relative translation, let $\mathbf{t}_{\text{tgt}} = (x_{\text{tgt}}, y_{\text{tgt}}, z_{\text{tgt}})$ and $\mathbf{t}_{\text{src}} = (x_{\text{src}}, y_{\text{src}}, z_{\text{src}})$ be the target translation and the source translation. A straightforward way to represent translation is $\mathbf{t}_{\blacksquare} = (\blacksquare_x, \blacksquare_y, \blacksquare_z) = \mathbf{t}_{\text{tgt}} - \mathbf{t}_{\text{src}}$. However, it is not easy for the network to estimate the relative translation in the 3D metric space given only 2D images without depth information. The network has to recognize the size of the object, and map the translation in 2D space to 3D according to the object size. Such a representation is not only difficult for the network to learn, but also has problems when dealing with unknown objects or objects with similar appearance but different sizes. Instead of training the network to directly regress to the vector in the 3D space, we propose to regress to object changes in the 2D image space. Specifically, we train the network to regress to the relative translation $\mathbf{t}_{\blacksquare} = (v_x, v_y, v_z)$, where v_x and v_y denote the number of pixels the object should move along the image x-axis and y-axis and v_z is the scale change of the object:

$$\begin{aligned} v_x &= f_x(x_{\text{tgt}}/z_{\text{tgt}} - x_{\text{src}}/z_{\text{src}}), \\ v_y &= f_y(y_{\text{tgt}}/z_{\text{tgt}} - y_{\text{src}}/z_{\text{src}}), \\ v_z &= \log(z_{\text{src}}/z_{\text{tgt}}), \end{aligned} \tag{2.3}$$

where f_x and f_y denote the focal lengths of the camera. The scale change v_z is defined to be independent of the absolute object size or distance by using the ratio between the distances of the rendered and observed object. We use logarithm for v_z to make sure that a value of zero corresponds to no change in scale or distance. Considering the fact that f_x and f_y are constant for a specific dataset, we simply fix it to 1 in training and testing the network.

Our representation of the relative transformation has several advantages. First, rotation does not influence the estimation of translation, so that the translation no longer needs to offset the movement caused by rotation around the camera center. Second, the intermediate variables v_x, v_y, v_z represent simple translations and scale change in the image space. Third, this representation does not require any prior knowledge of the object. Using such a representation, the DeepIM network can operate independently of the actual size of the object, its internal model coordinate framework, and the camera intrinsics. It only has to learn to transform the rendered image such

that it becomes more similar to the observed image.

2.3.4 Matching Loss

A straightforward way to train the pose estimation network is to use separate loss functions for rotation and translation. For example, we can use the angular distance between two rotations to measure the rotation error and use the ℓ_2 distance to measure the translation error. However, using two different loss functions for rotation and translation suffers from the difficulty of balancing the two losses. [Kendall and Cipolla, 2017] proposed a geometric reprojection error as the loss function for pose regression that computes the average distance between the 2D projections of 3D points in the scene using the ground truth pose and the estimated pose. Considering the fact that we want to accurately predict the object pose in 3D, we introduce a modified version of the geometric reprojection loss in [Kendall and Cipolla, 2017], and we call it the Point Matching Loss. Given the ground truth pose $\mathbf{p} = [\mathbf{R}|\mathbf{t}]$ and the estimated pose $\hat{\mathbf{p}} = [\hat{\mathbf{R}}|\hat{\mathbf{t}}]$, the point matching loss is computed as:

$$L_{\text{pose}}(\mathbf{p}, \hat{\mathbf{p}}) = \frac{1}{n} \sum_{i=1}^{n} \|(\mathbf{R}\mathbf{x}_i + \mathbf{t}) - (\hat{\mathbf{R}}\mathbf{x}_i + \hat{\mathbf{t}})\|_1, \tag{2.4}$$

where $\mathbf{x}_i$ denotes a randomly selected 3D point on the object model and n is the total number of points (we choose 3,000 points in our experiments). The formulation of point matching loss is similar to the one used to compute average distance (ADD) metric in Eq. 2.5. The main difference is that other than using ℓ_2 norm, point matching loss computes the average ℓ_1 distance between 3D points transformed by the ground truth pose and the estimated pose in order to avoid the large graident caused by outliers and maintain the stability of loss during training. In this way, it measures how the transformed 3D models match against each other for pose estimation. [Xiang et al., 2018] also uses a variant of the point matching loss for rotation regression.

2.3.5 Training and Testing

In training, we assume that we have 3D object models and images annotated with ground truth 6D object poses. By adding noises to the ground truth poses as the initial poses, we can generate the required observed and rendered inputs to the network along with the pose target output that is the pose difference between the ground truth pose and the noisy pose. Then we can train the network to predict the relative transformation between the initial pose and the target pose.

During testing, we find that the iterative pose refinement can significantly improve the accuracy. To see, let $\mathbf{p}^{(i)}$ be the pose estimate after the i-th iteration of the network. If the initial pose estimate $\mathbf{p}^{(0)}$ is relatively far from the correct pose, the rendered image $\mathbf{img}_{\text{rend}}(\mathbf{p}^{(0)})$ may have only little viewpoint overlap with the observed image $\mathbf{img}_{\text{obs}}$. In such cases, it is very difficult to accurately estimate the relative pose transformation $\mathbf{\Delta p}^{(0)}$ directly. This task is even harder if the network has no priori knowledge about the object to be matched. In general, it is reasonable to assume that if the network improves the pose estimate $\mathbf{p}^{(i+1)}$ by updating $\mathbf{p}^{(i)}$ with $\mathbf{\Delta p}^{(i)}$ in the i-th iteration, then the image rendered according to this new estimate, $\mathbf{img}_{\text{rend}}(\mathbf{p}^{(i+1)})$ is also more

similar to the observed image $\mathbf{img}_{\text{obs}}$ than $\mathbf{img}_{\text{rend}}(\mathbf{p}^{(i)})$ was in the previous iteration, thereby providing input that can be matched more accurately.

However, we found that, if we train the network to regress the relative pose in a single step, the estimates of the trained network do not improve over multiple iterations in testing. To generate a more realistic data distribution for training similar to testing, we perform multiple iterations during training as well. Specifically, for each training image and pose, we apply the transformation predicted from the network to the pose and use the transformed pose estimate as another training example for the network in the next iteration. By repeating this process multiple times, the training data better represents the test distribution and the trained network also achieves significantly better results during iterative testing (such an approach has also proven useful for iterative hand pose matching [Oberweger et al., 2015] and image alignment [Lin and Lucey, 2017]).

2.4 Experiments

We conduct extensive experiments on the LINEMOD dataset [Hinterstoisser et al., 2012a] and the Occlusion LINEMOD dataset [Brachmann et al., 2014] to evaluate our DeepIM framework for 6D object pose estimation. We test different properties of DeepIM and show that it surpasses other RGB-only methods by a large margin. We also show that our network can be applied to pose matching of unseen objects during training.

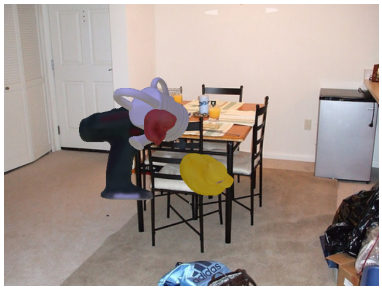
2.4.1 Training Implementation Details

Training Parameters: We use the pre-trained FlowNetSimple [Dosovitskiy et al., 2015] to initialize the weights in our network. Weights of the new layers are randomly initialized, except for the additional weights in the first conv layer that deals with the input masks and the fully-connected layer that predicts the translation, which are initialized with zeros. Other than predicting the pose transformation, the network also predicts the optical flow and the foreground mask. Including the two additional losses could slightly increase the pose estimation performance and make the training more stable. Specifically, we use the optical flow loss L_{flow} as in FlowNet [Dosovitskiy et al., 2015] and the sigmoid cross-entropy loss as the mask loss L_{mask}. Two deconvolutional blocks in FlowNet are inherited to produce the feature map used for the mask and the optical flow prediction, whose spatial scale is 0.0625. Two 1×1 convolutional layers with output channel 1 (mask prediction) and 2 (flow prediction) are appended after this feature map. The predictions are then bilinearly up-sampled to the original image size (480×640) to compute losses.

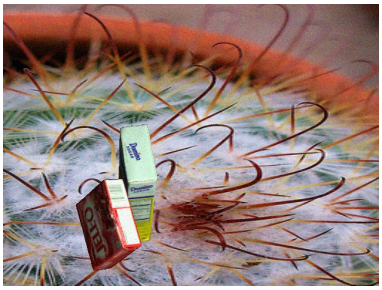
The overall loss is $L = \alpha L_{\text{pose}} + \beta L_{\text{flow}} + \gamma L_{\text{mask}}$, where we use $\alpha = 0.1$, $\beta = 0.25$, $\gamma = 0.03$ throughout the experiments (except some of our ablation studies). Each training batch contains 16 images. We train the network with 4 GPUs where each GPU processes 4 images. We generate 4 items for each image as described in Sec. 2.3.1: two images and two masks. The observed mask is randomly dilated with no more than 10 pixels to avoid over-fitting.

The Distribution of Rendered Pose during Training: The rendered image $\mathbf{img}_{\text{rend}}$ and mask $\mathbf{m}_{\text{rend}}$ are randomly generated during training without using prior knowledge of the initial poses

(a) Synthetic Data for LINEMOD



(b) Synthetic Data for Occlusion LINEMOD



(c) Synthetic Data for YCB-Video

Figure 2.6: Synthetic Data for the LINEMOD, Occlusion LINEMOD and YCB-Video separately. 2.6a shows the synthetic training data used when training on the LINEMOD dataset, only one object is presented in the image so there is no occlusion. 2.6b shows the synthetic training data used when training on the Occlusion LINEMOD dataset, multiple objects are presented in one image so one object may be occluded by other objects. 2.6c shows the synthetic training data used when training on the YCB-Video dataset. These images are rendered on the fly, so we only render two objects to maintain efficiency.

in the test set. Specifically, given a ground truth pose $\hat{\mathbf{p}}$, we add noises to $\hat{\mathbf{p}}$ to generate the rendered poses. For rotation, we independently add a Gaussian noise $\mathcal{N}(0, 15^2)$ to each of the three Euler angles of the rotation. If the angular distance between the new pose and the ground truth pose is more than 45°, we discard the new pose and generate another one in order to make sure the initial pose for refinement is within 45° of the ground truth pose during training. For translation, considering the fact that RGB-based pose estimation methods usually have larger standard deviation on depth estimation, the following Gaussian noises are added to the three components of the translation: $\Delta x \sim \mathcal{N}(0, 0.01^2), \Delta y \sim \mathcal{N}(0, 0.01^2), \Delta z \sim \mathcal{N}(0, 0.05^2)$, where the standard deviations are 1 cm, 1 cm and 5 cm, respectively.

Synthetic Training Data: Real training images provided in existing datasets may be highly correlated or lack images in certain situations such as occlusions between objects. Therefore, generating synthetic training data is essential to enable the network to deal with different scenarios in testing. In generating synthetic training data for the LINEMOD dataset, considering the fact that the elevation variation is limited in this dataset, we calculate the elevation range of the objects in the provided training data. Then we rotate the object model with a randomly generated quaternion and repeat it until the elevation is within this range. The translation is randomly generated using the mean and the standard deviation computed from the training set. During training, the background of the synthetic image is replaced by a randomly chosen indoor image from the PASCAL VOC dataset as shown in Fig. 2.6.

For the Occlusion LINEMOD dataset, multiple objects are rendered into one image in order to introduce occlusions among objects. The number of objects ranges from 3 to 8 in these synthetic images. As in the LINEMOD dataset, the quaternion of each object is also randomly generated to ensure that the elevation range is within that of training data in the Occlusion LINEMOD dataset. The translations of the objects in the same image are drawn according to the distributions of the

objects in the YCB-Video dataset [Xiang et al., 2018] by adding a small Gaussian noise.

For the YCB-Video dataset, synthetic images are generated on the fly. Other than the target object, we also render another object close to it to introduce partial occlusion.

The real training images may also lack variations in light conditions exhibited in the real world or in the testing set. Therefore, we add a random light condition to each synthetic image in both the LINEMOD dataset and the Occlusion LINEMOD dataset.

2.4.2 Testing Implementation Details

Testing Parameters: The mask prediction branch and the optical flow branch are removed during testing. Since there is no ground truth segmentation of the object in testing, we use the tightest bounding box of the rendered mask $\mathbf{m}_{\text{rend}}$ instead, so the network searches the neighborhood near the estimated pose to find the target object to match. Unless specified, we use the pose estimates from PoseCNN [Xiang et al., 2018] as the initial poses. Our DeepIM network runs at 12 fps per object using an NVIDIA 1080 Ti GPU with 2 iterations during testing.

Pose Initialization during inference: Our framework takes an input image and an initial pose estimation of an object in the image as inputs, and then refine the initial pose iteratively. In our experiments, we have tested two pose initialization methods.

The first one is PoseCNN [Xiang et al., 2018], a convolutional neural network designed for 6D object pose estimation. PoseCNN performs three tasks for 6D pose estimation, i.e., semantic labeling to classify image pixels into object classes, localizing the center of the object on the image to estimate the 3D translation of the object, and 3D rotation regression. In our experiments, we use the 6D poses from PoseCNN as initial poses for pose refinement.

To demonstrate the robustness of our framework on pose initialization, we have implemented a simple 6D pose estimation method for pose initialization, where we extend the Faster R-CNN framework designed for 2D object detection [Ren et al., 2015] to 6D pose estimation. Specifically, we use the bounding box of the object from Faster R-CNN to estimate the 3D translation of the object. The center of the bounding box is treated as the center of the object. The distance of the object is estimated by maximizing the overlap of the projection of the 3D object model with the bounding box. To estimate the 3D rotation of the object, we add a rotation regression branch to Faster R-CNN as in PoseCNN. In this way, we can obtain a 6D pose estimation for each detected object from Faster R-CNN.

In our experiments on the LINEMOD dataset described in Sec. 2.4.4, we have shown that, although the initial poses from Faster R-CNN are much worse than the poses from PoseCNN, our framework is still able to refine these poses using the *same* weights. The performance gap between using the two different pose initialization methods is quite small, which demonstrates the ability of our framework in using different methods for pose initialization.

2.4.3 Evaluation Metrics

We use the following three evaluation metrics for 6D object pose estimation. i) The *5°, 5cm* metric considers an estimated pose to be correct if its rotation error is within 5° and the translation error is below 5cm. ii) The *6D Pose* metric [Hinterstoisser et al., 2012a] computes the average distance between the 3D model points transformed using the estimated pose and the ground truth pose. For symmetric objects, we use the closest point distance in computing the average distance. An estimated pose is correct if the average distance is within 10% of the 3D model diameter. iii) The *2D Projection* metric computes the average distance of the 3D model points projected onto the image using the estimated pose and the ground truth pose. An estimated pose is correct if the average distance is smaller than 5 pixels.

k°, k cm: Proposed in Shotton et al. [2013]. The 5°, 5cm metric considers an estimated pose to be correct if its rotation error is within 5° and the translation error is below 5cm. We also provided the results with 2°, 2cm and 10°, 10cm in Table 2.6 to give a comprehensive view about the performance.

For symmetric objects such as eggbox and glue in the LINEMOD dataset, we compute the rotation error and the translation error against all possible ground truth poses with respect to symmetry and accept the result when it matches one of these ground truth poses.

6D Pose: Hinterstoisser et al. [2012a] use the average distance (ADD) metric to compute the averaged distance between points transformed using the estimated pose and the ground truth pose as in Eq. 2.5:

$$\textbf{ADD} = \frac{1}{m} \sum_{x \in \mathcal{M}} \|(\mathbf{R}\mathbf{x} + \mathbf{t}) - (\hat{\mathbf{R}}\mathbf{x} + \hat{\mathbf{t}})\|_2, \tag{2.5}$$

where m is the number of points on the 3D object model, $\mathcal{M}$ is the set of all 3D points of this model, $\mathbf{p} = [\mathbf{R}|\mathbf{t}]$ is the ground truth pose and $\hat{\mathbf{p}} = [\hat{\mathbf{R}}|\hat{\mathbf{t}}]$ is the estimated pose. Here the number of points m can be different from the number of points n used in Eq. 2.4 as the point clouds used for training is a subset randomly sampled from the original point clouds to reduce the time to compute the loss during training. $\mathbf{R}\mathbf{x} + \mathbf{t}$ indicates transforming the point with the given SE(3) transformation (pose) $\mathbf{p}$. Following [Brachmann et al., 2016], we compute the distance between all pairs of points from the model and regard the maximum distance as the diameter d of this model. Then a pose estimation is considered to be correct if the computed average distance is within 10% of the model diameter. In addition to using $0.1d$ as the threshold, we also provided pose estimation accuracy using thresholds $0.02d$ and $0.05d$ in Table 2.6. We use $0.1d$ as the threshold of 6D Pose metric in the following paper if not specified.

For symmetric objects, we use the closest point distance in computing the average distance for 6D pose evaluation as in Hinterstoisser et al. [2012a]:

$$\textbf{ADD-S} = \frac{1}{m} \sum_{\mathbf{x}_1 \in \mathcal{M}} \min_{\mathbf{x}_2 \in \mathcal{M}} \|(\mathbf{R}\mathbf{x}_1 + \mathbf{t}) - (\hat{\mathbf{R}}\mathbf{x}_2 + \hat{\mathbf{t}})\|_2. \tag{2.6}$$

In the YCB-Video Dataset, we use the metric ADD and ADD-S described in Xiang et al. [2018]. After getting the ADD and ADD-S distance described in Eq. 2.5 and Eq. 2.6, we vary the threshold from 0 to 10 cm and accumulate the area under the accuracy curves.

2D Projection: focuses on the matching of pose estimation on 2D images. This metric is considered to be important for applications such as augmented reality. We compute the error using Eq. 2.7 and accept a pose estimation when the 2D projection error is smaller than a predefined threshold:

$$\textbf{Proj. 2D} = \frac{1}{m} \sum_{x \in \mathcal{M}} \|\mathbf{K}(\mathbf{R}\mathbf{x} + \mathbf{t}) - \mathbf{K}(\hat{\mathbf{R}}\mathbf{x} + \hat{\mathbf{t}})\|_2, \tag{2.7}$$

where $\mathbf{K}$ denotes the intrinsic parameter matrix of the camera and $\mathbf{K}(\mathbf{R}\mathbf{x} + \mathbf{t})$ indicates transforming a 3D point according to the SE(3) transformation and then projecting the transformed 3D point onto the image. In addition to using 5 pixels as the threshold, we also show our results with the thresholds 2 pixels and 10 pixels. We use 5 pixels as the threshold of Proj. 2D metric in the following paper if not specified.

For symmetric objects such as eggbox and glue in the LINEMOD dataset, we compute the 2D projection error against all possible ground truth poses and accept the result when it matches one of these ground truth poses.

2.4.4 Experiments on the LINEMOD Dataset

The LINEMOD dataset contains 15 objects. We train and test our method on 13 of them as other methods in the literature. We follow the procedure in [Brachmann et al., 2016] to split the dataset into the training and test sets, with around 200 images for each object in the training set and 1,000 images in the test set. Fig. 2.9 shows a subset of objects used in LINEMOD dataset. These objects are textureless and thus difficult for pose estimation methods using only local features.

Training strategy: For every image, we generate 10 random poses near the ground truth pose, resulting in 2,000 training samples for each object in the training set. Furthermore, we generate 10,000 synthetic images for each object where the pose distribution is similar to the real training set. For each synthetic image, we generate 1 random pose near its ground truth pose. Thus, we have a total of 12,000 training samples for each object in training. The background of a synthetic image is replaced with a randomly chosen indoor image from PASCAL VOC [Everingham et al., 2010]. We train the networks for 8 epochs with initial learning rate 0.0001. The learning rate is divided by 10 after the 4th and 6th epoch, respectively.

Ablation study on iterative training and testing: Table 2.1 shows the results that use different numbers of iterations during training and testing. The networks with $train\_iter = 1$ and $train\_iter = 2$ are trained with 32 and 16 epochs respectively to keep the total number of updates the same as $train\_iter = 4$. The table shows that without iterative training ($train\_iter = 1$), multiple iteration testing does not improve, potentially even making the results worse ($test\_iter = 4$).

Table 2.1: Ablation study of the number of iterations during training and testing.

train iter	init	1			2			4		
test iter		1	2	4	1	2	4	1	2	4
5cm 5°	19.4	57.4	58.8	54.6	76.3	86.2	86.7	70.2	83.7	85.2
6D Pose	62.7	77.9	79.0	76.1	83.1	88.7	89.1	80.9	87.6	88.6
Proj. 2D	70.2	92.4	92.6	89.7	96.1	97.8	97.6	94.6	97.4	97.5

We believe that the reason is due to the fact that the network is not trained with enough rendered poses close to their ground truth poses. The table also shows that one more iteration during training and testing already improves the results by a large margin. The network trained with 2 iterations and tested with 2 iterations is slightly better than the one trained with 4 iterations and tested with 4 iterations. This may be because the LINEMOD dataset is not sufficiently difficult to generate further improvements by using 3 or 4 iterations. Since it is not straightforward to determine how many iterations to use in each dataset, we use 4 iterations during training and testing in all other experiments.

Ablation study on the zoom in strategy, network structures, transformation representations, and loss functions: Table 2.3 summarizes the ablation studies on various aspects of DeepIM. The "zoom" column indicates whether the network uses full images as its input or zoomed in bounding boxes up-sampled to the original image size. Comparing rows 5 and 7 shows that the higher resolution achieved via zooming in provides very significant improvements.

"Regressor": We train the DeepIM network jointly over all objects, generating a pose transformation independent of the specific input object (labeled "shared" in "regressor" column). Alternatively, we could train a different 6D pose regressor for each individual object by using a separate fully connected layer for each object after the final FC256 layer shown in Fig. 2.3. This setting is labeled as "sep." in Table 2.3. Comparing rows 3 and 7 shows that both approaches provide nearly indistinguishable results. But the shared network provides some efficiency gains.

"Network": Similarly, instead of training a single network over all objects, we could train separate networks, one for each object as in Rad and Lepetit [2017]. Comparing row 1 to 7 shows that a single, shared network provides better results than individual ones, which indicates that training on multiple objects can help the network learn a more general representation for matching. We also present an ablation study of mask prediction and flow prediction in Table 2.2. It shows that when trained with these two auxiliary branches, the network could achieve the highest performance.

"Coordinate": This column investigates the impact of our choice of coordinate frame to reason about object transformations, as described in Fig. 2.4. The row labeled "camera" provides results when choosing the camera frame of reference as the representation for the object pose, rows labeled "model" move the center of rotation to the object model and choose the object model coordinate frame to reason about rotations, and the "disentangled" rows provide our disentangled approach of moving the center into the object model while keeping the camera coordinate frame

Table 2.2: Ablation study on the role of mask prediction and flow prediction branch. The networks are trained 5 times for each setting on the object ape of the LINEMOD dataset. The numbers denote mean ± standard deviation.

methods		5cm 5°	6D Pose	Proj. 2D
mask	flow			
✓	✓	93.9±0.7	82.5±1.7	98.2±0.3
✓		91.7±0.4	82.5±1.6	97.7±0.1
	✓	89.2±2.1	63.7±3.4	98.4±0.2
		89.6±0.8	72.3±1.1	98.1±0.1

Table 2.3: Ablation study on different design choices of the DeepIM network on the LINEMOD dataset.

Row	methods					5cm 5°	6D Pose	Proj. 2D
	zoom	regressor	network	coordinate	loss			
1	✓	-	sep.	disentangled	PM	83.3	87.6	96.2
2	✓	sep.	shared	model	PM	79.2	87.5	95.4
3	✓	sep.	shared	disentangled	PM	86.6	89.5	96.7
4		shared	shared	camera	PM	16.6	44.3	62.5
5		shared	shared	disentangled	PM	38.3	65.2	80.8
6	✓	shared	shared	disentangled	Dist	86.5	79.2	96.2
7	✓	shared	shared	disentangled	PM	85.2	88.6	97.5

for rotations. Comparing rows 2 and 3 shows that reasoning in the camera rotation frame provides slight improvements. Furthermore, it should be noted that only our "disentangled" approach is able to operate on unseen objects. Comparing rows 4 and 5 shows the large improvements our representation achieves over the common approach of reasoning fully in the camera frame of reference.

"Loss": The traditional loss for pose estimation is specified by the distance ("Dist") between the estimated and ground truth 6D pose coordinates, i.e., angular distance for rotation and euclidean distance for translation. Comparing rows 6 and 7 indicates that our point matching loss ("PM") provides significantly better results especially on the 6D pose metric, which is the most important measure for reasoning in 3D space.

Application to different initial pose estimation networks: Table 2.4 provides results when we initialize DeepIM with two different pose estimation networks. The first one is PoseCNN [Xiang et al., 2018], and the second one is a simple 6D pose estimation method based on Faster R-CNN [Ren et al., 2015]. Specifically, we use the bounding box of the object from Faster R-CNN to estimate the 3D translation of the object. The center of the bounding box is treated as the center of the object. The distance of the object is estimated by maximizing the overlap of the projection of the 3D object model with the bounding box. To estimate the 3D rotation of the object, we add a ro-

Table 2.4: Ablation study on two different methods for generating initial poses on the LINEMOD dataset.

method	PoseCNN	PoseCNN +OURS	Faster R-CNN	Faster R-CNN +OURS
5cm 5°	19.4	85.2	11.9	83.4
6D Pose	62.7	88.6	33.1	86.9
Proj. 2D	70.2	97.5	20.9	95.7

Table 2.5: Comparison with state-of-the-art methods on the LINEMOD dataset: Brachmann Brachmann et al. [2016], BB8 Rad and Lepetit [2017], SSD-6D Kehl et al. [2017], Tekin Tekin et al. [2017], and PoseCNN Xiang et al. [2018].

Methods	Brachmann	BB8 w/ ref.	SSD-6D w/ ref.	Tekin	PoseCNN	PoseCNN + Ours
5cm 5°	40.6	69.0	-	-	19.4	**85.2**
6D Pose	50.2	62.7	79	55.95	62.7	**88.6**
Proj. 2D	73.7	89.3	-	90.37	70.2	**97.5**

tation regression branch to Faster R-CNN as in PoseCNN. As we can see in Table 2.4, our network achieves very similar pose estimation accuracy even when initialized with the estimates from the extension of Faster R-CNN, which are not as accurate as those provided by PoseCNN [Xiang et al., 2018].

Comparison with the state-of-the-art 6D pose estimation methods: Table 2.5 shows the comparison with the best color-only techniques on the LINEMOD dataset. DeepIM achieves very significant improvements over all prior methods, even those that also deploy refinement steps (BB8 [Rad and Lepetit, 2017] and SSD-6D [Kehl et al., 2017]).

Detailed Results on the LINEMOD Dataset: Table 2.6 shows our detailed results on all the 13 objects in the LINEMOD dataset. The network is trained and tested with 4 iterations and 8 epochs. Initial poses are estimated by PoseCNN [Xiang et al., 2018].

2.4.5 Experiments on the Occlusion LINEMOD Dataset

The Occlusion LINEMOD dataset proposed in Brachmann et al. [2014] shares the same images used in the LINEMOD dataset [Hinterstoisser et al., 2012a], but annotated 8 objects in one video that are heavily blocked by other objects.

Training: For every real image, we generate 10 random poses as described in Sec. 2.4.4. Considering the fact that most of the training data lacks occlusions, we generated about 20,000 synthetic images with multiple objects in each image. By doing so, every object has around 12,000 images

Table 2.6: Results of using more detailed thresholds on the LINEMOD dataset

metric	(n°, n cm)			6D Pose			Projection 2D		
threshold	(2, 2)	(5, 5)	(10,10)	0.02d	0.05d	0.10d	2 px.	5 px.	10 px.
ape	37.7	90.4	98.0	14.3	48.6	77.0	92.2	98.4	99.6
benchvise	37.6	88.7	98.2	37.5	80.5	97.5	67.7	97.0	99.6
camera	56.1	95.8	99.2	30.9	74.0	93.5	86.3	98.9	99.7
can	58.0	92.8	99.0	41.4	84.3	96.5	98.6	99.7	99.8
cat	33.5	87.6	97.8	17.6	50.4	82.1	88.4	98.7	100.0
driller	49.4	92.9	99.1	35.7	79.2	95.0	64.2	96.1	99.4
duck	30.8	85.2	98.5	10.5	48.3	77.7	88.1	98.5	99.8
eggbox	32.1	63.9	94.5	34.7	77.8	97.1	53.4	96.2	99.6
glue	32.8	83.0	98.0	57.3	95.4	99.4	81.5	98.9	99.7
holepuncher	8.7	54.5	93.8	5.3	27.3	52.8	59.1	96.3	99.5
iron	47.5	92.7	99.3	47.9	86.3	98.3	67.4	97.2	99.9
lamp	47.5	90.9	98.4	45.3	86.8	97.5	60.0	94.2	99.0
phone	34.8	89.6	98.6	22.7	60.5	87.7	75.9	97.7	99.8
MEAN	39.0	85.2	97.9	30.9	69.2	88.6	75.6	97.5	99.7

which are partially occluded, and a total of 22,000 images for each object in training. We perform the same background replacement and training procedure as in the LINEMOD dataset.

Comparison with the state-of-the-art methods: The comparison between our method and other RGB-only methods is shown in Fig. 2.8. We only show the plots with accuracies on the 2D Projection metric because these are the only results reported in Rad and Lepetit [2017] and [Tekin et al., 2017] (results for eggbox and glue use a symmetric version of this accuracy). It can be seen that our method greatly improves the pose accuracy generated by PoseCNN and surpasses all other RGB-only methods by a large margin. It should be noted that BB8 [Rad and Lepetit, 2017] achieves the reported results only when using ground truth bounding boxes during testing. Our method is even competitive with the results that use depth information and ICP to refine the estimates of PoseCNN. Fig. 2.9 shows some pose refinement results from our method on the Occlusion LINEMOD dataset.

Detailed Results on the Occlusion LINEMOD Dataset: Table 2.7 shows our results on the Occlusion LINEMOD dataset. We can see that DeepIM can significantly improve the initial poses from PoseCNN. Notice that the diameter here is computed using the extents of the 3D model following the setting of [Xiang et al., 2018] and other RGB-D based methods. Some qualitative results are shown in Figure 2.7.

Table 2.7: Results on the Occlusion LINEMOD dataset. The network is trained and tested with 4 iterations.

metric	(5°, 5cm)		6D Pose		Projection 2D	
method	Init.	Refined	Init.	Refined	Init.	Refined
ape	2.3	**51.8**	9.9	**59.2**	34.6	**69.0**
can	4.1	**35.8**	45.5	**63.5**	15.1	**56.1**
cat	0.3	**12.8**	0.8	**26.2**	10.4	**50.9**
driller	2.5	**45.2**	41.6	**55.6**	7.4	**52.9**
duck	1.8	**22.5**	19.5	**52.4**	31.8	**60.5**
eggbox	0.0	**17.8**	24.5	**63.0**	1.9	**49.2**
glue	0.9	**42.7**	46.2	**71.7**	13.8	**52.9**
hole.	1.7	**18.8**	27.0	**52.5**	23.1	**61.2**
MEAN	1.7	**30.9**	26.9	**55.5**	17.2	**56.6**

2.4.6 Experiments on the YCB-Video Dataset

The YCB-Video Dataset, which is proposed in [Xiang et al., 2018], annotates 21 YCB objects [Calli et al., 2015] in 92 video sequences (133,827 frames). It is a challenging dataset as the objects have varied sizes (diameter from 10 cm to 40 cm), different types of symmetries, and a large variety of occlusions and lighting conditions. We split the dataset as [Xiang et al., 2018], with 80 video sequences for training and 2,949 keyframes in the remaining 12 videos for testing.

Training Strategy: As images in one video are similar to those in nearby frames, we use 1 image out of every 10 images in the training set for training. Training batches consist of captured real images from the dataset (1/8) and synthetic images which are partially occluded and generated on the fly (7/8). The network is trained with 8 epochs and we decrease the learning rate after 4 and 6 epochs. We found that with large training sets and enough epochs it was not necessary to include the flow prediction and the masks in the input, so we removed those branches and the corresponding loss from this experiment. For different categories, they share the same network but use separate regressors to achieve the best performance.

Evaluation Metric: We follow the PoseCNN [Xiang et al., 2018] paper when evaluating the results which uses accuracy under curve of ADD (Eq. 2.5) and ADD-S (Eq. 2.6 for each object. We also report the results of ADD(-S) and AUC ADD(-S) metric which is similar to the one we used in LINEMOD [Brachmann et al., 2014]. More specifically, we use ADD when the object is not symmetric and use ADD-S when the object is symmetric. Then we compute the averaged accuracy as the final result.

Symmetric Objects: As described in Sec. 2.4.1, we only keep rendered poses that have an angular distance less than 45 degrees from ground truth poses during training, which means we don't need to take special care of objects which have a symmetry angle of more than 90 degrees. How-

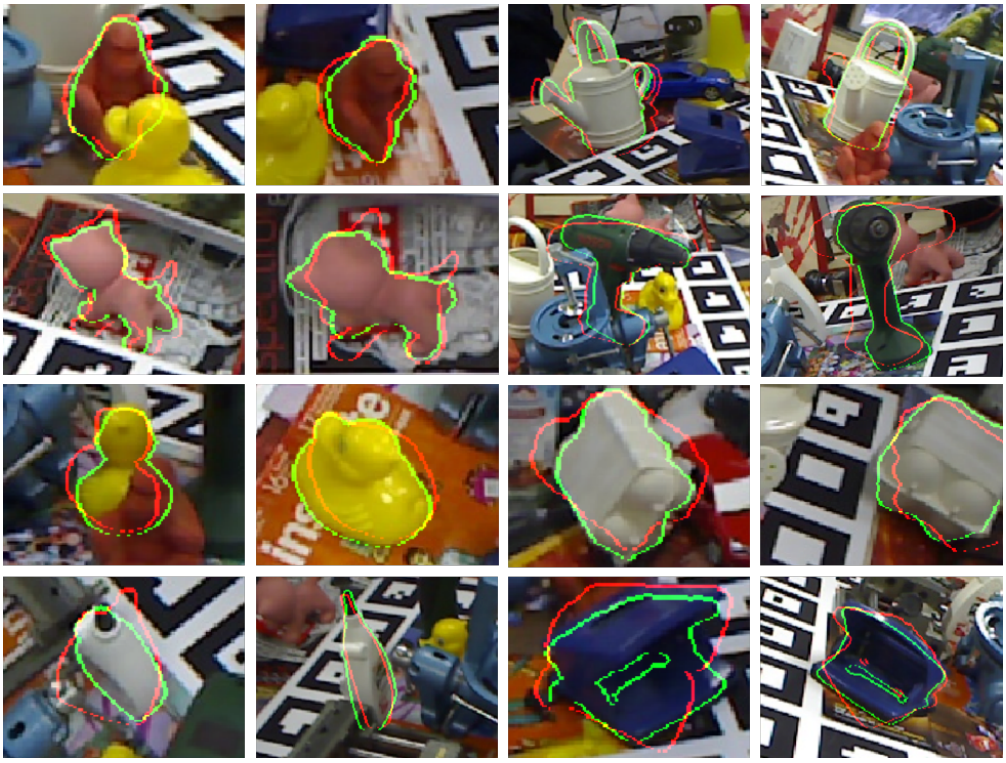


Figure 2.7: Some pose refinement results on the Occlusion LINEMOD dataset. The red and green lines represent the edges of 3D model projected from the initial poses and our refined poses respectively.

ever, object 024_bowl in the YCB-Video dataset is rotational symmetric. To deal with this kind of symmetry, rather than using the ground truth pose $\hat{\mathbf{p}}$ provided by the dataset to compute the loss, we choose the distance to the closest pose $\mathbf{p}^*$ among all poses that look the same as the ground truth pose:

$$\mathbf{p}^* = \arg\min_{\mathbf{p}\in\mathcal{Q}} \Theta(\mathbf{p}, \mathbf{p}_{\text{src}}) \tag{2.8}$$

Here, $\mathcal{Q}$ denotes the set of poses whose corresponding rendered images are the same as the one rendered using the ground truth pose. We assume that the rotation axis goes through the origin of the model frame so that no translation needs to be considered. In the experiment, we calibrate the rotation axis manually and use bisection search to locate the *closest ground truth pose*. Table. 2.8 compares networks trained with and without this strategy, showing that this training loss is useful.

Comparison with state-of-the-art methods: Table 2.10 compares our results with two state-of-the-art methods: PoseCNN [Xiang et al., 2018] and DenseFusion [Wang et al., 2019]. As can be seen, DeepIM greatly refines the initial pose provided by PoseCNN and is on par with those refined with ICP on many objects despite not using any depth or point cloud data. Notice that DeepIM produces low numbers on symmetric objects, like 024_bowl, under ADD metric. This is because the ADD metric cannot well represent the performance on symmetric objects as such objects have multiple correct poses but only one of these poses are labeled as the ground truth in

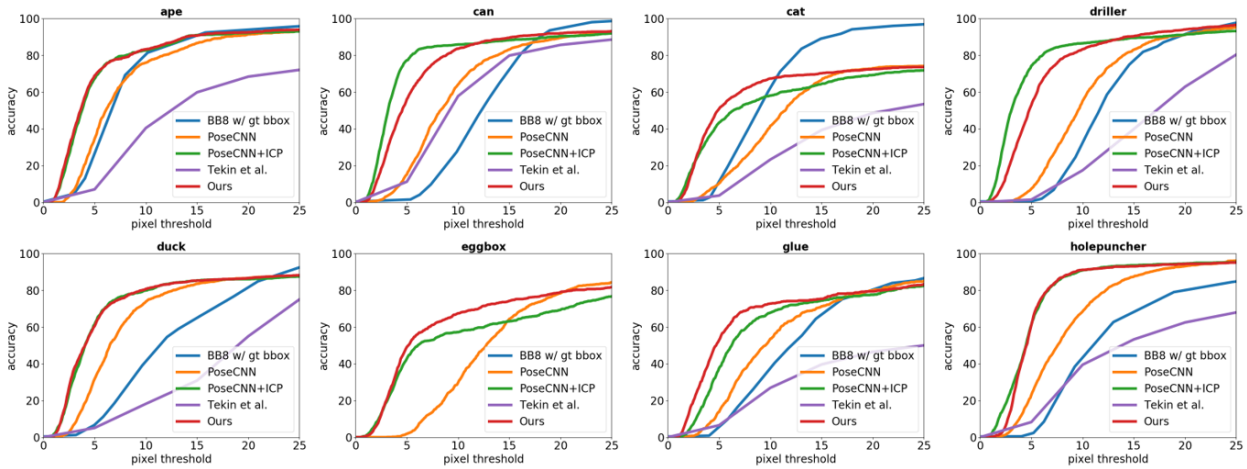


Figure 2.8: Comparison with state-of-the-art methods on the Occlusion LINEMOD dataset [Brachmann et al., 2014]. Accuracies are measured via the Projection 2D metric.

Figure 2.9: Examples of refined poses on the Occlusion LILNEMOD dataset using the results from PoseCNN [Xiang et al., 2018] as initial poses. The red and green lines represent the silhouettes of the initial estimates and our refined poses, respectively.

the dataset. Table 2.9 shows the result compared with PoseCNN [Xiang et al., 2018] and PoseRBPF [Deng et al., 2019] using the ADD(-S) metrci which can avoid such problems. Fig. 2.10 visualizes some pose refinement results from our method on the YCB-Video dataset.

Tracking in the YCB-Video Dataset: Considering the similarity between pose refinement and object tracking, it is natural to use DeepIM to track objects in videos. Therefore, we conducted an experiment testing DeepIM's ability to track objects in the YCB-Video dataset. Provided with the ground truth pose of an object in the first frame of each video, DeepIM can perform tracking by using the refined pose estimate from the previous frame as the initial pose of the next frame. Rather than doing inference only on key frames, we applied DeepIM to all images in the test video so that the object poses were close between successive frames.

In order to determine when DeepIM loses track of an object due to heavy occlusion, we follow

024_bowl	init	common	closest
ADD	54.2	55.6	68.4
ADD-S	76.0	70.6	80.9

Table 2.8: Ablation study about using *closest ground truth pose* to handle rotational symmetric objects. These three columns show the evaluation results of initial poses, poses refined by a DeepIM network that treats 024_bowl as a regular object, and poses refined by a network trained with *closest ground truth pose*. Initial poses are generated as rendered pose during training described in Sec. 2.4.1

Table 2.9: Overall results on YCB video results compared with PoseCNN [Xiang et al., 2018] and PoseRBPF [Deng et al., 2019]. The ADD(-S) metric and AUC ADD(-S) metric is introduced in Sec. 2.4.6

Methods	RGB				RGB-D		
	PoseCNN	PoseRBPF++	PoseCNN +DeepIM	DeepIM +Tracking	PoseCNN +ICP	PoseRBPF	PoseCNN +DeepIM
ADD(-S)<2cm	27.55	-	71.5	**79.0**	78.9	-	**90.3**
AUC of ADD(-S)	61.31	64.4	81.9	**85.9**	86.6	88.5	**90.4**

a simple strategy: we count the tracking as "lost" if the last iteration of the last 10 frames has an average rotation greater than 10 degrees or an average translation greater than 1 cm. Once the tracking is marked as lost, the network will be re-initialzed with PoseCNN's prediction. This strategy is designed with the intuition that successful tracking should have a small offset at the last iteration. Re-initialization happens every 340 frames on average. Table 2.9 and Table 2.10 shows our numerical results. Notice that the results of tracking are better than PoseCNN+DeepIM in most cases and are comparable to the results refined with ICP which uses depth information. Also note that the performance on object 036_wood_block is bad because the model of the wooden block is different from the object used in the actual dataset video, which makes it nearly impossible to match the model with the image.

Tracking YCB objects in real scenes: To demonstrate our framework's generalization, we use our network to track objects in real scenes. This means we don't have any prior knowledge about the lighting conditions, background, or camera parameters. Similar to tracking on the YCB-Video dataset, we use DeepIM to refine poses predicted from the previous frame. Thanks to the disentangled representation, we did not have to calibrate the camera to get its intrinsic matrix. Fig. 2.11 shows some tracking results using our method in the real world environment in real time.

Using Depth information: Other than using RGB images to do pose refinement, DeepIMcan be easily extended to utilize depth information to improve its performance. Here we append the depth images of the observed image and the rendered image with the two zero-initialized additional channels in the first convolution (one for the rendered depth and the other for the observed

Table 2.10: Detailed Results on the YCB-Video dataset compared with PoseCNN [Xiang et al., 2018] and DenseFusion [Wang et al., 2019]. The network is trained and tested with 4 iterations. The ADD and ADD-S is short for AUC of ADD and AUC of ADD-S.

Methods	RGB						RGB-D				
	PoseCNN		PoseCNN +DeepIM		DeepIM Tracking		PoseCNN +ICP		Dense Fusion	PoseCNN +DeepIM	
Evaluation Metric	ADD	ADD^S	ADD	ADD^S	ADD	ADD^S	ADD	ADD^S	ADD^S	ADD	ADD^S
002_master_chef_can	50.2	83.9	71.2	93.1	**89.0**	**93.8**	68.1	95.8	**96.4**	**78.0**	96.3
003_cracker_box	53.1	76.9	83.6	91.0	**88.5**	**93.0**	83.4	92.7	**95.5**	**91.4**	95.3
004_sugar_box	68.4	84.3	94.1	96.2	**94.3**	**96.3**	97.2	**98.2**	97.5	97.6	**98.2**
005_tomato_soup_can	66.2	81.0	86.1	92.4	**89.1**	**93.2**	81.8	94.5	94.6	**90.3**	**94.8**
006_mustard_bottle	81.0	90.4	91.5	**95.1**	**92.0**	**95.1**	**98.0**	98.6	97.2	97.1	**98.0**
007_tuna_fish_can	70.7	88.1	87.7	96.1	**92.0**	**96.4**	83.9	97.1	96.6	**92.2**	**98.0**
008_pudding_box	62.7	79.1	**82.7**	**90.7**	80.1	88.3	**96.6**	**97.9**	96.5	83.5	90.6
009_gelatin_box	75.2	87.2	91.9	94.3	**92.0**	**94.4**	**98.1**	**98.8**	98.1	98.0	98.5
010_potted_meat_can	59.5	78.5	76.2	86.4	**78.0**	**88.9**	**83.5**	**92.7**	91.3	82.2	90.3
011_banana	72.3	86.0	**81.2**	**91.3**	81.0	90.5	91.9	97.1	96.6	**94.9**	**97.6**
019_pitcher_base	53.3	77.0	90.1	94.6	**90.4**	**94.7**	96.9	97.8	97.1	**97.4**	**97.9**
021_bleach_cleanser	50.3	71.6	81.2	90.3	**81.7**	**90.5**	**92.5**	**96.9**	95.8	91.6	**96.9**
024_bowl	30.0	70.0	8.6	81.4	**38.8**	**90.6**	**47.6**	80.8	**88.2**	8.1	87.0
025_mug	58.5	78.2	81.4	91.3	**83.2**	**92.0**	81.1	95.0	97.1	**94.2**	**97.6**
035_power_drill	55.3	72.7	**85.5**	**92.3**	85.4	**92.3**	**97.7**	**98.2**	96.0	97.2	97.9
036_wood_block	26.6	64.3	**60.0**	**81.9**	44.3	75.4	70.9	87.6	89.7	**81.1**	**91.5**
037_scissors	35.8	56.9	60.9	75.4	**70.3**	**84.5**	78.4	91.7	95.2	**92.7**	**96.0**
040_large_marker	58.3	71.7	75.6	86.2	**80.4**	**91.2**	85.3	97.2	97.5	**88.9**	**98.2**
051_large_clamp	24.6	50.2	48.4	74.3	**73.9**	**84.1**	52.1	75.2	72.9	**54.2**	**77.9**
052_extra_large_clamp	16.1	44.1	31.0	73.3	**49.3**	**90.3**	26.5	64.4	69.8	**36.5**	**77.8**
061_foam_brick	72.9	88.2	35.9	81.9	**91.6**	**95.5**	**90.5**	97.4	92.5	48.2	**97.6**
MEAN	53.4	74.6	71.7	88.1	**79.3**	**91.0**	80.6	92.4	93.0	**80.7**	**94.0**

depth). To provide the network with information of the center of the object, we normalize the depth images by subtract them from the depth of the object's center. The results are shown in Table. 2.10.

2.4.7 Application to Unseen Objects and Unseen Categories

As stated in Sec. 2.3.3, we designed the disentangled pose representation such that it is independent of the coordinate frame and the size of a specific 3D object model. In other words, the transformation predicted from the network does not need to have prior knowledge about the model itself. Therefore, the pose transformations correspond to operations in the image space. This opens the question whether DeepIM can refine the poses of objects that are not included in the training set. From the experiment results we found that our network can perform accurate

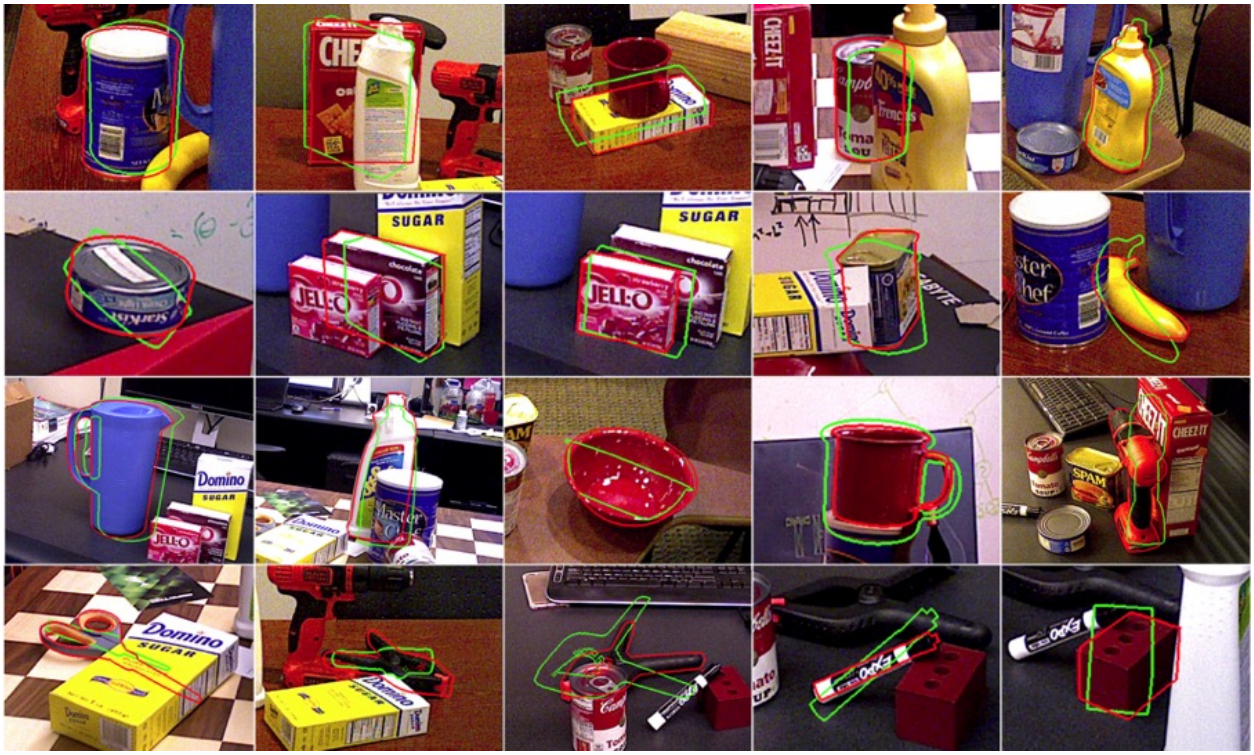


Figure 2.10: Examples of refined poses on the YCB-Video dataset which use results from PoseCNN [Xiang et al., 2018] as initial poses. The green and red lines represent the silhouettes of the initial estimates and our refined poses, respectively.

refinement on these unseen models. See Fig. 2.13 for example results. We also tested our framework on refining the poses of unseen object categories, where the training categories and the test categories are completely different.

Test on Unseen Objects: In this experiment, we explore the ability of the network in refining poses of objects that has never been seen in training. ModelNet [Wu et al., 2015] contains a large number of 3D models in different object categories. Here, we tested our network on three of them: *airplane, car and chair*. For each of these categories, we train a network on no more than 200 3D models and test its performance on 70 unseen 3D models from the same category. Similar to the way that we generate synthetic data as described in Sec 2.4.1, we generate 50 poses for each model

Table 2.11: Results on unseen objects. These models are not included in the training set.

category	airplane		car		chair	
method	Init.	Refined	Init.	Refined	Init.	Refined
5cm 5°	0.8	**68.9**	1.0	**81.5**	1.0	**87.6**
6D Pose	25.7	**94.7**	10.8	**90.7**	14.6	**97.4**
Proj. 2D	0.4	**87.3**	0.2	**83.9**	1.5	**88.6**

Figure 2.11: Examples on tracking in the real world, using the same network as in Table. 2.10 and no prior knowledge about focal length. The first row shows the images captured with a webcam and the second row renders the object onto the image based on the estimated pose.

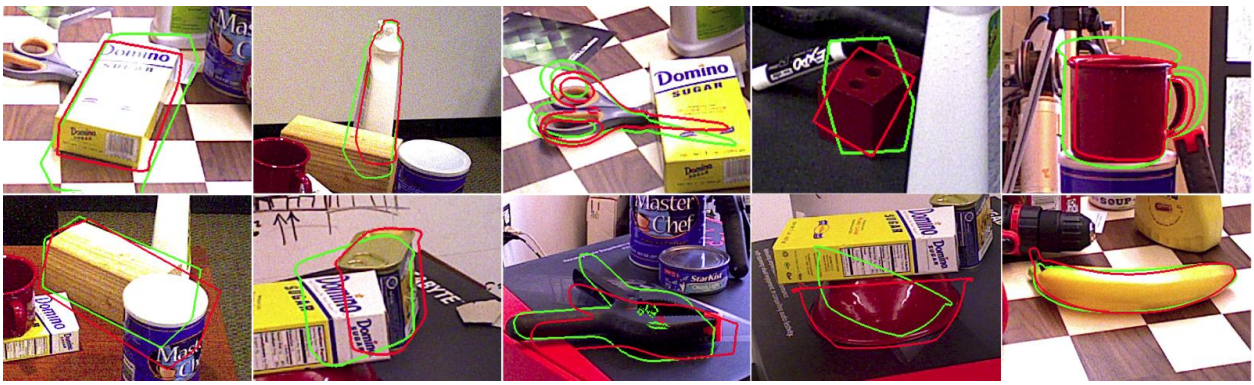


Figure 2.12: Failure cases in YCB-Video dataset. These images illustrate 5 different reasons we concluded that leads to fail cases.

as the target poses and train the network for 4 epochs. We use uniform gray texture for each model and add a light source which has a fixed relative position to the object to reflect the norms of the object. The initial pose used in training and testing is generated in the same way as we did in previous experiments as described in Sec. 2.4.1. The results are show in Table 2.11.

Test on Unseen Categories: We also tested our framework on refining the poses of unseen object categories, where the training categories and the test categories are completely different. We train the network on 8 categories from ModelNet [Wu et al., 2015]: *airplane, bed, bench, car, chair, piano, sink, toilet* with 30 models in each category and 50 image pairs for each model. The network was trained with 4 iterations and 4 epochs. Then we tested the network on 7 other categories: *bathtub, bookshelf, guitar, range hood, sofa, wardrobe, and tv stand*. The results are shown in Table. 2.12. It shows that the network indeed has learned some general features for pose refinement across different object categories.

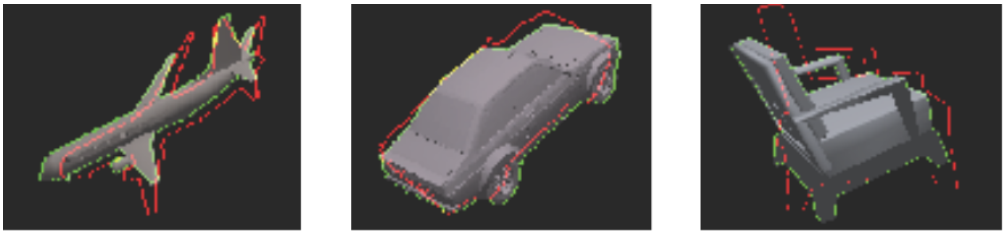


Figure 2.13: Results on pose refinement of 3D models from the ModelNet dataset. These instances were not seen in training. The red and green lines represent the edges of the initial estimates and our refined poses.

Table 2.12: Results on unseen categories. These categories has never been seen by the network during training.

metric	(5°, 5cm)		6D Pose		Projection 2D	
method	Init.	Refined	Init.	Refined	Init.	Refined
bathtub	0.9	**71.6**	11.9	**88.6**	0.2	**73.4**
bookshelf	1.2	**39.2**	9.2	**76.4**	0.1	**51.3**
guitar	1.2	**50.4**	9.6	**69.6**	0.2	**77.1**
range hood	1.0	**69.8**	11.2	**89.6**	0.0	**70.6**
sofa	1.2	**82.7**	9.0	**89.5**	0.1	**94.2**
wardrobe	1.4	**62.7**	12.5	**79.4**	0.2	**70.0**
tv stand	1.2	**73.6**	8.8	**92.1**	0.2	**76.6**

2.4.8 Limitations

Even with the necessity of CAD models of the object and either real or synthetic data for network training, there are still number of scenarios where it may not succeed. In Fig. 2.12 we show 10 instances that the network fails to refine to a correct pose. They can be grouped into 5 categories: 1) discrepancy between object models and images. This can be caused by bad light conditions or an inaccurate object model; 2) few patterns to match. This usually happens when only certain featureless side-views are visible or the object is heavily occluded; 3) objects' shapes are unusu al and difficult to learn; 4) the initial pose is too far away from the correct pose; 5) objects with tiny key components.

2.5 Discussion

In this chapter we introduce DeepIM, a novel framework for iterative pose matching using color images only. Given an initial 6D pose estimation of an object, we have designed a new deep neural network to directly output a *relative* pose transformation that improves the pose estimate. The network automatically learns to match object poses during training. We introduce an disentangled pose representation that is also independent of the object size and the coordinate frame of the 3D object model. In this way, the network can even match poses of unseen objects, as shown in our

experiments. Our method significantly outperforms state-of-the-art 6D pose estimation methods using color images only and provides performance close to methods that use depth images for pose refinement, such as using the iterative closest point algorithm.

However, the reliance on object CAD models restricts the applicability of this approach. Furthermore, with advancements in robot manipulation, the necessity and sufficiency of the 6D object pose for manipulation tasks are being questioned. Many approaches Mousavian et al. [2019]; Sundermeyer et al. [2021] do not require the 6D pose to determine affordance. Additionally, the 6D pose falls short in representing or solving tasks involving deformable or articulated objects. Therefore, in the next chapter, we explore alternative methods to guide robot manipulation more broadly.

Chapter 3

STOW: Discrete-Frame Segmentation and Tracking of Unseen Objects for Warehouse Picking Robots

Segmentation and tracking of unseen object instances in discrete frames pose a significant challenge in dynamic industrial robotic contexts, such as distribution warehouses. Here, robots must handle object rearrangement, including shifting, removal, and partial occlusion by new items, and track these items after substantial temporal gaps. The task is further complicated when robots encounter objects not learned in their training sets, which requires the ability to segment and track previously unseen items. Considering that continuous observation is often inaccessible in such settings, our task involves working with a discrete set of frames separated by indefinite periods during which substantial changes to the scene may occur. This task also translates to domestic robotic applications, such as rearrangement of objects on a table. In this chapter, in order to address these demanding challenges, we introduce new synthetic and real-world datasets that replicate these industrial and household scenarios. We also propose a novel paradigm for joint segmentation and tracking in discrete frames along with a transformer module that facilitates efficient inter-frame communication. The experiments we conduct show that our approach significantly outperforms recent methods. For additional code, dataset, results and videos, please visit website.

3.1 Overview

Object segmentation and tracking, a key perception task for robotic picking, is particularly important in warehouse environments, where millions of commodity items are organized daily on warehouse shelves for storage and categorization, as shown in Figure 3.1. Future intelligent robots must acquire strong perception capabilities to help human workers stow and fetch items from these shelves. These capabilities include detecting objects with diverse geometries in cluttered scenes and tracking them while other items are being added or picked up. Despite considerable

research progress in this area, it remains notoriously difficult to detect and track unknown objects in highly cluttered environments.

Researchers commonly address this task using a *segment-then-track* paradigm Goyal et al. [2022]; Lu et al. [2023], which executes the two procedures sequentially. During segmentation, each frame is handled as an independent image on which advanced unseen object instance segmentation methods are applied Xie et al. [2021]; Xiang et al. [2021]; Lu et al. [2022c]; Kirillov et al. [2023]. The subsequent tracking step leverages varied techniques Lowe [2004]; Teed and Deng [2020]; Cheng and Schwing [2022]; Yang et al. [2023] to group masks of the same object across frames. However, this approach has inherent limitations. Segmentation methods struggle to resolve ambiguities because they cannot utilize information from other frames to enhance segmentation within each frame. Tracking lacks alternatives when segmentation fails, since its success heavily relies on consistent object appearance or location across consecutive frames. These drawbacks limit the paradigm's effectiveness when there are crowded scenes and discrete frames.

Figure 3.1: A densely packed shelf environment. The shelf holds objects from a wide array of categories. During the stowing process, a human operator may obscure the camera's view and rearrange the objects within the bin. The robot's task is to pick a specific object as directed by the given order index of the object's placement in the bin.

Our task resembles video instance segmentation (VIS), which involves segmenting and tracking objects in videos, with the leaderboard predominantly occupied by *simultaneous segmenting and tracking* methods Heo et al. [2022]; Cheng et al. [2021]; Huang et al. [2022]. This similarity suggests the potential to adopt their methods to realize an end-to-end solution in our task. However, their methods, mainly designed for videos with continuous frames, display subpar performance when faced with significant object movements between frames, a prominent challenge in our task.

We therefore introduce STOW in this chapter, Discrete-Frame **S**egmentation and **T**racking of Unseen **O**bjects for **W**arehouse Picking Robots, a new framework for addressing challenges in our context. STOW consists of a new paradigm to jointly perform segmentation and tracking and a novel module, called the multi-frame attention layer, that facilitates efficient inter-frame communication. It succeeds in simultaneously achieving **high segmentation accuracy, high tracking accuracy, and high robustness to the sim-to-real gap.** Remarkably, even when trained exclusively on synthetic images, our method **significantly surpasses baseline on real data and live robot experiments.**

In summary, our main contributions are:

1. *Task formulation* for unseen object instance segmentation and tracking in discrete frames as well as realistic *synthetic data generation* and *real dataset data collection and manual labeling* for bins in the shelf and tabletop environments, facilitating research in this domain

2. *A new paradigm* to perform joint segmentation and tracking in discrete frames, along with a *new module, i.e., multi-frame attention*, that efficiently communicates information across

frames

3. *Experiments conducted on real data and on a working robot* to verify our network's superior performance

3.2 Related Works

Unseen object instance segmentation. In computer vision, traditional object instance segmentation requires prior knowledge about the objects. In contrast, our work targets potentially unseen objects without such knowledge. Previous efforts, such as UCN Xiang et al. [2021], UOIS Xie et al. [2021], and MF Lu et al. [2022c], addressed unseen object discovery in single-frame images using varied strategies, e.g., RGB-D feature embeddings and metric learning loss. A recent model, SAM Kirillov et al. [2023], also demonstrates robust segmentation across a wide variety of objects by training on a large amount of data. While these works focus on segmenting and tracking in single-frame images, our research extends it to multiple frames.

Video object segmentation. Like the video object segmentation task Xu et al. [2018]; Pont-Tuset et al. [2017]; Cheng and Schwing [2022], we track unseen objects in the test set. However, the VOS task assumes accessibility to an object's mask in one frame, which is not applicable in our task. For video object instance segmentation, previous works adhere to the *tracking by detection* paradigm, e.g., Bewley et al. [2016] and similarly Wojke et al. [2017]; Luiten et al. [2020]; Garg and Goel [2021], and often address the problem of multi-object tracking (MOT), i.e., the estimation of bounding boxes and identities of objects in consecutive RGB image streams. MOT tasks usually focus on tasks in traffic scenes and objects like people Zheng et al. [2015]; Wu et al. [2017] and vehicles.

Video Instance Segmentation. Our task diverges from existing VIS (Video Instance Segmentation) datasets Yang et al. [2019]; Qi et al. [2022] in two main aspects. First, while VIS employs a closed-set category approach for detection/segmentation, our open-set problem adds complexity by recognizing instances regardless of class, as opposed to relying on learned patterns. Second, unlike VIS's focus on continuous, limited-changes video sequences, our dataset emphasizes tracking amid drastic changes between frames, making it ill-suited for benchmarking with VIS datasets.

Unsupervised multi-object tracking. The unsupervised video instance segmentation task introduced in DAVIS 2019 Caelles et al. [2019] bears similarity to Video Instance Segmentation (VIS), with a focus on open-set category objects akin to our task. However, our approach diverges in two key aspects: (1) despite targeting unseen objects, DAVIS 2019 predominantly includes humans, vehicles, and animals, contrasting with the warehouse objects our study emphasizes, and (2) akin to VIS, it necessitates objects to exhibit continuous movement, thereby avoiding an ill-defined task.

Image co-segmentation. Our task is similar to object co-segmentation Rother et al. [2006], which extracts recurring objects from an image pair or a set of images. While the goal of co-segmentation is to identify shared objects in a scene, we focus on instance segmentation and do not allow similar objects of the same class. Distinguishing between an object *instance* and an object *class* is a crucial requirement for industrial warehouses.

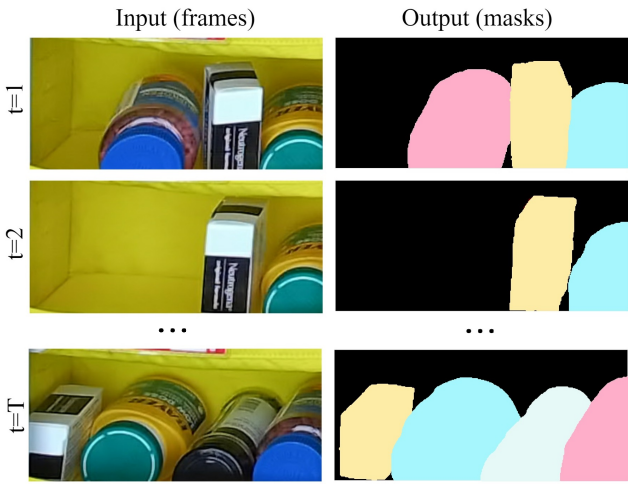


Figure 3.2: Task Illustration: The left column presents the images inputted into our network, while the right column showcases the expected segmentation and tracking outcomes. Identical colors indicate the same object.

Temporal Attention. In temporal attention, our multi-frame method aligns with but uniquely stands out from prior works. Context R-CNN integrates per-RoI features from various frames to enhance current detections. Unlike its static approach and two-phase method, we embed temporal attention within each block, updating all frame object queries post each temporal attention. Meanwhile, Zhao et al. [2022] proposed module called Alignment-guided Attention (ATA) which apply temporal attention on patches with similar features through bipartite matching. Unlike ATA's 1D fixed-size patch focus, our technique employs all object queries across all images, capturing varied masks and accessing broader information.

3.3 Problem Formulation

In this work, we introduce the novel task of segmenting and tracking unseen objects given a series of input discrete image frames. Though challenging, this task has broad applications in the field of robotics and is particularly useful in warehouse environments, such as that shown in Figure 3.1. Scenes involving warehouse shelves can be exceedingly packed and cluttered, consisting of a vast assortment of items, including some that have not been previously encountered. Moreover, there may be temporal gaps between successive snapshots of the scene, during which human workers may place new objects, robots may retrieve existing items, and some objects may undergo changes in pose.

Formally, we formulate the problem as follows. The input to the task is a sequence of images $\mathcal{I} = \{I_1, I_2, \cdots, I_T \mid I_t \in \mathbb{R}^{H \times W \times C_I}\}$, where H and W are the height and width of the images, respectively, and C_I represents the number of channels, i.e., 3 for RGB images and 4 for RGB-D inputs. The task involves detecting, segmenting, and tracking $K_\mathcal{I}$ object instances that appear in these input images, where $K_\mathcal{I}$ may not be known beforehand. The output of the task is a set of binary object instance masks, $\mathcal{M}_t = \{M_t^1, M_t^2, \cdots, M_t^{K_\mathcal{I}} \mid M_t^i \in \{0,1\}^{H \times W}, i = 1, 2, \cdots, K_\mathcal{I}\}$, corresponding to each input image $I_t \in \mathcal{I}$. Figure 3.2 shows the problem setting under consideration.

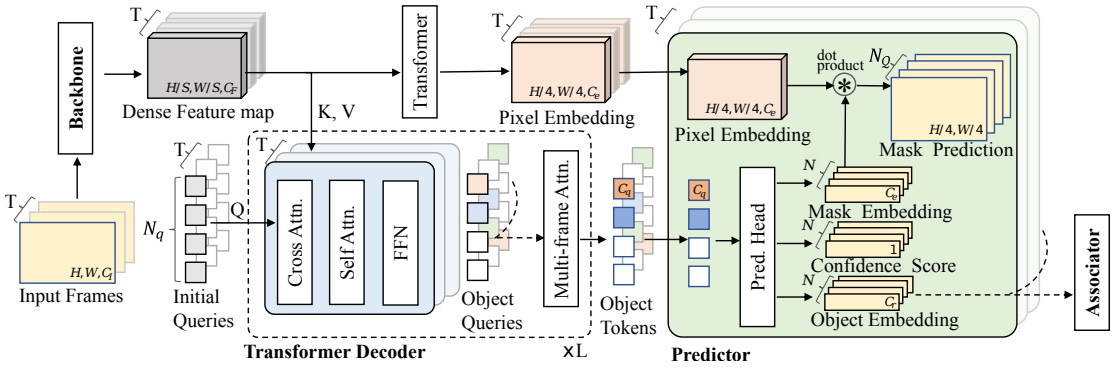


Figure 3.3: As outlined by a dashed rectangle, our transformer decoder ingests dense feature maps converted from input frames and produces object tokens for each image. These tokens predict confidence scores, mask embeddings for mask prediction, and object embeddings for association. We also introduce a novel "multi-frame attention" layer, which attends to object queries from all frames.

3.3.1 Method

Our system (Figure 3.3) uses query-based transformer architectures Cheng et al. [2022]; Huang et al. [2022] for object detection and segmentation, which we describe in Sec. 3.3.2. To enable the tracking of object instances across discrete image frames, we introduce the learning of additional object embeddings for tracking (Sec. 3.3.4) as well as a multi-frame attention layer to distinguish between identical or distinct object instances (Sec. 3.3.3). Sec. 3.3.5 discusses training specifics and loss functions.

3.3.2 Backbone Architecture

Following Cheng et al. [2022]; Huang et al. [2022], the backbone of our network consists of three components: (1) a ResNet-based image encoder, (2) a transformer-based object query decoder, and (3) several prediction heads tasked with determining object properties, such as the likelihood of object existence and object masks.

3.3.3 Multi-Frame Attention

ResNet-based image encoder. We use ResNet-50 He et al. [2016] to transform every input frame $I_t \in \mathbb{R}^{H \times W \times C_I}$ $(t \in 1, 2, \ldots, T)$ into a dense low-resolution feature map $F_t \in \mathbb{R}^{\frac{H}{S} \times \frac{W}{S} \times C_F}$. Here, C_F denotes the channel dimension of the output dense feature map, while $S = 32$ is the downsampling ratio used in this work.

Transformer-based object query decoder. We employ a DETR-like transformer decoder Carion et al. [2020] that takes the produced dense feature map F_t as input and learns to decode a set of N_q object tokens $\{\mathbf{q}_t^1, \mathbf{q}_t^2, \cdots, \mathbf{q}_t^{N_q}\} \in \mathbb{R}^{C_q}$ as the outputs. Each object token contains the latent information necessary for tasks such as classification estimation, mask prediction, and tracking in discrete frames. Our transformer decoder consists of $L = 10$ transformer blocks; each block contains one cross-attention layer, one self-attention layer, one feed-forward layer, and one novel multi-frame attention layer that correlates object features across different image frames (Sec. 3.3.3).

Prediction head for object masks. To get a per-pixel segmentation mask $M_t^i \in \{0,1\}^{H\times W}$ for each object token $\mathbf{q}_t^i$, we first use a two-layer multilayer perceptron (MLP), which maps an input object token q_t^i to a mask embedding $\mathbf{e}_t^i \in \mathbb{R}^{C_e}$; we then employ a multi-scale deformable attention transformer module (MSDeformAttn) Zhu et al. [2020] to convert the dense feature map F_t to a pixel embedding map $P_t \in \mathbb{R}^{\frac{H}{4}\times\frac{W}{4}\times C_e}$. Here, C_e denotes the channel dimensions used for the mask and pixel embeddings. We calculate the dot product between the object embedding $\mathbf{e}_t^i$ and the pixel embedding P_t in order to obtain the mask prediction $\hat{M}_t^i$ at a reduced resolution of $\frac{H}{4}\times\frac{W}{4}$. Subsequently, a bilinear upsampling operator is applied to map $\hat{M}_t^i$ back to the original image resolution for the final mask prediction $M_t^i \in \{0,1\}^{H\times W}$.

Prediction head for object existence scores. Taking the object token $\mathbf{q}_t^i$ as input, we leverage a simple linear layer to estimate an object existence likelihood score $s_t^i \in [0,1]$. In the context of unseen object instance segmentation, we are not concerned with precise target object categories, nor do we have access to this knowledge. This characteristic simplifies the task into a binary classification, the objective of which is to estimate the confidence score of whether each segment corresponds appropriately to an object.

3.3.4 Object Embedding for Tracking

In addition to the predicted object existence score and its segmentation mask in each frame, we add a new prediction head for object tracking embedding to enable the association of object tokens belonging to the same object from different input frames. Specifically, we employ a two-layer MLP to learn a mapping from the input object query $\mathbf{q}_t^i$ to another object embedding used for tracking $\mathbf{r}_t^i \in \mathbb{R}^{C_r}$. Here, C_r represents the number of channels of the object embedding vector.

During the inference phase, we implement an associator to group object tokens with similar object embeddings. Specifically, we sequentially traverse each frame in the sequence. For each frame, we retain only those object tokens $\mathbf{q}^i$ whose confidence scores surpass a predefined threshold δ_{score}. For each input sequence, we maintain a trajectory bank, $\mathcal{T}$, that encompasses the trajectories of all observed objects. Each trajectory in the trajectory bank $\mathcal{T}^i \in \mathcal{T}$ consists of object tokens $\mathbf{q}^i$ that are considered to belong to the object i. Subsequently, we compute the similarity score between these selected object tokens and the previous trajectory using the following equation:

$$Sim(\mathbf{q}^i, \mathcal{T}^j) = \max(R(\mathbf{q}^i) \cdot R(\mathbf{q}_k^j)), \text{for } \mathbf{q}_k^j \in \mathcal{T}^j. \tag{3.1}$$

In this equation, R is the function that transforms the object token q^i into the object embedding r^i.

After this step, we use the Hungarian algorithm Kuhn [1955] to search for an optimal bipartite matching $\hat{\varrho}$ from all possible bipartite matchings $\mathcal{P}$ that can maximize the overall similarity between object tokens $\mathbf{q}$ and the trajectory bank $\mathcal{T}$:

$$\hat{\varrho} = \arg\max_{\varrho\in\mathcal{P}} \sum_{i}^{N_q} Sim(\mathbf{q}^i, \mathcal{T}^{\varrho(i)}). \tag{3.2}$$

We initialize the trajectory bank, $\mathcal{T}$, with an adequate number of false alarm tokens, $\mathcal{T}_{FA}$, each of which exhibits a constant similarity δ_{match} to all object tokens. The hyper-parameter δ_{match} can also be interpreted as a false alarm threshold in matching. Any predicted object tokens assigned to

tokens from $\mathcal{T}_{FA}$ are assumed not to match any existing trajectory; thus, a new trajectory is opened for them.

Our tracking method also enables the possibility of handling multiple identical objects in the same scene. Specifically, object tokens corresponding to identical objects are likely to be recognized due to the expectation that object embeddings are near to them.

The previous modules we introduced work independently for each frame without any cross-frame information exchange. To facilitate efficient communication between frames, we introduce a new component, *the multi-frame attention layer*, into the transformer decoder.

The multi-frame attention layer is an extension of the self-attention layer, which operates on object queries from a single frame; it attends to object queries from all accessible frames. To illustrate, we denote the intermediate object queries after each feed-forward network as $\mathbf{X}_l^t$, where l and t indicate the index of transformer blocks and frames, respectively. A standard self-attention layer (with a residual path) computes the following (we omit the normalization term $\sqrt{d_k}$ here for simplicity):

$$\text{SelfAttn}(\mathbf{X}_l^t) = \text{softmax}(f_Q(\mathbf{X}_l^t) \cdot f_K(\mathbf{X}_l^t)^T) f_V(\mathbf{X}_l^t) + \mathbf{X}_l^t. \tag{3.3}$$

In contrast, our multi-frame attention layer computes:

$$\text{MultiFrameAttn}(\mathbf{X}_l^t) = \text{softmax}(f_Q(\mathbf{X}_l^t) \cdot f_K(\mathbf{X}_l)^T) f_V(\mathbf{X}_l) + \mathbf{X}_l^t. \tag{3.4}$$

Here, $\mathbf{X}_l$ represents the set of object queries from all frames $\mathbf{X}_l = \{\mathbf{X}_l^t, \text{ for } t = 1, 2, \ldots, T\}$. The functions f_Q, f_K, and f_V are linear transformations that convert $\mathbf{X}_l$ or $\mathbf{X}_l^t$ into a C-dim space.

The multi-frame attention layer is positioned at the end of each transformer decoder block, which is repeated L times in our network. Therefore, output object queries can incorporate the dense feature map of the current frame to update their prediction after communicating with object queries from other frames. Furthermore, the multi-frame attention layer is computationally efficient since it attends only to object queries from all frames, typically around 100 queries per frame vs the 1200 queries per frame used in Mask2Former-video Cheng et al. [2021] for images with size 640×480. Such efficiency lets it process long-term input sequences and large-scale images.

3.3.5 Training and Losses

Our model adopts the same loss function as utilized in Mask2Former Cheng et al. [2022] for classification and mask prediction. This includes the softmax cross-entropy loss, denoted as $\mathcal{L}_{\text{class}}$, for classification, along with binary cross-entropy loss, denoted as $\mathcal{L}_{\text{ce}}$, and the Dice loss, denoted as $\mathcal{L}_{\text{dice}}$, for mask prediction. To address tracking loss, we employ the contrastive loss $\mathcal{L}_{\text{contra}}$ used in DCN Florence [2020] along with the softmax loss (also referred to as the n-pair loss or InfoNCE loss) from CLIP Radford et al. [2021]. For an object token $q_{\hat{t}}^{o_i}$, object tokens paired with the same object o_i in different frames $q_t^{o_i}, t \neq \hat{t}$ are treated as positive samples and pushed closer, while tokens assigned to different objects or backgrounds are treated as negative pairs and pushed away. See the supplementary material for more details.

Consistent with the approach in DETR Carion et al. [2020] and Mask2Former Cheng et al.

[2022], we leverage the Hungarian algorithm Kuhn [1955] to establish a bipartite matching between the predicted object tokens and ground truth that minimizes the overall loss. Notably, we exclude the tracking loss from the Hungarian algorithm's computation given that $\mathcal{L}_{softmax}$ is influenced by the object tokens contributing to this loss. The final loss is $\mathcal{L}_{\text{total}} = \lambda_{class}\mathcal{L}_{\text{class}} + \lambda_{\text{ce}}\mathcal{L}_{\text{ce}} + \lambda_{dice}\mathcal{L}_{\text{dice}} + \lambda_{\text{contra}}\mathcal{L}_{\text{contra}} + \lambda_{\text{softmax}}\mathcal{L}_{\text{softmax}}$.

3.4 Experiments

We assess our methodology in two typical environments—bin in a shelf and tabletop—both of which are representative settings in a multitude of warehouse and domestic applications. For each setting, we generate corresponding synthetic data for the training phase and collect and annotate real-world data for evaluation. We also integrate our approach into a bin-picking robotic system for practical experimentation.

We train our models separately on distinct synthetic datasets for the shelf and table environments. Each run exclusively uses one dataset and is evaluated against the corresponding real-world test set. Inference takes around 0.4 seconds for a 15-frame sequence on an RTX 2080Ti. Additional training details are in the supplementary material.

3.4.1 Dataset and Evaluation

Synthetic data. We construct mesh models for shelf and table bins using textures from the CC0 dataset ambientCG [2023], and object meshes from the Google Scanned dataset Downs et al. [2022], consisting of over 1000 models. Excluding 70 with isolated parts, we utilize 900 objects for the training set and 100 for validation. Objects are randomly rotated and positioned to avoid collision, with no heavy occlusion in the shelf environment. In total, we generate around 10,000 sequences for the shelf, each with at least 2 packed frames, and 2,000 sequences for the tabletop, each containing 15 frames.

Real-world data. For real-world scenarios, we collect and manually label 44 sequences with 220 images for the shelf scenario and 20 sequences with 280 images for the tabletop scenario. Our dataset includes over 150 diverse objects. These range from relatively simple objects, such as boxes and bottles, to more complex ones, like transparent water bottles enclosed in plastic bags. In each sequence, we progressively add objects until either the bin is full or around 10 objects are placed on the table. Object rearrangement could occur between any two frames, leading to significant changes in object location and appearance.

Evaluation metrics. We adopt the evaluation method of the VIS challenge Yang et al. [2019], a modified version of the MS-COCO metric Lin et al. [2014]. In video instance segmentation, each object is represented by a series of masks, and the Intersection over Union (IoU) is calculated at the level of these mask sequences. To construct the Precision-Recall (PR) curve, the confidence threshold is systematically varied, with each threshold yielding a distinct data point on the curve. The area under the PR curve provides the Average Precision (AP).

In the context of this study, if not further specified, AP@0.5 denotes the average precision at an IoU threshold of 0.5. Similarly, AP@all represents the mean AP calculated over multiple IoU thresholds, specifically from 50% to 95% in 5% increments.

Method	Shelf		Tabletop	
	AP@all	AP@0.5	AP@all	AP@0.5
MinVIS	6.3	21.2	0.7	0.0
Mask2Former Video	35.0	66.1	27.7	56.7
VITA	42.7	70.1	26.6	55.0
STOW (Ours)	**55.6**	**81.3**	**49.7**	**75.4**

Table 3.1: Comparison between our method and leading video instance segmentation methods. Networks are trained on synthetic data and tested on real, unseen data. All use ResNet-50He et al. [2016] and train on an identical number of images.

3.4.2 Results and Analysis

Baseline methods. We benchmark our approach against three state-of-the-art Video Instance Segmentation (VIS) methods: MinVIS Huang et al. [2022], Mask2Former-video Cheng et al. [2021], and VITA Heo et al. [2022]. To ensure an equitable comparison, all methods utilize ResNet-50 as the backbone, and the hyperparameters (such as batch size, maximum iterations, and the number of sampled frames) are standardized to match ours. Hence, all methods are trained on an identical number of images.

Qualitative results. As depicted in Table 3.1, our method notably outperforms existing VIS techniques, yielding an approximate 10% improvement in the shelf environment and a 20% increase in the table environment, even without using the multi-frame attention layer. Interestingly, MinVIS exhibits subpar performance in our task. We speculate that this is due to its reliance on the proximity between object tokens as a measure of similarity; this method presupposes that the changes between frames are minimal and that objects maintain similar locations and appearances. However, these conditions do not consistently hold in our task, potentially explaining the subpar performance.

Quantative results. We also show visualization results on a subset of real test data in Figure 3.4 and Figure 3.5. Our approach adeptly handles frame changes amid various types of noise. Despite significant movement and rotation between frames, the network successfully segments and tracks a broad array of object categories. It efficiently segments and continues tracking any new objects that are introduced. Figure 3.5 also demonstrates the method's robustness against backgrounds: it avoids predicting them as objects even though walls are not included in the training set. Supplementary materials contain additional results that contrast our approach with other methods.

3.4.3 Ablation Study

Frame attention. We evaluate the performance of our method with and without the cross-frame attention module on both the shelf and table scenes using ResNet-50 and Swin-T backbones. The cross-frame attention module yields consistent performance improvements across all configurations, as shown in Table 3.2.

Sim2Real gap. We evaluate our methods on the synthetic validation set, which contains objects not included in the training set, and compare it to the number we tested on the real test set. Results, shown in Table 3.3, reveal that other methods get results that are relatively to ours for

Multi Frame	Shelf		Tabletop	
	AP@all	AP@0.5	AP@all	AP@0.5
-	51.8	78.7	44.4	68.5
✓	**55.6**	**81.3**	**49.7**	**75.4**

Table 3.2: With and without the multi-frame attention layer. The left column denotes whether we incorporate multi-frame attention in this experiement. All other hyper-parameters remain the same. Use of the frame attention layer boosts both shelf and tabletop environment performance by ∼5%.

Method	Synthetic		Real	
	AP@all	AP@0.5	AP@all	AP@0.5
MinVIS	0.3	2.6	0.7	0.0
M2F-V	71.6	83.7	27.7	56.7
VITA	69.4	81.9	26.6	55.0
STOW (Ours)	**74.1**	**89.3**	**49.7**	**75.4**

Table 3.3: Ablation study on solving the sim2real gap. After training on synthetic tabletop training set, we separately evaluate each method on a synthetic tabletop validation set and a real tabletop test set; note that objects in the synthetic validation set are not included in the synthetic training set.

the synthetic set, but their performance drops dramatically when we evaluate on the test set. This implies that they have difficulties solving the sim2real gap.

3.4.4 Real Robot Applications

We integrated our visual perception technique into an autonomous shelf-picking system Grotz et al. [2023]. The system's multi-component software architecture is managed by a state machine. The system setup uses a UR16e industrial robot situated in front of an industrial warehouse shelf filled with objects. Within the robotics community, it's standard to address perception problems by combining unseen instance segmentation methods with other established techniques, as seen in Goyal et al. [2022],Gouda et al. [2022], Chen et al. [2023], Sundermeyer et al. [2021]. Other than VITA Heo et al. [2022] in Table 3.1, we also combine UCN Xiang et al. [2021], a staple in unseen

Figure 3.4: Visualized results for the shelf environment. Masks with the same color and index are associated and predicted as the same object by the network.

Figure 3.5: Visualized results for the table environment. Masks with the same color and index are associated and predicted as the same object by the network.

instance segmentation, with SIFT Lowe [2004], a renowned keypoint extraction method, reflecting the conventional solution for this challenge. The center of the mask is used as the grasping point for suction cap. Our evaluation protocol employs a fixed set of diverse items, stowed at specific locations and orientations within the bins, to ensure reproducibility and comparability of results since performance can fluctuate with different item configurations and inherent system stochasticity.

We testing each method with 82 trials across different levels of difficulty, involving over 100 objects. For UCN Xiang et al. [2021]+SIFT Lowe [2004], the picking success rates stand at 40.2%. Using VITA Heo et al. [2022], it is 46.3%. With our STOW method, this rate increase to 74.4%.

3.5 Limitations

Our method, while effective, has limitations in handling highly cluttered environments and complex objects. False positives and negatives occur in object detection, especially in intricate settings. The segmentation process can result in over- or under-segmentation due to complex object boundaries and textures. In object tracking, we sporadically encounter mistracking incidents and occasional failures to distinguish between multiple objects. Refer to the supplementary material for detailed analyses and examples of these limitations.

3.6 Discussion

In this chapter, we introduce the task of segmenting and tracking unseen objects in discrete frames which is widely used in robotics tasks but under investigation. We formulated the problem and collected both synthetic and real datasets. We also propose a novel paradigm for joint segmentation and tracking, incorporating multi-frame attention for better inter-frame communication. Even

when trained solely on synthetic data, our method adeptly handles clustering and large movements of unseen objects in real-world sequences. Our innovative approach excels in segmenting and tracking within both shelf and tabletop settings, surpassing state-of-the-art techniques with a 10%-20% improvement in AP in real-world scenarios and more than 20% success rate in robot experiments.

Although masks boast significant generalization abilities for unfamiliar objects and offer universality in representation, they are largely tied to specific tasks such as picking and placing. This can lead to uncertainty when tasks lack well-defined masks, like wiping a table or folding a towel. Furthermore, the focus remains on object segmentation and tracking with minimal input from language instructions and complex task comprehension. To address these challenges and devise a method capable of comprehending intricate language instructions and offering guidance for both novel objects and varied manipulation tasks, we present a new approach in the subsequent chapter.

Chapter 4

HAMSTER: Hierarchical Action Models for Open-World Robot Manipulation

Large foundation models have shown strong open-world generalization to complex problems in vision and language, but similar levels of generalization have yet to be achieved in robotics. One fundamental challenge is the lack of robotic data, which are typically obtained through expensive on-robot operation. A promising remedy is to leverage cheaper, "off-domain" data such as action-free videos, hand-drawn sketches or simulation data.

In this chapter, we posit that *hierarchical* vision-language-action (VLA) models can be more effective in utilizing off-domain data than standard monolithic VLA models that directly finetune vision-language models (VLMs) to predict actions. In particular, we study a class of hierarchical VLA models, where the high-level VLM is finetuned to produce a coarse 2D path indicating the desired robot end-effector trajectory given an RGB image and a task description. The intermediate 2D path prediction is then served as guidance to the low-level, 3D-aware control policy capable of precise manipulation. Doing so alleviates the high-level VLM from fine-grained action prediction, while reducing the low-level policy's burden on complex task-level reasoning. We show that, with the hierarchical design, the high-level VLM can transfer across significant domain gaps between the off-domain finetuning data and real-robot testing scenarios, including differences on embodiments, dynamics, visual appearances and task semantics, etc. In the real-robot experiments, we observe an average of 20% improvement in success rate across seven different axes of generalization over OpenVLA, representing a 50% relative gain. Visual results are provided at: `https://hamster-robot.github.io/`

4.1 Overview

Developing general robot manipulation policies has been notoriously difficult. With the advent of large vision-language models (VLMs) that display compelling generalization capabilities, there is optimism that the same recipe is directly applicable to robot manipulation. A line of prior work [Brohan et al., 2023a; Kim et al., 2024; Black et al., 2024] builds open-world vision-language-action models (VLAs) by finetuning off-the-shelf pretrained VLMs to directly produce robot ac-

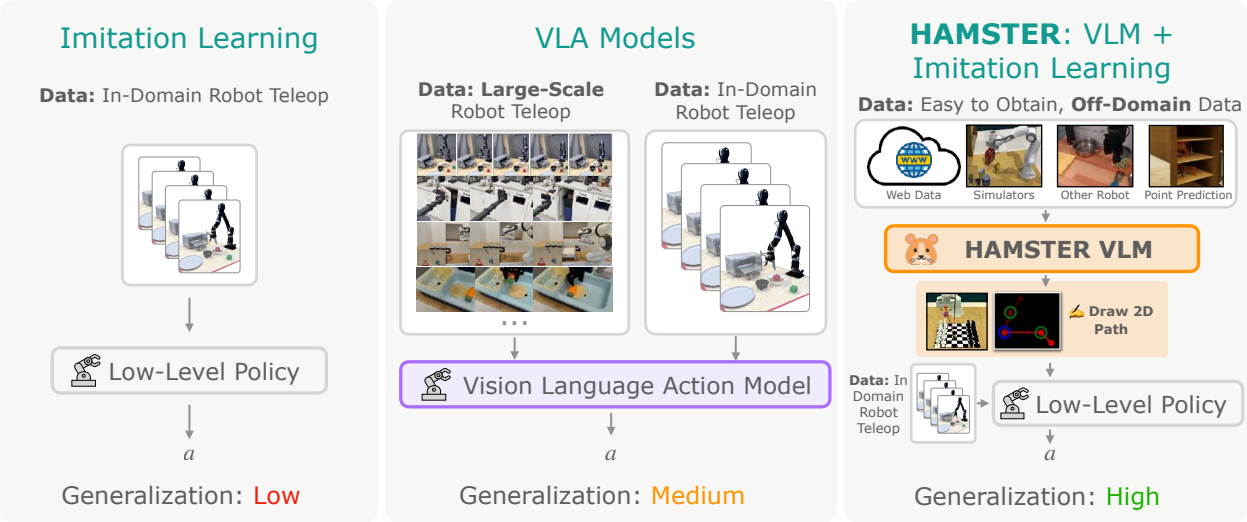


Figure 4.1: Overview of HAMSTER, VLAs and "smaller" imitation learning methods. HAMSTER's hierarchical design results in better generalization with a small amount of in-domain data. HAMSTER is able to utilize cheap training sources such as videos or simulations for enhanced generalization.

tions. These VLA models, which we refer to in this work as *monolithic* VLA models, rely crucially on large robotics datasets, complete with on-robot observations, e.g., images and proprioceptive states, and actions. However, on-robot data is expensive, since end-to-end observation-action pairs are typically collected on the robot hardware through, e.g., teleoperation. Despite recent community-wide efforts in building large-scale robotics datasets [O'Neill et al., 2024; Khazatsky et al., 2024], the size, quality, and diversity of existing robotics datasets are still limited, and monolithic VLA models have yet to demonstrate emergent capability comparable to VLMs and LLMs in other domains of study. Moreover, monolithic VLA models are constrained by their inference frequency to achieve dexterous and dynamic manipulation tasks [Brohan et al., 2023a; Kim et al., 2024].

On the other hand, relatively small robot policy models have shown impressive dexterity and robustness. Such models have demonstrated promise across a range of complex tasks involving contact-rich manipulation and 3D reasoning, spanning domains from tabletop manipulation [Shridhar et al., 2023; Goyal et al., 2023, 2024; Ke et al., 2024] to fine dexterous manipulation [Chi et al., 2023; Zhao et al., 2023]. Trained on relatively small datasets, these models show local robustness, and can achieve dexterous and high-precision control. However, they are often brittle to drastic changes in the environment or semantic description of the tasks [Pumacay et al., 2024]. These models also can struggle to effectively leverage simulation data for real-world manipulation tasks due to sim-to-real gaps in visual appearances and system dynamics [Li et al., 2024; Mandlekar et al., 2021].

In this chapter, we ask – how can we marry the generalization benefits of large VLMs, with the efficiency, local robustness, and dexterity of small policy models? Our key insight is that, instead of directly predicting robot actions, VLMs can be fine-tuned to produce intermediate representations as high-level guidance on solving the robot manipulation task. The intermediate representation can then be consumed by the low-level policy model to produce actions, alleviating

the low-level policy from the burden of long-horizon planning and complex, semantic reasoning. Further, if the intermediate representations are chosen such that they are *1)* easily obtainable from image sequences; *2)* largely embodiment agnostic; and *3)* sufficiently robust to subtle changes in dynamics, the VLM can be fine-tuned with *off-domain* data where robot actions are unavailable or inaccurate. Such off-domain data does not need to be collected on the actual robot hardware. Examples of off-domain data include action-free video data, simulation data, human videos, and videos of robot with different embodiments. These off-domain data are generally easier to collect and may already be abundant in existing datasets. We hypothesize, and show experimentally in Fig 4.7, that this hierarchical separation can allow VLA models to more effectively bridge the domain gap between off-domain data and in-domain robotic manipulation.

To this end, we propose a hierarchical architecture for VLAs, HAMSTER (**H**ierarchical **A**ction **M**odels with **S**epara**TE**d Path **R**epresentations), where large fine-tuned VLMs are connected to low-level policy models via 2D path representations[1]. A 2D path is a coarse trajectory of the 2D image-plane position of the robot end-effector[2], as well as where the gripper state changes, i.e., opens and closes (see Fig. 4.2). These 2D paths can be obtained cheaply and automatically from data sources such as action-free videos or physics simulations, using point tracking [Doersch et al., 2023; Karaev et al., 2025], hand-sketching [Gu et al., 2023], or proprioceptive projection. This allows HAMSTER can effectively leverage these abundant and inexpensive off-domain data when fine-tuning the high-level VLM. The hierarchical design presented in HAMSTER also offers additional advantages through the decoupling of VLM training and low-level action prediction. Specifically, while the higher-level VLM is predicting semantically meaningful trajectories from monocular RGB camera inputs, the lower-level policy models can additionally operate from rich 3D and proprioceptive inputs. In doing so, HAMSTER inherits the semantic reasoning benefits of VLMs along with the 3D reasoning and spatial awareness benefits of 3D policy models [Goyal et al., 2024; Ke et al., 2024]. Moreover, the high-level VLM and low-level policy model can be queried at different frequencies

In summary, we study a family of hierarchical VLA models HAMSTERs, where finetuned VLMs are connected to low-level 3D policy models [Goyal et al., 2024; Ke et al., 2024]. The 2D paths produced by high-level VLMs serve as guidance for a low-level policy that operates on rich 3D and proprioceptive inputs, allowing low-level policies to focus on robustly generating precise, spatially-aware actions. In our experiments, we observe an average of 20% improvement in success rate over seven different axes of generalization over OpenVLA [Kim et al., 2024], which amounts to 50% relative gain, as shown in Table B.3. Since HAMSTER is built on both open-source VLMs and low-level policies, it can serve as a fully open-sourced enabler for the community-building vision-language-action models. It is important to note that while we are certainly not the first to propose hierarchical VLA models [Gu et al., 2023; Nasiriany et al., 2024a], we propose the novel insight that this type of hierarchical decomposition allows for these models to make use of abundant off-domain data for improving real-world control. This opens the door to alternative ways of training large vision-language-action models using cheaper and more abundant data

[1]Representations similar to 2D paths has been explored in the robot learning literature [Gu et al., 2023], primarily as a technique for flexible task specification. We refer readers to section 4.2 for a detailed discussion.

[2]For human video, this corresponds to the position of the palm center or fingertips.

sources.

4.2 Related Work

LLMs and VLMs for robotics. Early attempts in leveraging LLMs and VLMs for robotics are through pretrained language [Jang et al., 2022; Shridhar et al., 2023; Singh et al., 2023] and visual [Shah and Kumar, 2021; Parisi et al., 2022; Nair et al., 2023; Ma et al., 2023] representations. However, these are insufficient for complex semantic reasoning and generalization to the open world [Brohan et al., 2023a]. Recent research has focused on directly leveraging open world reasoning and generalization capability of LLMs and VLMs, by prompting or fine-tuning them to, e.g., generate plans [Duan et al., 2024; Huang et al., 2023b; Lin et al., 2023; Liang et al., 2023; Singh et al., 2023; Brohan et al., 2023b] or construct value [Huang et al., 2023a] and reward functions [Kwon et al., 2023; Sontakke et al., 2023; Yu et al., 2023b; Ma et al., 2024; Wang et al., 2024]. Our work is more closely related to VLA models, summarized below.

Monolithic VLA models as language-conditioned robot policies. Monolithic VLA models have been proposed to produce robot actions given task description and image observations directly [Brohan et al., 2023a; Jiang et al., 2023; Team et al., 2024; Kim et al., 2024; Radosavovic et al., 2023]. Monolithic VLA models are often constructed from VLMs [Liu et al., 2024d; Bai et al., 2023; Driess et al., 2023; Lin et al., 2024], and are trained on large-scale on-robot data [Brohan et al., 2023a; O'Neill et al., 2024; Khazatsky et al., 2024] to predict actions as text or special tokens. However, due to the lack of coverage in existing robotics datasets, they must be finetuned in-domain on expensive on-robot data. Their action frequency is also constrained by inference frequency, limiting their capability to achieve dexterous and dynamic tasks. The most relevant monolithic VLA model to our work is LLARVA [Niu et al., 2024], which predicts end-effector trajectories in addition to robot actions. However, LLARVA only uses trajectory prediction as an auxiliary task to improve the action prediction of a monolithic VLA model. In contrast, our work takes a hierarchical approach, enabling us to use specialist lower-level policies that take in additional inputs the VLMs cannot support, such as 3D pointclouds, to enable better imitation learning. Our predicted paths then enable these lower-level policies to generalize more effectively.

VLMs for predicting intermediate representations. Our work bears connections to prior methods using vision-language models to predict intermediate representations. These methods can be categorized by the choice of predicted representations:

Point-based predictions: A common intermediate prediction interface has been keypoint affordances [Stone et al., 2023; Sundaresan et al., 2023; Nasiriany et al., 2024b; Yuan et al., 2024b; Kuang et al., 2024]. Keypoint affordances can be obtained through using open-vocabulary detectors [Minderer et al., 2022], iterative prompting of VLMs [Nasiriany et al., 2024b], or fine-tuning detectors to identify certain parts of an object by semantics [Sundaresan et al., 2023]. Perhaps most related to our work, Yuan et al. [2024b] finetune a VLM to predict objects of interest as well as free space for placing an object, and Liu et al. [2024b] propose a mark-based visual prompting procedure to predict keypoint affordances as well as a fixed number of waypoints. As opposed to these, our work finetunes a VLM model to not just predict points but rather entire 2D paths, making it more

broadly applicable across robotic tasks.

Trajectory-based predictions: The idea of using trajectory-based task specifications to condition low-level policies was proposed in RT-trajectory [Gu et al., 2023], largely from the perspective of flexible task specification. This work also briefly discusses the possibility of combining trajectory-conditioned model with trajectory sketches generated by a pre-trained VLM. Complementary to RT-Trajectory, the focus of this work is less on the use of trajectory sketches for task specification, but rather a hierarchical design of VLAs such that the high-level VLM can be fine-tuned with relative cheap and abundant data sources. This could include data such as action-free videos, or simulation data that look very different from the real world. We show that the emergent generalization capability of VLMs from its web-scale pretraining allows it transfer to test scenarios of interest with considerable visual and semantic variations. While RT-trajectory uses human effort or off-the-shelf pre-trained VLMs to generate trajectories, we show that fine-tuning VLM models on cheap data sources can generate significantly more accurate and generalizable trajectories (see Table. B.2). Moreover, our instantiation of this architecture enables the incorporation of rich 3D and proprioceptive information, as compared to monocular 2D policies [Gu et al., 2023].

Similarly, the emergence of track-any-point (TAP) models [Doersch et al., 2023; Wang et al., 2023] has enabled policies conditioned on object trajectories [Yuan et al., 2024a; Xu et al., 2024; Bharadhwaj et al., 2024] or points sampled from a fixed grid in the image [Wen et al., 2023]. While our current formulation focuses on end-effector trajectories, this framework can naturally extend to predicting object trajectories or other motion cues. By leveraging the predictive capabilities of VLMs, such an extension could further enhance the model's ability to generalize across diverse scenarios and improve its capacity for fine-grained motion reasoning.

Leveraging simulation data for training robot policies. There has been extensive work on leveraging simulation for robot learning. Simulation data is popular in reinforcement learning (RL), as RL on real robotic systems is often impractical due to high sample complexity and safety concerns [Lee et al., 2020; Handa et al., 2023; Torne et al., 2024]. Recently, simulation has been also exploited to directly generate [Fishman et al., 2022] or bootstrap [Mandlekar et al., 2023] large-scale datasets for imitation learning, to reduce the amount of expensive robot teleoperation data needed. Our work takes a different approach – using simulation data to finetune a VLM, and showing that VLM is able to transfer the knowledge learned from simulation data to real robot systems, despite considerable visual differences. A related observation is recently made by [Yuan et al., 2024b], but they use keypoint affordances as the interface between the VLM and the low-level policy as opposed to more general expressive 2D path representations.

4.3 Background

Imitation Learning via Supervised Learning. Imitation learning trains a policy $\pi_\theta(a \mid s, o, z)$ from expert demonstrations, where s denotes proprioceptive inputs, o includes perceptual observations (e.g., RGB images, depth), and z provides task instructions. Given an expert dataset $\mathcal{D} = \{(s_i, o_i, z_i, a_i)\}_{i=1}^N$, the policy is optimized via maximum likelihood estimation, maximizing $\mathbb{E}_{(s_i,o_i,z_i,a_i)\sim\mathcal{D}}\left[\log \pi_\theta\left(a_i \mid s_i, o_i, z_i\right)\right]$. Despite advancements in architectures such as 3D policy representations [Goyal et al., 2023; Ke et al., 2024], generalizing to novel semantic or visual variations

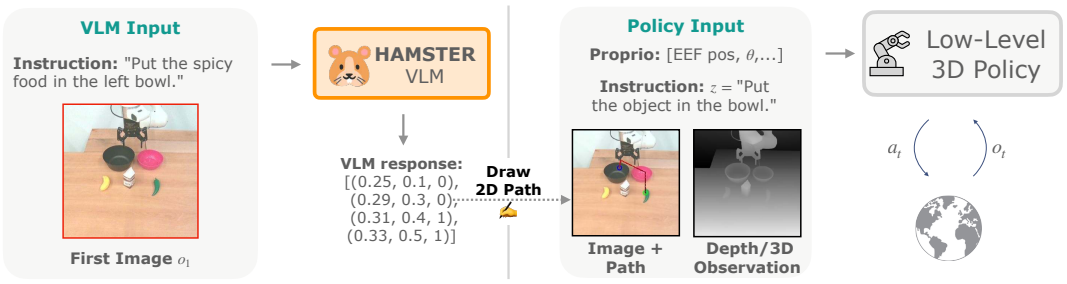


Figure 4.2: Depiction of HAMSTER's execution. The high-level VLM is called once to generate the 2D path. The low-level policy is conditioned on the 2D path and interacts with the environment sequentially to execute low-level actions. The path predicted by the VLM enhances the low-level policy generalization capability.

remains challenging. In this paper, we explore how VLMs can enhance imitation learning models for better generalization.

Vision-Language Models. VLMs [Liu et al., 2024a; Lin et al., 2024; Liu et al., 2024d] are large transformer models [Vaswani et al., 2023] that accept both vision and text tokens to generate text responses. They are pre-trained on extensive multimodal datasets [Zhu et al., 2023; Byeon et al., 2022] and later fine-tuned on high-quality, task-specific data [Shen et al., 2021; Lu et al., 2022b]. By tokenizing each modality into a shared space, these models autoregressively produce sequences of text tokens conditioned on an image and prior tokens. In our work, we assume access to such a pre-trained, text-and-image VLM [Lin et al., 2024; Liu et al., 2024d], further fine-tuned via a supervised loss that minimizes the negative log-likelihood of the target tokens.

4.4 HAMSTER: Hierarchical Action Models for Robotic Learning

In this work, we examine how VLA models can leverage relatively abundant data and demonstrate cross-domain transfer capabilities, as opposed to relying purely on expensive observation-language-action data collected on a robot. HAMSTER is a family of hierarchical VLA models designed for this purpose, exhibiting generalizable and robust manipulation. It consists of two interconnected models: first, a higher-level VLM that is finetuned on large-scale, off-domain data to produce intermediate 2D path guidance (detailed in Section 4.4.1), and second, a low-level policy that produces actions conditioned on 2D paths (detailed in Section 4.4.2).

The primary advantages of finetuning such a hierarchical VLM that produces intermediate representations as opposed to directly producing actions a with a monolithic model [Kim et al., 2024; Brohan et al., 2023a; Black et al., 2024] are threefold: *1)* our hierarchical VLM can leverage off-domain datasets lack of precise actions, e.g., simulation and videos; *2)* we find empirically that hierarchical VLMs producing 2D paths generalize more effectively cross-domain than monolithic VLA models; and *3)* the hierarchical design provides more flexibility on the sensory modality, and allows for asynchronous query of large high-level VLA models and small low-level policy models.

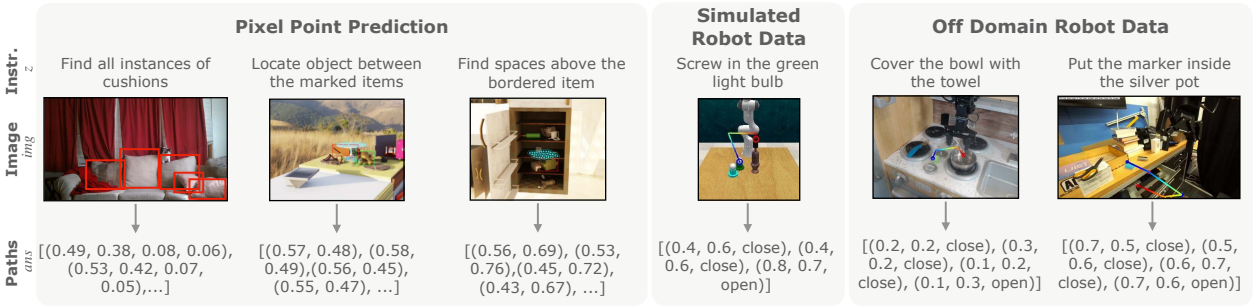


Figure 4.3: Off Domain Training Data: $\mathcal{D}_{\text{off}}$ contains (a) Pixel Point Prediction: 770k object location tasks from RoboPoint. (b) Simulated Robot Data: 320k 2D end-effector paths from RLBench environment. (c) Real Robot Data: 110k 2D end-effector paths from Bridge and DROID trajectories.

4.4.1 HAMSTER's VLM for producing 2D Paths Trained from Off-Domain Data

The high-level VLM of HAMSTER predicts a coarse 2D path p to achieve the task given a monocular RGB image $\texttt{img}$ and language instruction z, i.e., $\hat{p} \sim \texttt{VLM}(\texttt{img}, z)$. The 2D path p describes a coarse trajectory of the robot end-effector, or human hand in the case of human videos, on the input camera image. It also contains information about the gripper state. Formally, the 2D path is defined as $p = [(x_t, y_t, \texttt{gripper\_open}_t)]_t$ where $x_t, y_t \in [0, 1]$ are *normalized pixel locations* of the end effector's (or hand) position at step t, and $\texttt{gripper\_open}_t$ is a binary value indicating the gripper state, i.e., open and close.

Although, any pretrained text-and-image-input VLM [Lin et al., 2024; Liu et al., 2024d; Achiam et al., 2023] can be used to predict such a 2D path by casting an appropriate prompt, we find that pre-trained VLMs struggle with predicting such a path in a zero-shot manner (see Table B.2). Therefore, we finetune pre-trained VLMs on datasets that ground VLMs to robot scenes and path predictions collected from easier-to-obtain sources, i.e., internet visual-question-answering data, robot data from other modalities, and simulation data. This is in contrast to work such as Gu et al. [2023], where pre-trained VLMs are tasked with directly performing spatially relevant path generation.

We use VILA-1.5-13b [Lin et al., 2024] as our base VLM, a 13-billion-parameter vision language model trained on interleaved image-text datasets and video captioning data. Although it is possible to curate a dataset on path prediction $\{(\texttt{img}_i, z_i, p_i)\}_i$ and train the VLM *only* on the dataset, the literature [Brohan et al., 2023a; Yuan et al., 2024b] has shown that *co-training* the VLM on a variety of relevant tasks, all framed as VQA tasks, can help retain the VLM's generalization capability. To this end, we curate a multi-domain dataset to finetune this model for effective 2D path prediction.

Finetuning Objective and Datasets.

Predicting the 2D path of the end-effector requires understanding *what* objects to manipulate in a given task in terms of their pixel positions, but also reasoning about *how* a robot should perform the task. To enable this understanding, we collate a diverse off-domain dataset $\mathcal{D}_{\text{off}}$ from a wide range of modalities, including real-world data, visual question-answering data, and simulation

data. Importantly, *none* of this off-domain data used to train the VLM comes from the deployment environment, thereby emphasizing generalizability.

We assemble a dataset $\mathcal{D}_{\text{off}} = \{(\texttt{img}_i, z_i, \texttt{ans}_i)\}_{i=1}^{M}$ of image inputs $\texttt{img}_i$, language prompts z_i, and answer $\texttt{ans}_i$ consisting of three types of *off-domain* data: (1) pixel point prediction tasks (*what*); (2) simulated robotics tasks (*what and how*); (3) a real robot dataset consisting of trajectories (*what and how*). We detail each dataset below; see Figure 4.3 for visualization of each dataset's prompts and corresponding answers.

Pixel Point Prediction. For pixel point prediction, we use the RoboPoint dataset [Yuan et al., 2024b] with 770k pixel point prediction tasks, with most answers represented as a list of 2D points corresponding to locations on the image. A sample consists of a prompt z like `Locate object between the marked items`, an input image `img` and answer `ans` like `[(0.25, 0.11), (0.22, 0.19), (0.53, 0.23)]`.[3] See the left of Figure 4.3 for an example. This dataset consists of data automatically generated in simulation and collected from existing real-world datasets; its diverse tasks enable the HAMSTER VLM to reason about pixel-object relationships across diverse scenes while retaining its semantic generalization capabilities.

Simulated Robot Data. We additionally generate a dataset of simulated robotics tasks from RLBench [James et al., 2020], a simulator of a Franka robot performing tabletop manipulation for a wide array of both prehensile and non-prehensile tasks. We use the simulator's built-in planning algorithms to automatically generate successful manipulation trajectories. Given a trajectory, we use the first frame from the front camera as the image input `img`. We construct prompt z to instruct the VLM to provide a sequence of points denoting the trajectory of the robot gripper to achieve the given language instruction (see Figure 4.2). The ground-truth 2D path $p = [(x_t, y_t, \texttt{gripper\_open}_t)]_t$ is given by propriceptive projection using forward kinematics and camera parameters.

We generate 1000 episodes for each of 81 robot manipulation tasks in RLBench, each episode with $\sim$4 language instructions, for a total of around $320k$ $(\texttt{img}, z, \texttt{ans})$ tuples, where $\texttt{ans} = p$. See the middle of Figure 4.3 for an example.

Real Robot Data. Using real robot data allows us to ensure the VLM can reason about objects and robot gripper paths when conditioned on scenes, including real robot arms. We use existing, online robot datasets *not from the deployment environment* to enable this VLM ability. We source 10k trajectories from the Bridge dataset [Walke et al., 2023; O'Neill et al., 2024] consisting of a WidowX arm (different embodiment from test robot) performing manipulation tasks and around 45k trajectories from DROID [Khazatsky et al., 2024]. We covert both datasets to VQA dataset in as similar way as the simulated RL-Bench data, where the 2D paths are extracted from proprioception and camera parameters (see the right of Figure 4.3 for an example). Note that we essentially utilize the robot data as video data, where the end effector is tracked over time. In principle, this could be done with any number of point-tracking methods [Doersch et al., 2023] on raw video as well, with no action or proprioceptive labels.

We finetune the HAMSTER VLM on all three types of data by randomly sampling from all samples in the entire dataset with equal weight. We also include a 660k-sample VQA dataset [Liu et al.,

[3]Note that this is not a temporally ordered path, but rather a set of unordered points of interest in an image.

2024c] for co-training to preserve world knowledge. We train with the standardized supervised prediction loss to maximize the log-likelihood of the answers $\texttt{ans}$: $\mathbb{E}_{(\texttt{img}_i, z_i, \texttt{ans}_i) \sim \mathcal{D}_{\text{off}}} \log \text{VLM}\left(\texttt{ans}_i \mid \texttt{img}_i, z_i\right)$.

Remark. One issue with simulation and real robot data is that the extracted 2D paths p can be extremely long, e.g., exceeding one hundred steps. Since we want the HAMSTER VLM to reason at a *high level* instead of on the same scale as the low-level control policy, we simplify the paths p^o with the Ramer-Douglas-Peucker algorithm [Ramer, 1972; Douglas and Peucker, 1973] that reduces curves composed of line segments to similar curves composed of fewer points. We refer readers to Appendix B.7 for an ablation study.

4.4.2 Path Guided Low-Level Policy Learning

The low-level policy of HAMSTER $\pi_\theta(a \mid s, o, z, p)$ is conditioned on proprioceptive and perceptive observations, (optional) language instruction and, importantly, 2D path. While a low-level control policy *can* learn to solve the task without 2D path, the paths allow the low-level policy to forgo long-horizon and semantic reasoning and focus on local and geometric predictions to produce robot actions. As we find empirically (see Figure 4.4), 2D paths allow for considerably improved visual and semantic generalization of low-level policies.

HAMSTER's general path-conditioning framework allows lower-level policies to take in proprioceptive and perceptual (e.g., depth images) observations, that are not input to the high-level VLM. We consider low-level policies based on 3D perceptual information, i.e., $o = (\texttt{img}, \texttt{pointcloud})$, available at test time on a robotic platform with standard depth cameras. We study two choices of policy architecture, RVT-2 [Goyal et al., 2024] and 3D-DA [Ke et al., 2024] which has shown state-of-the-art results on popular robot manipulation benchmark [James et al., 2020].

Conditioning on Paths. Most policy architectures use the form $\pi_\theta(a \mid s, o, z)$ without 2D path inputs. One naïve option is to concatenate the path with proprioceptive or language inputs. However, because 2D paths vary in length, the architecture must handle variable-length inputs. To incorporate the 2D path $\hat{p}$ from the VLM without major modifications, we alternatively overlay the 2D path onto the image observation [Gu et al., 2023]. Our implementation follows this approach by drawing colored trajectories on all images in the trajectory $o_i^1, \ldots, o_i^T$: points at each (x_t, y_t) are connected with line segments using a color gradient to indicate temporal progression (see Figure 4.2(b)), and circles mark changes in gripper status (e.g., green for closing, blue for opening). If the policy architecture allows images with more than three channels, we can also include path drawing as separate channels, instead of overlaying it on the RGB channel. We empirically study both drawing strategies, overlay and concatenating channels, in section 4.5.3.

Policy Training. To train the policy, we collect a relatively small-scale task-specific dataset $\mathcal{D} = \{(s_i, o_i, z_i, a_i)\}_{i=1}^N$ on the robot hardware. During training, we use *oracle* 2D paths constructed by proprioception projection, similar to how the 2D paths are constructed for the VLM training data, and construct path-labeled dataset $\mathcal{D}_{\text{path}} = \{(s_i, o_i, z_i, p_i, a_i)\}_{i=1}^N$. We train a policy $\pi_\theta(a \mid s, o, z, p)$ with standard supervised imitation learning objectives on $\mathcal{D}_{\text{path}}$ to maximize the log-likelihood of the dataset actions: $\mathbb{E}_{(s_i, o_i, z_i, p_i, a_i) \sim \mathcal{D}_{\text{path}}} \log \pi_\theta(a_i \mid s_i, o_i, z_i, p_i)$. For further implementation details, see Appendix B.2.

Inference Speed. Monolithic VLAs query the VLM at every action step [Kim et al., 2024;

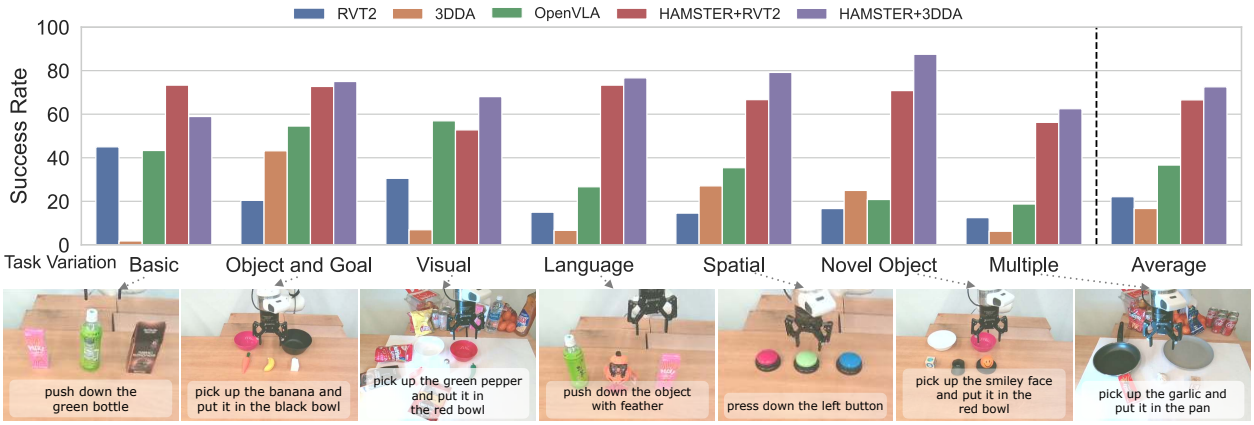


Figure 4.4: Depiction of quantitative real-world policy execution results on a real-world robot, evaluated across different axes of generalization and across both prehensile and non-prehensile tasks. Across all generalization axes, HAMSTER outperforms monolithic VLAs and the base 3D imitation learning policies.

Brohan et al., 2023a], which can be very expensive with large VLMs. For example, OpenVLA's 7B-parameter VLA only runs at 6Hz on an RTX 4090 [Kim et al., 2024]. Instead, HAMSTER's hierarchical design allows us to query the VLM only one or few times during an episode to generate 2D paths $\hat{p}$ that can be followed by low-level policy for multiple steps. Therefore, HAMSTER can be scaled to large VLM backbones without needing end-users to be concerned about inference speed.

4.5 Experimental Evaluation

We evaluate our approach in both simulation and real-world experiments to the following key questions. Do hierarchical VLAs:

- **Q1** Generalize behaviors to unseen scenarios with significant visual and semantic variation?
- **Q2** Achieve stronger cross-domain generalization than monolithic architectures?
- **Q3** Facilitate learning of non-prehensile and long-horizon tasks?
- **Q4** Exhibit strong demonstration efficiency?
- **Q5** Have improved visual + semantic reasoning due to hierarchy and VLM fine-tuning?

4.5.1 Real World Evaluation on Tabletop Manipulation

To answer **Q1**, our real-world evaluation experiments aim to test the generalization capability of hierarchical VLA models across significant semantic and visual variations. In particular, we consider a variant of HAMSTER that uses a VLM (VILA-1.5-13b [Lin et al., 2024]) finetuned on the data mixture in Section 4.4.1 as the high-level predictor, with two low-level 3D policy architectures - RVT-2 [Goyal et al., 2024] and 3D Diffuser Actor (3D-DA) [Ke et al., 2024] as choices of the low-level policy, as described in Section 4.4.2. The low-level 3D policies are trained with 320 episodes collected via teleoperation shown in Fig. 4.3. Importantly, the high-level VLM has not seen any in-domain data and is only finetuned on the off-domain data described in Section 4.4.1. This suggests that any generalization that the VLM shows result from cross-domain transfer.

Baseline comparisons. To answer **Q2**, we compare HAMSTER with a state-of-the-art monolithic VLA, OpenVLA [Kim et al., 2024] as well as non-VLM 3D policies, RVT-2 [Goyal et al., 2024] and 3D-DA [Ke et al., 2024]. For fair comparison, we finetune OpenVLA on the collected in-domain data described above since OpenVLA showed poor zero-shot generalization. The 3D policy (RVT-2, 3D-DA) baselines are trained with the same teleoperation data used to train the low-level policy in HAMSTER but without the intermediate 2D path representation from HAMSTER's VLM.

Finetuning OpenVLA with RLBench. To ensure our method's advantage over OpenVLA [Kim et al., 2024] is not solely due to RLBench data, we fine-tuned OpenVLA on the same RLBench dataset used for HAMSTER's VLM—1,000 episodes per task across 81 tasks (using only episodes with good front-camera visibility)—until achieving over 90% token accuracy [Kim et al., 2024]. We then fine-tuned this model on our tasks following the procedure in Appendix B.3.2. In real-world pick-and-place experiments (6 trials over 6 "Basic" tasks as shown in Table B.1), RLBench-finetuned OpenVLA averaged a success score of 0.54 versus 0.58 for the model without RLBench fine-tuning. This suggests that monolithic VLA architectures like OpenVLA gain little benefit from RLBench data, likely due to mismatches in action and observation spaces relative to the real-world setup.

Quantitative Results. Figure 4.4 summarizes our real-world results. To answer **Q3**, we evaluate across multiple task types, including 'pick and place,' and nonprehensile tasks such as 'press buttons' and 'knock down objects.' We also test generalization across various axes (**Q1**) – *obj and goal:* unseen object-goal combinations; *visual:* visual changes in table texture, lighting, distractor objects; *language:* unseen language instructions (e.g., candy $\rightarrow$ sweet object); *spatial:* unseen spatial object relationships in the instruction; *novel object:* unseen objects; and lastly, *multiple:* a combination of multiple variations. In total, we evaluate each model on 74 tasks for 222 total evaluations. Detailed results and the success score metric are provided in Appendix Table B.1.

Qualitative Eval on Various Tasks. In addition to the quantitative evaluation conducted for comparison with OpenVLA, we also present qualitative results that demonstrate how HAMSTER's hierarchical structure enables low-level policy models to generalize to more complex tasks. Figure B.1 illustrates the diverse tasks HAMSTER can handle, including unfolding a towel, opening and closing drawers, pressing buttons, wiping surfaces, and cleaning tables. These tasks present challenges such as varying lighting conditions, cluttered backgrounds, and semantic understanding requiring external world knowledge. Additionally, HAMSTER demonstrates the ability to perform long-horizon tasks—none of which are part of the in-domain training set used to train the policy model.

Overall, we find that HAMSTER significantly outperforms monolithic VLA models and (non-VLM) 3D policies by over **2x** and **3x**, respectively, on average. This is significant because this improved performance is in the face of considerable visual and semantic changes in the test setting, showing the ability of HAMSTER to generalize better than monolithic VLA models or non-VLM base models. We further group results by task type in Table B.3, where we see HAMSTER outperforms OpenVLA across all task types (pick and place, press button, and knock down). See Appendix B.3 for evaluation conditions, a task list, and other experiment details, and Appendix B.5

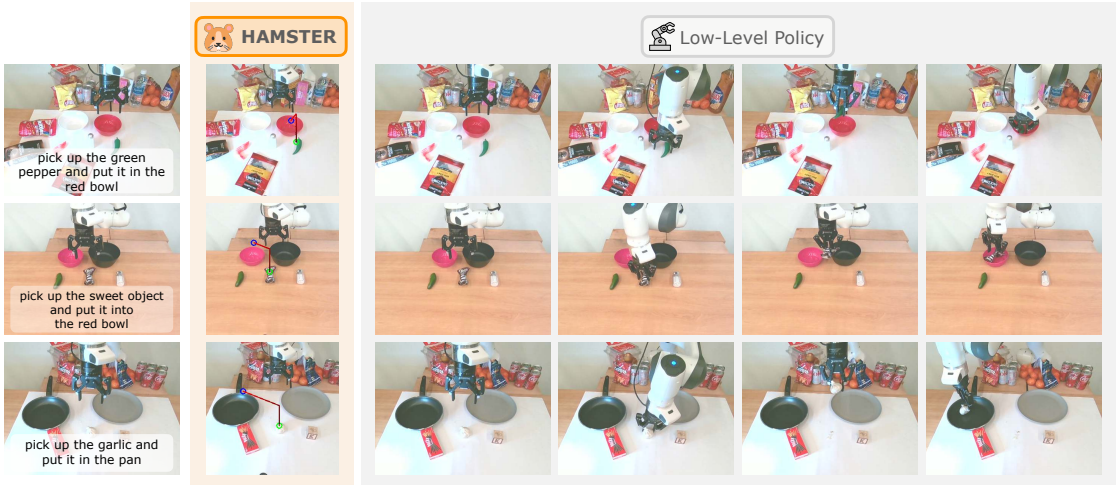


Figure 4.5: Example real-world HAMSTER rollouts demonstrate its strong performance in novel scenes achieved by leveraging VLMs' generalization capabilities and the robust execution of low-level 3D policies.

Method	Success
3D-DA	0.18 ± 0.10
HAMSTER+3D-DA (50%)	0.36 ± 0.04
HAMSTER+3D-DA	$\mathbf{0.43 \pm 0.05}$

Table 4.1: Results on Colosseum demonstrate that HAMSTER is data efficient, achieving 2X the success score of 3D-DA with just 50% of the data.

Method	**Original Camera**		**Novel Camera**	
	Success	Complete	Success	Complete
OpenVLA	0.60	0.30	0.23	0.00
HAMSTER+RVT2	0.83	0.70	0.73	0.40
HAMSTER+RVT2 (Concat)	**1.00**	**1.00**	**0.98**	**0.90**

Table 4.2: Real world results demonstrate HAMSTER generalizes to better to novel camera views (see Fig.Figure 4.6). We ran 10 trails and report averaged success score (success) described in Table B.1 and number of successful executions (complete).

for failure modes.

4.5.2 Simulation Evaluation

Overall Results. For further investigation into **Q1**, **Q2**, and **Q3**, we conducted a controlled simulation evaluation using Colosseum [Pumacay et al., 2024], which provides significant visual and semantic variations across pick-place and non-prehensile tasks. Pairing our high-level VLM with the state-of-the-art 3D-DA [Ke et al., 2024] policy on RLBench, we compared HAMSTER against a vanilla 3D-DA implementation without path guidance. As shown in Table 4.3 over 5 seeds, HAMSTER outperforms the vanilla approach by an average of 31%. This improvement stems from training with path-drawn images, which encourages the policy to focus on the path rather than extraneous visual features, thereby enhancing robustness to visual variations. We refer readers to Pumacay et al. [2024] for details on the variations and Appendix B.6 for further simulation experiment details.

HAMSTER with Fewer Demonstrations. We also test HAMSTER's ability to work well with limited demonstrations to answer **Q4**. We test on a subset of 5 Colosseum tasks[4]. Results in Table 4.1 demonstrate that HAMSTER+3D-DA with just 50% of the data still achieves 2x the success

[4]SLIDE_BLOCK_TO_TARGET, PLACE_WINE_AT_RACK_LOCATION, INSERT_ONTO_SQUARE_PEG, STACK_CUPS, SETUP_CHESS

	Avg.	no var	bac tex	cam pos	distractor	lig col	man obj col	man obj siz
3D-DA[Ke et al.]	0.35 ± 0.04	0.43 ± 0.06	0.34 ± 0.07	0.35 ± 0.11	0.39 ± 0.11	0.44 ± 0.13	0.41 ± 0.04	0.41 ± 0.11
HAMSTER (w 3D-DA)	$\mathbf{0.46} \pm 0.04$	$\mathbf{0.57} \pm 0.03$	$\mathbf{0.48} \pm 0.08$	$\mathbf{0.39} \pm 0.06$	$\mathbf{0.41} \pm 0.05$	$\mathbf{0.59} \pm 0.04$	$\mathbf{0.57} \pm 0.08$	$\mathbf{0.51} \pm 0.10$
	man obj tex	rec obj col	rec obj siz	rec obj tex	rlb and col	rlb var	tab col	tab tex
3D-DA[Ke et al.]	0.27 ± 0.04	0.34 ± 0.10	0.36 ± 0.05	0.36 ± 0.12	0.07 ± 0.03	0.45 ± 0.12	0.42 ± 0.06	0.23 ± 0.04
HAMSTER (w 3D-DA)	$\mathbf{0.48} \pm 0.06$	$\mathbf{0.48} \pm 0.05$	$\mathbf{0.40} \pm 0.05$	$\mathbf{0.56} \pm 0.09$	$\mathbf{0.11} \pm 0.10$	$\mathbf{0.58} \pm 0.04$	$\mathbf{0.56} \pm 0.03$	$\mathbf{0.35} \pm 0.07$

Table 4.3: Simulation evaluation of HAMSTER across different visual variations. We test vanilla 3D Diffuser Actor and HAMSTER across variations in Colosseum [Pumacay et al., 2024] and find that HAMSTER generalizes more effectively than 3D Diffuser Actor. Avg. indicates mean across variations, including no variation.

rate of standard 3D-DA, demonstrating that HAMSTER is demonstration-efficient for the downstream imitation learning tasks.

4.5.3 VLM Generalization Studies

Finally, we answer **Q5**: can HAMSTER's hierarchy enable superior visual and semantic reasoning?

Camera View Invariance. We test HAMSTER+RVT2 against OpenVLA from a new camera angle (Figure 4.6) across 10 pick-and-place trials using 6 training objects and 3 training containers to check HAMSTER's visual spatial reasoning. The results in Table 4.2 show that HAMSTER significantly outperforms OpenVLA and remains robust to new camera angles, benefiting from its VLM trained on diverse *off-domain* tasks across various viewpoints. Additionally, we compare HAMSTER+RVT2 (Concat), where instead of overlaying the path on the input RGB image, we modify RVT-2 to accept a 6-channel input by concatenating the original RGB image with a separate RGB image containing only the drawn path. We can easily apply this due to HAMSTER's hierarchical nature. Concatenated paths actually achieve the best performance, demonstrating the effectiveness of this path representation, though it is less general and not compatible with all imitation learning policy architectures (such as 3D-DA as it uses a pre-trained image encoder expecting 3 input channels). One possible explanation is that RVT2's virtual reprojection can fragment the 2D path when it is directly drawn on the image, making it harder for RVT2 to decode. By providing a dedicated path channel (via concatenation), path guidance is preserved more effectively.

Figure 4.6: Camera pos. for view invariance: old (right) and new (left).

VLM generalization We further demonstrate the benefit of HAMSTER's hierarchy by demonstrating that the VLM generalizes well to visually unique and semantically challenging tasks due to its off-domain fine-tuning. We visualize example HAMSTER path drawings in Figure 4.7, demonstrating HAMSTER's VLM itself effectively reasons semantically and visually for unseen tasks. We further investigate VLM performance in Appendix B.4.1, where we find that (1) HAMSTER outperforms zero-shot path generation from closed-source VLMs [Gu et al., 2023; Liang et al., 2023] and (2) that inclusion of simulation data improves HAMSTER's real-world performance. Both results point to the benefit of explicit hierarchy: off-domain VLM fine-tuning that

Method	Core VQA Benchmarks					Robustness / Probing Benchmarks								
	VQA^{v2}	GQA	VizWiz	SQA^{I}	VQA^{T}	POPE	MME	MMB	MMB^{CN}	SEED	SEED^{I}	LLaVA^{W}	MM-Vet	MMMU^{val}
VILA1.5-13B	82.8	64.3	62.6	**80.1**	**65.0**	**86.3**	1569.6	74.9	66.3	**65.1**	**72.6**	80.8	44.3	**37.9**
HAMSTER (ours)	**82.9**	**64.9**	**63.4**	78.0	61.4	85.8	**1588.4**	**75.3**	**67.1**	64.2	71.9	**81.2**	**44.4**	37.8

Table 4.4: Comparison across visual-language benchmarks, grouped into core VQA tasks (left of the vertical bar) and robustness/probing datasets (right). **HAMSTER (ours)** uses the same LLM and image resolution as VILA1.5-13B but is trained without curated vision-language finetuning. Best results are in **bold**. Benchmarks: VQA-v2 Goyal et al. [2017]; GQA Hudson and Manning [2019]; VizWiz Gurari et al. [2018]; SQA^{I}: ScienceQA-IMG Lu et al. [2022a]; VQA^{T}: TextVQA Singh et al. [2019]; POPE Li et al. [2023b]; MME Fu et al. [2024a]; MMB: MMBench Liu et al. [2024e]; MMB^{CN}: MMBench-Chinese Liu et al. [2024e]; SEED: SEED-Bench Li et al. [2023a]; SEED^{I}: SEED-Bench (Image) Li et al. [2023a]; LLaVA^{W}: LLaVA-Bench (In-the-Wild) Liu et al. [2023]; MM-Vet Yu et al. [2023a]; MMMU^{val} Yue et al. [2024].

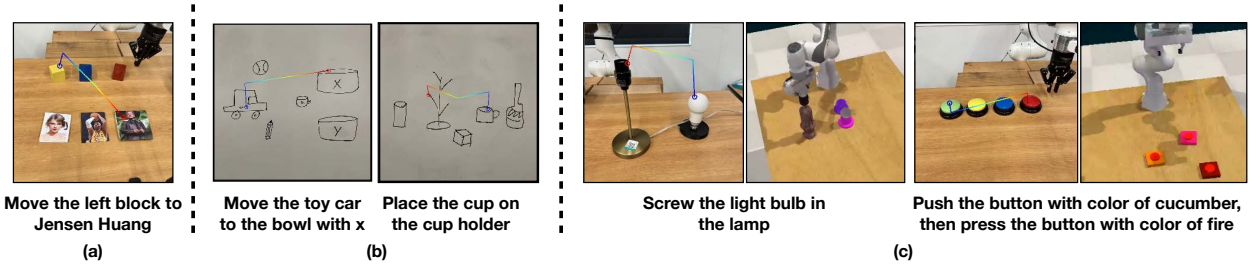


Figure 4.7: HAMSTER's VLM demonstrates strong generalization to unseen scenarios. From left to right: (a) leveraging world knowledge for user-specified tasks, (b) handling out-of-domain inputs like human-drawn sketches, and (c) transferring from diverse simulations to visually distinct real-world tasks. Blue-to-red lines indicate motion, with blue and red circles marking grasp and release points, respectively.

improves its performance. See Appendix B.4.1 for further details.

To quantitatively investigate whether HAMSTERretains broad commonsense knowledge, we evaluate it on 15 visual-question-answering and multimodal reasoning benchmarks. As shown in Table 4.4, HAMSTER matches the performance of VILA1.5-13B—which is HAMSTER's base model—demonstrating that our model behaves as a general-purpose VLM rather than a narrow, domain-specific system.

4.6 Limitations

This work represents an initial step towards developing versatile, hierarchical VLA methods, with numerous opportunities for future improvement and expansion. The proposed work only generates points in 2D space, without making native 3D predictions. This prevents the VLM from having true spatial 3D understanding. Moreover, the interface of just using 2D paths is a bandwidth limited one, which cannot communicate nuances such as force or rotation.

4.7 Discussion

In this chapter, we introduce HAMSTER, hierarchical VLA models that achieve robust generalization in robotic manipulation, consisting of a finetuned VLM that accurately predicts 2D paths and a low-level policy that learns to generate actions using the 2D paths. This two-step architec-

ture enables visual generalization and semantic reasoning across considerable domain shifts while enabling specialist policies, like ones conditioned on 3D inputs, to execute low-level actions.

This work represents an initial step towards developing versatile, hierarchical VLA methods. The proposed work only generates points in 2D space, without making native 3D predictions. This prevents the VLM from having true spatial 3D understanding. Moreover, the interface of just using 2D paths is a bandwidth limited one, which cannot communicate nuances such as force or rotation. In the future, investigating learnable intermediate interfaces is a promising direction. Moreover, training these VLMs directly from large-scale human video datasets would also be promising.

Chapter 5

Conclusion

This dissertation tackles the challenge of open-world generalization in robotic manipulation by leveraging off-domain data. Rather than relying on narrowly collected, task-specific robot data, we demonstrate that simulated environments, vision-language corpora, and pretrained models can enable more scalable and robust systems.

Our three primary contributions—**DeepIM**, **STOW**, and **HAMSTER**—form a progressive trajectory from low-level object state estimation to high-level semantic task reasoning. Each system tackles generalization at a different level of abstraction, offering complementary strategies for building open-world capable robotic systems.

DeepIM (Deep Iterative Matching for 6D Pose Estimation) addresses the problem of 6D object pose estimation using a novel iterative render-and-compare paradigm. Unlike prior methods that directly regress pose parameters, DeepIM refines initial pose estimates through iterative alignment between rendered object images and observed RGB inputs. Its design employs a disentangled SE(3) representation to decouple rotation and translation, improving both convergence and interpretability.

A key innovation of DeepIM is its reliance solely on RGB input, avoiding the need for depth sensors or handcrafted features. This makes it broadly applicable across consumer-grade hardware and diverse environments. By zooming in on object-centric patches and leveraging pixel-level feedback, DeepIM effectively "matches" appearance across views—even for unseen objects and viewpoints. This strategy parallels traditional ICP algorithms using point cloud but is learning-based and avoids explicit correspondences or feature engineering.

DeepIM has demonstrated strong performance across standard benchmarks, including LINEMOD, Occlusion LINEMOD, and YCB-Video, where it achieves state-of-the-art accuracy in pose refinement. However, the assumption that objects are rigid and have well-defined geometries limits its applicability to real-world scenarios involving deformable or articulated objects. As robotics applications expand to such settings, new representations beyond static 6D poses will be necessary.

To move beyond rigid object representations and support broader scene-level perception, **STOW** (Discrete-Frame Segmentation and Tracking of Unseen Objects for Warehouse Picking Robots) introduces a discrete-frame approach to instance segmentation and temporal tracking. This system enables robots to understand and interact with multi-object environments in which object identi-

ties and spatial configurations evolve over time.

STOW is trained entirely on synthetic data, yet exhibits strong generalization to real-world warehouse environments. This success is enabled by a carefully designed architecture that decouples instance recognition from tracking logic, using multi-frame attention and object-centric embeddings for cross-frame correspondence. The system produces temporally consistent object masks even under clutter and occlusion, facilitating robust downstream manipulation.

STOW was deployed in a real robotic warehouse system, demonstrating the practicality of sim-to-real transfer when paired with appropriate inductive biases. However, STOW is limited to perception tasks—it does not interpret task goals or language, and cannot reason about interactions that require world knowledge or contextual understanding. It excels at answering "what" and "where" but remains disconnected from the higher-level "why" and "how" of manipulation.

To bridge this gap, we introduce **HAMSTER** (Hierarchical Action Models for Open-World Robot Manipulation), which tackles task-level generalization by integrating pretrained vision-language models (VLMs) with end-effector control via an intermediate abstraction: *spatial sketch trajectories*. These 2D trajectories represent a concise, interpretable summary of what the robot should do in a given task, and serve as a bridge between high-level semantic goals and low-level execution.

Unlike traditional architectures that entangle task understanding and control, HAMSTER adopts a *dual-system* architecture inspired by Kahneman's *Thinking, Fast and Slow* Kahneman [2011]. The "slow" system—a VLM—interprets instructions and visual context to produce a spatial plan, while the "fast" system—a reactive controller—executes this plan in closed-loop. This separation allows each component to be developed and optimized independently, enabling modularity, sample efficiency, and generalization across tasks.

Crucially, HAMSTER shows that vision-language models pretrained on internet-scale data can be repurposed for embodied reasoning when paired with appropriate abstractions. By working in 2D image space rather than requiring dense 3D modeling, the system avoids brittle assumptions and exhibits robustness to variations in geometry, lighting, and phrasing. We demonstrate its ability to generalize across a range of tasks and scenes—including both pick-and-place and more challenging non-prehensile tasks involving deformable or articulated objects.

Nonetheless, managing the interaction between the "fast" and "slow" systems remains a key challenge, particularly in tasks requiring real-time responsiveness or fine-grained adjustments. Furthermore, reliance on VLMs introduces risks of grounding failures, hallucinated plans, and overconfident outputs. Addressing these issues will require stronger feedback integration, uncertainty modeling, and grounding-aware learning.

Together, DeepIM, STOW, and HAMSTER illustrate a progression of ideas for building open-world robotic systems—from low-level geometric reasoning, to object-centric visual tracking, to language-conditioned semantic planning. Each system demonstrates that off-domain data can expand the capabilities of robots to operate in the complexity and diversity of the real world.

5.1 Future Work

Moving forward, several important research directions arise from the limitations and insights uncovered through DeepIM, STOW, and HAMSTER.

First, at the semantic level, relying solely on supervised fine-tuning is insufficient due to its dependence on narrow, task-specific annotations. We propose introducing a pretraining phase tailored to robot-relevant objectives as a scalable alternative. This raises several open questions: What types of internet-scale or synthetic data best support robotic instruction following? How can visual-linguistic tasks be structured to promote actionable understanding rather than passive recognition or captioning? Pretraining objectives such as world modeling or hand pose prediction may enable models to not only interpret the world but also anticipate the outcomes of their actions.

Second, in bridging semantic plans and low-level execution, spatial sketches offer a promising but limited abstraction. They are lightweight and generalizable, yet may fall short for tasks requiring contact-rich reasoning, force feedback, or dynamic corrections. Future work could expand the action representation space—e.g., from 2D to 3D motion plans, symbolic programs, or hierarchical subgoals—while maintaining interpretability. Simultaneously, robustness must be improved through uncertainty modeling, counterfactual data augmentation, multimodal feedback, and controller ensembles.

Third, lifelong learning and adaptation are essential for closing the sim-to-real and pretrain-to-deploy gaps. Even large pretrained models cannot anticipate every object, layout, phrasing, or embodiment encountered in deployment. Online adaptation methods—such as self-supervised skill refinement, language-conditioned corrective feedback, and human-in-the-loop tuning—will be key to improving real-world reliability. In particular, data from teleoperation and recovery from failure events offers a rich source of corrective supervision.

Fourth, generalization at scale demands better infrastructure for evaluation. Existing benchmarks often focus on narrow domains, limiting their ability to reflect open-world complexity. There is a pressing need for compositional, diverse, and scalable benchmarks that assess performance across object types, linguistic variations, environmental settings, and robot embodiments. Such benchmarks would provide more rigorous insights into generalization capacity and accelerate progress across the field.

Finally, as robots gain autonomy, we must address safety, transparency, and ethical alignment. Models like HAMSTER, which leverage large pretrained VLMs, inherit both their strengths and biases. Future work should develop methods for auditing and constraining these models, particularly when they interact with humans or make decisions in physical environments. Techniques such as interpretable plan visualization, goal verification, and confidence-aware fallback policies can improve trustworthiness and deployment safety.

In summary, advancing open-world robotics will require the integration of better semantic abstractions, broader training data, stronger feedback mechanisms, and scalable adaptation strategies. By pursuing these directions, we move closer to developing reliable, general-purpose robotic systems capable of operating in complex, real-world environments.

Bibliography

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

ambientCG. 2023. Creative Commons Zero (CC0). `https://ambientcg.com`.

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. 2023. Qwen-vl: A frontier large vision-language model with versatile abilities. *arXiv preprint arXiv:2308.12966*.

Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. 2008. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359.

Paul J Besl and Neil D McKay. 1992. Method for registration of 3-d shapes. In *Sensor Fusion IV: Control Paradigms and Data Structures*, volume 1611, pages 586–607. International Society for Optics and Photonics.

Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. 2016. Simple online and realtime tracking. In *2016 IEEE international conference on image processing (ICIP)*, pages 3464–3468.

Homanga Bharadhwaj, Roozbeh Mottaghi, Abhinav Gupta, and Shubham Tulsiani. 2024. Track2act: Predicting point tracks from internet videos enables diverse zero-shot robot manipulation. *arXiv preprint arXiv:2405.01527*.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. 2024. *pi*_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*.

Eric Brachmann, Alexander Krull, Frank Michel, Stefan Gumhold, Jamie Shotton, and Carsten Rother. 2014. Learning 6D object pose estimation using 3D object coordinates. In *European Conference on Computer Vision (ECCV)*.

Eric Brachmann, Frank Michel, Alexander Krull, Michael Ying Yang, Stefan Gumhold, and Carsten Rother. 2016. Uncertainty-driven 6D pose estimation of objects and scenes from a single RGB image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3364–3372.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. 2023a. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. 2022. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*.

Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. 2023b. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on robot learning*, pages 287–318. PMLR.

Minwoo Byeon, Beomhee Park, Haecheon Kim, Sungjun Lee, Woonhyuk Baek, and Saehoon Kim. 2022. Coyo-700m: Image-text pair dataset. `https://github.com/kakaobrain/coyo-dataset`.

Sergi Caelles, Jordi Pont-Tuset, Federico Perazzi, Alberto Montes, Kevis-Kokitsi Maninis, and Luc Van Gool. 2019. The 2019 davis challenge on vos: Unsupervised multi-object segmentation. *arXiv:1905.00737*.

Berk Calli, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. 2015. The ycb object and model set: Towards common benchmarks for manipulation research. In *Advanced Robotics (ICAR), 2015 International Conference on*, pages 510–517. IEEE.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. 2020. End-to-end object detection with transformers. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*, pages 213–229. Springer.

J. Carreira, P. Agrawal, K. Fragkiadaki, and J. Malik. 2016. Human pose estimation with iterative error feedback. In *IEEE conference on Computer Vision and Pattern Recognition (CVPR)*.

Jianqiu Chen, Mingshan Sun, Tianpeng Bao, Rui Zhao, Liwei Wu, and Zhenyu He. 2023. 3d model-based zero-shot pose estimation pipeline. *ArXiv*, abs/2305.17934.

Bowen Cheng, Anwesa Choudhuri, Ishan Misra, Alexander Kirillov, Rohit Girdhar, and Alexander G Schwing. 2021. Mask2former for video instance segmentation. *arXiv preprint arXiv:2112.10764*.

Bowen Cheng, Ishan Misra, Alexander G Schwing, Alexander Kirillov, and Rohit Girdhar. 2022. Masked-attention mask transformer for universal image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1290–1299.

Ho Kei Cheng and Alexander G Schwing. 2022. Xmem: Long-term video object segmentation with an atkinson-shiffrin memory model. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXVIII*, pages 640–658. Springer.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. 2023. Diffusion policy: Visuomotor policy learning via action diffusion. In *Robotics: Science and Systems XIX, Daegu, Republic of Korea, July 10-14, 2023*.

Alvaro Collet, Manuel Martinez, and Siddhartha S Srinivasa. 2011. The MOPED framework: Object recognition and pose estimation for manipulation. *International Journal of Robotics Research (IJRR)*, 30(10):1284–1306.

Gabriele Costante and Thomas Alessandro Ciarfuglia. 2018. LS-VO: Learning dense optical subspace for robust visual odometry estimation. *IEEE Robotics and Automation Letters*, 3(3):1735–1742.

Navneet Dalal and Bill Triggs. 2005. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 886–893.

Xinke Deng, Arsalan Mousavian, Yu Xiang, Fei Xia, Timothy Bretl, and Dieter Fox. 2019. Poserbpf: A rao-blackwellized particle filter for 6d object pose tracking. In *Robotics: Science and Systems (RSS)*.

Carl Doersch, Yi Yang, Mel Vecerik, Dilara Gokay, Ankush Gupta, Yusuf Aytar, Joao Carreira, and Andrew Zisserman. 2023. Tapir: Tracking any point with per-frame initialization and temporal refinement. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10061–10072.

Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. 2015. Flownet: Learning optical flow with convolutional networks. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2758–2766.

David H Douglas and Thomas K Peucker. 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica*, 10(2):112–122.

Laura Downs, Anthony Francis, Nate Koenig, Brandon Kinman, Ryan Hickman, Krista Reymann, Thomas B McHugh, and Vincent Vanhoucke. 2022. Google scanned objects: A high-quality dataset of 3d scanned household items. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 2553–2560. IEEE.

Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. 2023. Palm-e: An embodied multimodal language model. In *International Conference on Machine Learning*, pages 8469–8488. PMLR.

Jiafei Duan, Wentao Yuan, Wilbert Pumacay, Yi Ru Wang, Kiana Ehsani, Dieter Fox, and Ranjay Krishna. 2024. Manipulate-anything: Automating real-world robots using vision-language models. *arXiv preprint arXiv:2406.18915*.

Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. 2010. The pascal visual object classes (voc) challenge. *IEEE International Journal of Computer Vision (ICCV)*, 88(2):303–338.

Adam Fishman, Adithyavairavan Murali, Clemens Eppner, Bryan Peele, Byron Boots, and Dieter Fox. 2022. Motion policy networks. In *Conference on Robot Learning, CoRL 2022, 14-18 December 2022, Auckland, New Zealand*, volume 205 of *Proceedings of Machine Learning Research*, pages 967–977. PMLR.

Peter R Florence, Lucas Manuelli, and Russ Tedrake. 2018. Dense object nets: Learning dense visual object descriptors by and for robotic manipulation. *arXiv preprint arXiv:1806.08756*.

Peter Raymond Florence. 2020. *Dense visual learning for robot manipulation*. Ph.D. thesis, Massachusetts Institute of Technology.

Chaoyou Fu, Yi-Fan Zhang, Shukang Yin, Bo Li, Xinyu Fang, Sirui Zhao, Haodong Duan, Xing Sun, Ziwei Liu, Liang Wang, et al. 2024a. Mme-survey: A comprehensive survey on evaluation of multimodal llms. *arXiv preprint arXiv:2411.15296*.

Zipeng Fu, Tony Z Zhao, and Chelsea Finn. 2024b. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. *arXiv preprint arXiv:2401.02117*.

Shubhika Garg and Vidit Goel. 2021. Mask selection and propagation for unsupervised video object segmentation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1680–1690.

Mathieu Garon, Pierre-Olivier Boulet, Jean-Philippe Doironz, Luc Beaulieu, and Jean-François Lalonde. 2016. Real-time high resolution 3d data on the hololens. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct)*, pages 189–191. IEEE.

Mathieu Garon and Jean-François Lalonde. 2017. Deep 6-dof tracking. *IEEE transactions on visualization and computer graphics*, 23(11):2410–2418.

Theophile Gervet, Zhou Xian, Nikolaos Gkanatsios, and Katerina Fragkiadaki. 2023. Act3d: Infinite resolution action detection transformer for robotic manipulation. *arXiv preprint arXiv:2306.17817*.

Ross Girshick. 2015. Fast R-CNN. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448.

Anas Gouda, Abraham Ghanem, and Christopher Reining. 2022. Dopose-6d dataset for object segmentation and 6d pose estimation. *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 477–483.

Ankit Goyal, Valts Blukis, Jie Xu, Yijie Guo, Yu-Wei Chao, and Dieter Fox. 2024. Rvt2: Learning precise manipulation from few demonstrations. *RSS*.

Ankit Goyal, Arsalan Mousavian, Chris Paxton, Yu-Wei Chao, Brian Okorn, Jia Deng, and Dieter Fox. 2022. Ifor: Iterative flow minimization for robotic object rearrangement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14787–14797.

Ankit Goyal, Jie Xu, Yijie Guo, Valts Blukis, Yu-Wei Chao, and Dieter Fox. 2023. Rvt: Robotic view transformer for 3d object manipulation. In *Conference on Robot Learning*, pages 694–710. PMLR.

Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. 2017. Making the v in vqa matter: Elevating the role of image understanding in visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6904–6913.

Kristen Grauman, Andrew Westbury, Eugene Byrne, Zachary Chavis, Antonino Furnari, Rohit Girdhar, Jackson Hamburger, Hao Jiang, Miao Liu, Xingyu Liu, et al. 2022. Ego4d: Around the world in 3,000 hours of egocentric video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18995–19012.

Markus Grotz, Soofiyan Atar, Yi Li, Paolo Torrado, Boling Yang, Nick Walker, Michael Murray, Maya Cakmak, and Joshua Smith. 2023. Towards robustly picking unseen objects from densely packed shelves. In *Workshop on Perception and Manipulation Challenges for Warehouse Automation*.

Chunhui Gu and Xiaofeng Ren. 2010. Discriminative mixture-of-templates for viewpoint classification. In *European Conference on Computer Vision (ECCV)*, pages 408–421.

Jiayuan Gu, Sean Kirmani, Paul Wohlhart, Yao Lu, Montserrat Gonzalez Arenas, Kanishka Rao, Wenhao Yu, Chuyuan Fu, Keerthana Gopalakrishnan, Zhuo Xu, Priya Sundaresan, Peng Xu, Hao Su, Karol Hausman, Chelsea Finn, Quan Vuong, and Ted Xiao. 2023. Rt-trajectory: Robotic task generalization via hindsight trajectory sketches.

Danna Gurari, Qing Li, Abigale J Stangl, Anhong Guo, Chi Lin, Kristen Grauman, Jiebo Luo, and Jeffrey P Bigham. 2018. Vizwiz grand challenge: Answering visual questions from blind people. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3608–3617.

Ankur Handa, Arthur Allshire, Viktor Makoviychuk, Aleksei Petrenko, Ritvik Singh, Jingzhou Liu, Denys Makoviichuk, Karl Van Wyk, Alexander Zhurkevich, Balakumar Sundaralingam, et al. 2023. Dextreme: Transfer of agile in-hand manipulation from simulation to reality. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5977–5984. IEEE.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Miran Heo, Sukjun Hwang, Seoung Wug Oh, Joon-Young Lee, and Seon Joo Kim. 2022. Vita: Video instance segmentation via object token association. *arXiv preprint arXiv:2206.04403*.

S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, , and N. Navab. 2012a. Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes. In *Asian Conference on Computer Vision (ACCV)*.

Stefan Hinterstoisser, Cedric Cagniart, Slobodan Ilic, Peter Sturm, Nassir Navab, Pascal Fua, and Vincent Lepetit. 2012b. Gradient response maps for real-time detection of textureless objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 34(5):876–888.

Stefan Hinterstoisser, Vincent Lepetit, Naresh Rajkumar, and Kurt Konolige. 2016. Going further with point pair features. In *European Conference on Computer Vision (ECCV)*, pages 834–848.

Tomáš Hodan, Pavel Haluza, Štepán Obdržálek, Jiri Matas, Manolis Lourakis, and Xenophon Zabulis. 2017. T-less: An rgb-d dataset for 6d pose estimation of texture-less objects. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 880–888. IEEE.

De-An Huang, Zhiding Yu, and Anima Anandkumar. 2022. Minvis: A minimal video instance segmentation framework without video-based training. *arXiv preprint arXiv:2208.02245*.

Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. 2023a. Voxposer: Composable 3d value maps for robotic manipulation with language models. In *Conference on Robot Learning*, pages 540–562. PMLR.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. 2023b. Inner monologue: Embodied reasoning through planning with language models. In *Conference on Robot Learning*, pages 1769–1782. PMLR.

Drew A Hudson and Christopher D Manning. 2019. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6700–6709.

Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. 2020. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026.

Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. 2022. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*, pages 991–1002. PMLR.

Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. 2023. Vima: General robot manipulation with multimodal prompts. In *International Conference on Machine Learning*.

Andrew E Johnson and Martial Hebert. 1999. Using spin images for efficient object recognition in cluttered 3d scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, (5):433–449.

Frédéric Jurie and Michel Dhome. 2001. Real time 3d template matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages I–I.

Daniel Kahneman. 2011. *Thinking, fast and slow*. macmillan.

Nikita Karaev, Ignacio Rocco, Benjamin Graham, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. 2025. Cotracker: It is better to track together. In *European Conference on Computer Vision*, pages 18–35. Springer.

Tsung-Wei Ke, Nikolaos Gkanatsios, and Katerina Fragkiadaki. 2024. 3d diffuser actor: Policy diffusion with 3d scene representations. In *First Workshop on Vision-Language Models for Navigation and Manipulation at ICRA 2024*.

Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. 2017. Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1521–1529.

Alex Kendall and Roberto Cipolla. 2017. Geometric loss functions for camera pose regression with deep learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, et al. 2024. Droid: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. 2024. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. 2023. Segment anything. *arXiv preprint arXiv:2304.02643*.

Alexander Krull, Eric Brachmann, Frank Michel, Michael Ying Yang, Stefan Gumhold, and Carsten Rother. 2015. Learning analysis-by-synthesis for 6D pose estimation in RGB-D images. In *IEEE International Conference on Computer Vision (ICCV)*, pages 954–962.

Yuxuan Kuang, Junjie Ye, Haoran Geng, Jiageng Mao, Congyue Deng, Leonidas Guibas, He Wang, and Yue Wang. 2024. Ram: Retrieval-based affordance transfer for generalizable zero-shot robotic manipulation. *arXiv preprint arXiv:2407.04689*.

Harold W Kuhn. 1955. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. 2023. Reward design with language models. In *The Eleventh International Conference on Learning Representations*.

Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. 2020. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986.

Bohao Li, Rui Wang, Guangzhi Wang, Yuying Ge, Yixiao Ge, and Ying Shan. 2023a. Seed-bench: Benchmarking multimodal llms with generative comprehension. *arXiv preprint arXiv:2307.16125*.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, et al. 2024. Evaluating real-world robot manipulation policies in simulation. *arXiv preprint arXiv:2405.05941*.

Yifan Li, Yifan Du, Kun Zhou, Jinpeng Wang, Wayne Xin Zhao, and Ji-Rong Wen. 2023b. Evaluating object hallucination in large vision-language models. *arXiv preprint arXiv:2305.10355*.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. 2023. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE.

Chen-Hsuan Lin and Simon Lucey. 2017. Inverse compositional spatial transformer networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2568–2576.

Ji Lin, Hongxu Yin, Wei Ping, Pavlo Molchanov, Mohammad Shoeybi, and Song Han. 2024. Vila: On pre-training for visual language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 26689–26699.

Kevin Lin, Christopher Agia, Toki Migimatsu, Marco Pavone, and Jeannette Bohg. 2023. Text2motion: From natural language instructions to feasible plans. *Autonomous Robots*, 47(8):1345–1365.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024a. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Fangchen Liu, Kuan Fang, Pieter Abbeel, and Sergey Levine. 2024b. Moka: Open-vocabulary robotic manipulation through mark-based visual prompting. *arXiv preprint arXiv:2403.03174*.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2024c. Improved baselines with visual instruction tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 26296–26306.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2024d. Visual instruction tuning. *Advances in neural information processing systems*, 36.

Ming-Yu Liu, Oncel Tuzel, Ashok Veeraraghavan, and Rama Chellappa. 2010. Fast directional chamfer matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1696–1703.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. Ssd: Single shot multibox detector. In *European Conference on Computer Vision (ECCV)*, pages 21–37.

Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, et al. 2024e. Mmbench: Is your multi-modal model an all-around player? In *European conference on computer vision*, pages 216–233. Springer.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440.

David G Lowe. 1999. Object recognition from local scale-invariant features. In *IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 1150–1157.

David G Lowe. 2004. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60:91–110.

Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. 2022a. Learn to explain: Multimodal reasoning via thought chains for science question answering. *Advances in Neural Information Processing Systems*, 35:2507–2521.

Pan Lu, Swaroop Mishra, Tony Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. 2022b. Learn to explain: Multimodal reasoning via thought chains for science question answering. In *The 36th Conference on Neural Information Processing Systems (NeurIPS)*.

Yangxiao Lu, Yuqiao Chen, Nicholas Ruozzi, and Yu Xiang. 2022c. Mean shift mask transformer for unseen object instance segmentation. *arXiv preprint arXiv:2211.11679*.

Yangxiao Lu, Ninad Khargonkar, Zesheng Xu, Charles Averill, Kamalesh Palanisamy, Kaiyu Hang, Yunhui Guo, Nicholas Ruozzi, and Yu Xiang. 2023. Self-supervised unseen object instance segmentation via long-term robot interaction. *arXiv preprint arXiv:2302.03793*.

Jonathon Luiten, Idil Esen Zulfikar, and Bastian Leibe. 2020. Unovost: Unsupervised offline video object segmentation and tracking. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 2000–2009.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Eureka: Human-level reward design via coding large language models. In *The Twelfth International Conference on Learning Representations*.

Yecheng Jason Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. 2023. Vip: Towards universal visual reward and representation via value-implicit pre-training. In *The Eleventh International Conference on Learning Representations*.

Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. 2023. Mimicgen: A data generation system for scalable robot learning using human demonstrations. In *Conference on Robot Learning*, pages 1820–1864. PMLR.

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. 2021. What matters in learning from offline human demonstrations for robot manipulation. In *Conference on Robot Learning (CoRL)*.

Fabian Manhardt, Wadim Kehl, Nassir Navab, and Federico Tombari. 2018. Deep model-based 6d pose refinement in rgb. In *European Conference on Computer Vision (ECCV)*, pages 800–815.

Nicolas Mellado, Dror Aiger, and Niloy J Mitra. 2014. Super 4pcs fast global pointcloud registration via smart indexing. In *Computer Graphics Forum*, volume 33, pages 205–215. Wiley Online Library.

Ajmal S Mian, Mohammed Bennamoun, and Robyn Owens. 2006. Three-dimensional model-based object recognition and segmentation in cluttered scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 28(10):1584–1601.

Frank Michel, Alexander Kirillov, Erix Brachmann, Alexander Krull, Stefan Gumhold, Bogdan Savchynskyy, and Carsten Rother. 2017. Global hypothesis generation for 6D object pose estimation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Matthias Minderer, Alexey Gritsenko, Austin Stone, Maxim Neumann, Dirk Weissenborn, Alexey Dosovitskiy, Aravindh Mahendran, Anurag Arnab, Mostafa Dehghani, Zhuoran Shen, et al. 2022. Simple open-vocabulary object detection. In *European Conference on Computer Vision*, pages 728–755. Springer.

Matthias Minderer, Alexey A. Gritsenko, and Neil Houlsby. 2023. Scaling open-vocabulary object detection. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Košecká. 2017. 3D bounding box estimation using deep learning and geometry. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5632–5640.

Arsalan Mousavian, Clemens Eppner, and Dieter Fox. 2019. 6-dof graspnet: Variational grasp generation for object manipulation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2901–2910.

Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. 2023. R3m: A universal visual representation for robot manipulation. In *Conference on Robot Learning*, pages 892–909. PMLR.

Soroush Nasiriany, Sean Kirmani, Tianli Ding, Laura Smith, Yuke Zhu, Danny Driess, Dorsa Sadigh, and Ted Xiao. 2024a. Rt-affordance: Affordances are versatile intermediate representations for robot manipulation. *arXiv preprint arXiv:2411.02704*.

Soroush Nasiriany, Fei Xia, Wenhao Yu, Ted Xiao, Jacky Liang, Ishita Dasgupta, Annie Xie, Danny Driess, Ayzaan Wahid, Zhuo Xu, et al. 2024b. Pivot: Iterative visual prompting elicits actionable knowledge for vlms. In *International Conference on Machine Learning*.

David Nistér. 2005. Preemptive ransac for live structure and motion estimation. *Machine Vision and Applications*, 16(5):321–329.

Dantong Niu, Yuvan Sharma, Giscard Biamby, Jerome Quenum, Yutong Bai, Baifeng Shi, Trevor Darrell, and Roei Herzig. 2024. LLARVA: Vision-action instruction tuning enhances robot learning. In *8th Annual Conference on Robot Learning*.

M. Oberweger, P. Wohlhart, and V. Lepetit. 2015. Training a feedback loop for hand pose estimation. In *IEEE International Conference on Computer Vision (ICCV)*.

Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. 2024. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0.

Simone Parisi, Aravind Rajeswaran, Senthil Purushwalkam, and Abhinav Gupta. 2022. The unsurprising effectiveness of pre-trained vision models for control. In *international conference on machine learning*, pages 17359–17371. PMLR.

Jordi Pont-Tuset, Federico Perazzi, Sergi Caelles, Pablo Arbeláez, Alex Sorkine-Hornung, and Luc Van Gool. 2017. The 2017 davis challenge on video object segmentation. *arXiv preprint arXiv:1704.00675*.

Wilbert Pumacay, Ishika Singh, Jiafei Duan, Ranjay Krishna, Jesse Thomason, and Dieter Fox. 2024. The colosseum: A benchmark for evaluating generalization for robotic manipulation. *arXiv preprint arXiv:2402.08191*.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. *IEEE Computer Vision and Pattern Recognition (CVPR)*, 1(2):4.

Jiyang Qi, Yan Gao, Yao Hu, Xinggang Wang, Xiaoyu Liu, Xiang Bai, Serge Belongie, Alan Yuille, Philip HS Torr, and Song Bai. 2022. Occluded video instance segmentation: A benchmark. *International Journal of Computer Vision*, 130(8):2022–2039.

Mahdi Rad and Vincent Lepetit. 2017. BB8: A scalable, accurate, robust to partial occlusion method for predicting the 3D poses of challenging objects without using depth. In *IEEE International Conference on Computer Vision (ICCV)*.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR.

Ilija Radosavovic, Baifeng Shi, Letian Fu, Ken Goldberg, Trevor Darrell, and Jitendra Malik. 2023. Robot learning with sensorimotor pre-training. In *Conference on Robot Learning*, pages 683–693. PMLR.

Urs Ramer. 1972. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244–256.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788.

S. Ren, K. He, R. Girshick, and J. Sun. 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS)*.

Carsten Rother, Tom Minka, Andrew Blake, and Vladimir Kolmogorov. 2006. Cosegmentation of image pairs by histogram matching-incorporating a global constraint into mrfs. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 993–1000. IEEE.

Fred Rothganger, Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. 2006. 3D object modeling and recognition using local affine-invariant image descriptors and multi-view spatial constraints. *International Journal of Computer Vision (IJCV)*, 66(3):231–259.

Szymon Rusinkiewicz and Marc Levoy. 2001. Efficient variants of the icp algorithm. In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pages 145–152. IEEE.

Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. 2009. Fast point feature histograms (fpfh) for 3d registration. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3212–3217. Citeseer.

Joaquim Salvi, Carles Matabosch, David Fofi, and Josep Forest. 2007. A review of recent range image registration methods with accuracy evaluation. *Image and Vision computing*, 25(5):578–596.

Aseem Saxena, Harit Pandya, Gourav Kumar, Ayush Gaud, and K Madhava Krishna. 2017. Exploring convolutional networks for end-to-end visual servoing. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3817–3823.

Rutav M Shah and Vikash Kumar. 2021. Rrl: Resnet as representation for reinforcement learning. In *International Conference on Machine Learning*, pages 9465–9476. PMLR.

Zejiang Shen, Kyle Lo, Lucy Lu Wang, Bailey Kuehl, Daniel S. Weld, and Doug Downey. 2021. Incorporating visual layout structures for scientific text classification. *ArXiv*, abs/2106.00676.

Jamie Shotton, Ben Glocker, Christopher Zach, Shahram Izadi, Antonio Criminisi, and Andrew Fitzgibbon. 2013. Scene coordinate regression forests for camera relocalization in RGB-D images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2930–2937.

Mohit Shridhar, Lucas Manuelli, and Dieter Fox. 2023. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Conference on Robot Learning*, pages 785–799. PMLR.

Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Amanpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and Marcus Rohrbach. 2019. Towards vqa models that can read. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8317–8326.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE.

Sumedh Anand Sontakke, Jesse Zhang, Séb Arnold, Karl Pertsch, Erdem Biyik, Dorsa Sadigh, Chelsea Finn, and Laurent Itti. 2023. Roboclip: One demonstration is enough to learn robot policies. In *NeurIPS*.

Austin Stone, Ted Xiao, Yao Lu, Keerthana Gopalakrishnan, Kuang-Huei Lee, Quan Vuong, Paul Wohlhart, Sean Kirmani, Brianna Zitkovich, Fei Xia, et al. 2023. Open-world object manipulation using pre-trained vision-language models. In *Conference on Robot Learning*, pages 3397–3417. PMLR.

Priya Sundaresan, Suneel Belkhale, Dorsa Sadigh, and Jeannette Bohg. 2023. Kite: Keypoint-conditioned policies for semantic manipulation. In *Conference on Robot Learning*, pages 1006–1021. PMLR.

Martin Sundermeyer, Zoltan-Csaba Marton, Maximilian Durner, Manuel Brucker, and Rudolph Triebel. 2018. Implicit 3d orientation learning for 6d object detection from rgb images. In *European Conference on Computer Vision (ECCV)*, pages 699–715.

Martin Sundermeyer, Arsalan Mousavian, Rudolph Triebel, and Dieter Fox. 2021. Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13438–13444. IEEE.

Richard S. Sutton. 2019. The bitter lesson. `http://www.incompleteideas.net/IncIdeas/BitterLesson.html`. Accessed: 2025-05-16.

Gary KL Tam, Zhi-Quan Cheng, Yu-Kun Lai, Frank C Langbein, Yonghuai Liu, David Marshall, Ralph R Martin, Xian-Fang Sun, and Paul L Rosin. 2013. Registration of 3d point clouds and meshes: a survey from rigid to nonrigid. *IEEE transactions on visualization and computer graphics*, 19(7):1199–1217.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. 2024. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*.

Zachary Teed and Jia Deng. 2020. Raft: Recurrent all-pairs field transforms for optical flow. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, pages 402–419. Springer.

Bugra Tekin, Sudipta N Sinha, and Pascal Fua. 2017. Real-time seamless single shot 6D object pose prediction. *arXiv preprint arXiv:1711.08848*.

Pascal Willy Theiler, Jan Dirk Wegner, and Konrad Schindler. 2015. Globally consistent registration of terrestrial laser scans via graph optimization. *ISPRS Journal of Photogrammetry and Remote Sensing*, 109:126–138.

Henning Tjaden, Ulrich Schwanecke, and Elmar Schömer. 2017. Real-time monocular pose estimation of 3D objects using temporally consistent local color histograms. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 124–132.

Federico Tombari, Samuele Salti, and Luigi Di Stefano. 2010. Unique signatures of histograms for local surface description. In *European Conference on Computer Vision (ECCV)*, pages 356–369. Springer.

Marcel Torne, Anthony Simeonov, Zechu Li, April Chan, Tao Chen, Abhishek Gupta, and Pulkit Agrawal. 2024. Reconciling reality through simulation: A real-to-sim-to-real approach for robust manipulation. *Robotics: Science and Systems*.

Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. 2018. Deep object pose estimation for semantic robotic grasping of household objects. In *Conference on Robot Learning*, pages 306–316.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need.

Homer Walke, Kevin Black, Abraham Lee, Moo Jin Kim, Max Du, Chongyi Zheng, Tony Zhao, Philippe Hansen-Estruch, Quan Vuong, Andre He, Vivek Myers, Kuan Fang, Chelsea Finn, and Sergey Levine. 2023. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning (CoRL)*.

Chen Wang, Danfei Xu, Yuke Zhu, Roberto Martín-Martín, Cewu Lu, Li Fei-Fei, and Silvio Savarese. 2019. Densefusion: 6d object pose estimation by iterative dense fusion. *arXiv preprint arXiv:1901.04780*.

Qianqian Wang, Yen-Yu Chang, Ruojin Cai, Zhengqi Li, Bharath Hariharan, Aleksander Holynski, and Noah Snavely. 2023. Tracking everything everywhere all at once. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 19795–19806.

Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. 2017. Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2043–2050. IEEE.

Yufei Wang, Zhanyi Sun, Jesse Zhang, Zhou Xian, Erdem Biyik, David Held, and Zackory Erickson. 2024. Rl-vlm-f: Reinforcement learning from vision language foundation model feedback. In *International Conference on Machine Learning*.

Chuan Wen, Xingyu Lin, John So, Kai Chen, Qi Dou, Yang Gao, and Pieter Abbeel. 2023. Any-point trajectory modeling for policy learning. *arXiv preprint arXiv:2401.00025*.

Nicolai Wojke, Alex Bewley, and Dietrich Paulus. 2017. Simple online and realtime tracking with a deep association metric. In *2017 IEEE international conference on image processing (ICIP)*, pages 3645–3649. IEEE.

Ancong Wu, Wei-Shi Zheng, Hong-Xing Yu, Shaogang Gong, and Jianhuang Lai. 2017. Rgb-infrared cross-modality person re-identification. In *Proceedings of the IEEE international conference on computer vision*, pages 5380–5389.

Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 2015. 3D shapenets: A deep representation for volumetric shapes. In *IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920.

Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. 2018. PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes. *Robotics: Science and Systems (RSS)*.

Yu Xiang, Christopher Xie, Arsalan Mousavian, and Dieter Fox. 2021. Learning rgb-d feature embeddings for unseen object instance segmentation. In *Conference on Robot Learning*, pages 461–470. PMLR.

Christopher Xie, Yu Xiang, Arsalan Mousavian, and Dieter Fox. 2021. Unseen object instance segmentation for robotic environments. *IEEE Transactions on Robotics*, 37(5):1343–1359.

Mengda Xu, Zhenjia Xu, Yinghao Xu, Cheng Chi, Gordon Wetzstein, Manuela Veloso, and Shuran Song. 2024. Flow as the cross-domain manipulation interface. In *8th Annual Conference on Robot Learning*.

Ning Xu, Linjie Yang, Yuchen Fan, Dingcheng Yue, Yuchen Liang, Jianchao Yang, and Thomas Huang. 2018. Youtube-vos: A large-scale video object segmentation benchmark. *arXiv preprint arXiv:1809.03327*.

Jiaolong Yang, Hongdong Li, Dylan Campbell, and Yunde Jia. 2016. Go-icp: a globally optimal solution to 3d icp point-set registration. *arXiv preprint arXiv:1605.03344*.

Jinyu Yang, Mingqi Gao, Zhe Li, Shang Gao, Fangjing Wang, and Feng Zheng. 2023. Track anything: Segment anything meets videos.

Linjie Yang, Yuchen Fan, and Ning Xu. 2019. Video instance segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5188–5197.

Weihao Yu, Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Zicheng Liu, Xinchao Wang, and Lijuan Wang. 2023a. Mm-vet: Evaluating large multimodal models for integrated capabilities. *arXiv preprint arXiv:2308.02490*.

Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montserrat Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, et al. 2023b. Language to rewards for robotic skill synthesis. In *Conference on Robot Learning*, pages 374–404. PMLR.

Chengbo Yuan, Chuan Wen, Tong Zhang, and Yang Gao. 2024a. General flow as foundation affordance for scalable robot learning. *arXiv preprint arXiv:2401.11439*.

Wentao Yuan, Jiafei Duan, Valts Blukis, Wilbert Pumacay, Ranjay Krishna, Adithyavairavan Murali, Arsalan Mousavian, and Dieter Fox. 2024b. Robopoint: A vision-language model for spatial affordance prediction in robotics. In *8th Annual Conference on Robot Learning*.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. 2024. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9556–9567.

Andy Zeng, Kuan-Ting Yu, Shuran Song, Daniel Suo, Ed Walker, Alberto Rodriguez, and Jianxiong Xiao. 2017. Multi-view self-supervised deep learning for 6D pose estimation in the amazon picking challenge. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1386–1383.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. 2023. Learning fine-grained bimanual manipulation with low-cost hardware. In *Robotics: Science and Systems XIX, Daegu, Republic of Korea, July 10-14, 2023*.

Yizhou Zhao, Zhenyang Li, Xun Guo, and Yan Lu. 2022. Alignment-guided temporal attention for video action recognition. *Advances in Neural Information Processing Systems*, 35:13627–13639.

Liang Zheng, Liyue Shen, Lu Tian, Shengjin Wang, Jingdong Wang, and Qi Tian. 2015. Scalable person re-identification: A benchmark. In *Proceedings of the IEEE international conference on computer vision*, pages 1116–1124.

Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. 2016. Fast global registration. In *European Conference on Computer Vision (ECCV)*, pages 766–782. Springer.

Wanrong Zhu, Jack Hessel, Anas Awadalla, Samir Yitzhak Gadre, Jesse Dodge, Alex Fang, Youngjae Yu, Ludwig Schmidt, William Yang Wang, and Yejin Choi. 2023. Multimodal C4: An open, billion-scale corpus of images interleaved with text. *arXiv preprint arXiv:2304.06939*.

Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. 2020. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*.

Appendix A

STOW

A.1 Dataset Detail

A.1.1 Synthetic Data

We built a synthetic dataset using high-quality household models from the GoogleScanned datasetDowns et al. [2022] with two typical settings:

1. Shelf
2. Tabletop.

Shelf environment. In shelf environments or other bin-based object arrangements, the objects are akin to books and are constrained to a shortest-dimension-faces-outward orientation. This scheme ensures that each object is guaranteed to have at least one visible face, but it also leads to significant occlusion among objects. The camera is positioned at the front of the bin to capture images of the scene, subject to random perturbations in the location that inject noise into the data.

Given that each bin contains a maximum of 3 to 5 objects, segmentation and tracking tasks become trivial if the scene contains fewer than 3 objects. To address this issue, image frames are generated only when the bin is nearly full. We leverage approximately 900 objects sourced from the Google Scanned dataset, resulting in a training set of approximately 9000 image pairs. We use the remaining 100 objects to generate approximately 1000 image pairs for the test set. Each image pair may exhibit the introduction of a new object in addition to existing objects undergoing a flipping operation or relocation with a certain probability.

Tabletop environments. Generating datasets of objects placed on a table requires different settings given the absence of walls and typically larger surface area than in a bin-based object arrangements. As a result, we adopt an alternative strategy for dataset generation. Specifically, each sequence consists of 15 images, with the first 10 images incrementally introducing new objects while shuffling existing objects between frames. No new objects are added in the final 5 frames, though the shuffling of existing objects persists. Due to the random placement of objects on the

Figure A.1: Some objects used during the evaluation. Objects vary greatly in shape and physical properties, with some being partially transparent or wrapped in a bag.

table, instances of full occlusion may occur in certain frames and subsequently reappear in subsequent frames.

To construct our training and testing datasets, we utilize 900 objects sourced from the Google Scanned dataset, producing 2000 sequences for the training set, with the remaining 100 objects used to generate 500 sequences for the test set.

A.1.2 Real Data

As we did for the synthetic evaluation, we split the evaluation into shelf and tabletop environments, the most common real-world scenarios encountered. To evaluate our method on challenging real-world scenarios, we need a large variety of objects; Figure A.1 depicts some of the objects used during the tablefop evaluation. For shelf environments, we use an Azure Kinect RGB-D sensor, and for tabletop ones we use an Intel Realsense D455 camera. Camera distance ranges from 1 to 1.5 meters. Each time an object is placed on the table or in a new bin, a new image is captured. Objects can be rearranged to maximize space utilization as they are placed in the scene. After all the objects are placed in the scene, we also displace the objects for a more refined evaluation. Camera images are manually labeled using the interactive segmentation of the object tracking framework XMem Cheng and Schwing [2022]. We collected and annotated more than 280 images with more than 150 different objects for the tabletop scenario and 220 images for the shelf scenario.

A.2 Training and Inference Details

A.2.1 Training Details

We set the maximum number of iterations to 16k using an initial learning rate of 1e-5, which was then dropped by 0.1 after 14k iterations. The number of classes is set to 1 since we are aiming to handle unseen objects. For the shelf dataset, we trained our network with a batch size of 32 and leveraged 2 frames from each sequence; for the table dataset, we set the batch size to 8 and

randomly selected 4 frames from each sequence. To enhance the diversity of our dataset, we applied random color jittering and rotation to the input before feeding it to the network. The training process was executed on a single NVIDIA A-40 GPU and took approximately 13 hours.

During the training phase, we excluded the initial predicted object embedding, which was directly generated from the query feature. Additionally, when handling negative queries, we adopted a more selective approach by considering only queries whose IoU with any ground truth was lower than 0.6 rather than regarding all unmatched queries as negatives. This was motivated by the lack of clarity regarding which patches truly represent objects in unseen object settings (in contrast to close-set settings).

A.2.2 Associator

We show below an example of code demonstrating how to associate object tokens from a new frame with the trajectory bank built in previous frames. In implementation, we set $\sigma_{score} = 0.6$ and $\sigma_{\text{match}} = 0.2$ (similarity ranged in $[-1, 1]$).

```
def associate_one_frame(traj_bank, object_tokens_cur_frame, delta_score, delta_trac
    object_tokens = [x for x in queries_this_frame if x['score']>delta_score]
    num_trackers = len(traj_bank)
    Nq = len(object_tokens)
    similarity = torch.ones(num_trackers+Nq, num_pred)*delta_track

    # Extract object embedding from current frame's object tokens
    obj_embed = torch.stack([x['obj_embed'] for x in object_tokens])

    # Compute similarity between object embedding of trajectory and current frame's
    for traj_idx, traj in enumerate(traj_bank):
        traj_obj_embed = torch.stack([x['obj_embed'] for x in traj])
        sim = traj_obj_embed @ obj_embed
        similarity[traj_idx] = sim.max(dim=0)[0]

    # Perform Hungarian matching to find bipartite matching which have hightest sir
    traj_indices, obj_token_indices = hungarian_matching(-similarity)

    # Update tracker
    for traj_idx, token_idx in zip(traj_indices, obj_token_indices):
        if traj_idx > num_trackers:
            # if it is not matched with any existing trajectory
            traj_bank.append([object_tokens[token_idx]])
        else:
            traj_bank[traj_idx].append(object_tokens[token_idx])
    return traj_bank
```

A.2.3 Loss

We keep the loss function that Mask2Former used for classification and mask prediction, which means binary cross entropy and dice loss for mask prediction and softmax cross entropy loss for classification.

For the object embedding head, we also use two losses: contrastive loss and softmax loss (or n-pair loss and InfoNCE loss).

Contrastive Loss. We use contrastive loss modified from DCNFlorence et al. [2018] with hard-negative scaling from Florence [2020].

$$\mathcal{L}_{\text{matches}}(Q) = \frac{1}{N_{\text{matches}}} \sum_{N_{\text{matches}}} D(q_{t1}^{o_i}, q_{t2}^{o_i})^2 \tag{A.1}$$

$$\mathcal{L}_{\text{non-matches}}(Q) = \frac{1}{N_{\text{hard-neg}}} \sum_{N_{\text{non-matches}}} (0, M - D(q_{t1}^{o_i}, q_{t2}^{o_j})_{i \neq j}) \tag{A.2}$$

$$\mathcal{L}(Q) = \mathcal{L}_{\text{matches}}(Q) + \mathcal{L}_{\text{non-matches}}(Q), \tag{A.3}$$

where

$$N_{\text{hard-negatives}} = \sum_{N_{\text{non-matches}}} \mathbb{1}(M - D(q_{t1}^{o_i}, q_{t2}^{o_i}) > 0). \tag{A.4}$$

Here, Q denotes all object tokens from images, and $q_t^{o_i}$ denotes the object tokens assigned to object o_i in frame t. M is the margin parameter used to ensure that non-matched pairs have a distance of at least M apart. The distance function D is the cosine distance function, as in UCN Xiang et al. [2021], which is defined as:

$$D(q^i, q^j) = \frac{1}{2}(1 - r^i \cdot r^j). \tag{A.5}$$

Here, $r^i = \frac{f(q^i)}{|f(q^i)|}$ is the object embedding of object token i, which is computed by first forwarding the query to a linear layer f and then normalizing it to a unit vector. To expedite the training process, we selectively incorporate a subset of negative queries to contribute to the contrastive loss, thereby enhancing its efficiency.

An illustration of the contrastive loss is shown in Figure A.2. Assuming that three frames are sampled from a sequence during training, the contrastive loss will be computed between all frames. In the figure, matched pairs are denoted by dark gray and apply loss according to Equation A.1, while non-matched pairs are denoted by light gray and apply loss according to Equation A.2.

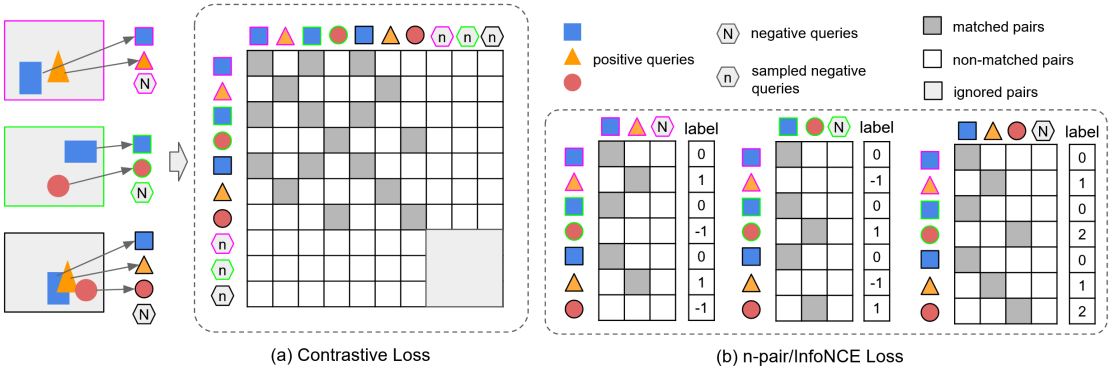


Figure A.2: Tracking loss. In this example, three frames are sampled from a sequence, denoted with different border colors. Object tokens that match the objects in the images are represented by a blue square, an orange triangle, and a red circle. The hexagon denotes background object tokens that do not match to any objects. (a) the contrastive loss is computed between all frames, where matched pairs (dark gray) apply loss using Equation A.1, non-matched pairs (white) apply loss using Equation A.2, and ignored pairs (light gray) do not contribute to the loss. (b) the n-pair/InfoNCE loss is computed over all positive queries and queries from each frame. Equivalent to using a softmax cross-entropy while setting the label of the index of queries assigned to the same object.

Softmax Loss We also modified the N-pair/InfoNCE loss used in CLIPRadford et al. [2021].

$$\mathcal{L}_{\text{softmax}}(t) = -\sum_{k \in Q^+} \sum_{i \in O(t)} \frac{\exp(r_t^k \cdot r_t^i \cdot e^\tau)}{\sum_{j \in Q_t} \exp(r_t^k \cdot r_t^i \cdot e^\tau)} \tag{A.6}$$

$$\mathcal{L}_{\text{softmax}} = \frac{1}{T} \sum_{t \in 1, \cdots, T} \mathcal{L}_{\text{softmax}}(t), \tag{A.7}$$

where Q^+ denotes all positive queries, $O(t)$ denotes all objects in frame t, and Q_t denotes all queries in frame t. This is also shown in Fig. A.2-b, where the label of each row corresponds to the index of the query assigned to the same object. If there are n identical objects in the same frame, the softmax loss should be extended by copying all queries in this frame n times, each time keeping only one query for that object. This allows queries containing the same object to be converted into multiple query sets, each consisting of only the target object.

Thus, the final tracking loss can be represented as

$$\mathcal{L}_{\text{track}} = \lambda_{\text{contra}} \mathcal{L}_{\text{contra}} + \lambda_{\text{softmax}} \mathcal{L}_{\text{softmax}}. \tag{A.8}$$

method	syn., video AP		syn., image AP		real, video AP		real, image AP	
	AP@all	AP@0.5	AP@all	AP@0.5	AP@all	AP@0.5	AP@all	AP@0.5
UCN Xiang et al. [2021]	-	-	67.3	82.9	-	-	52.4	86.6
MinVIS Huang et al. [2022]	0.3	2.6	82.4	91.8	0.7	0.0	54.5	72.7
Mask2Former Video Cheng et al. [2021]	71.6	83.7	72.8	82.6	27.7	56.7	38.9	57.3
VITA Heo et al. [2022]	69.4	81.9	70.6	80.2	26.6	55.0	41.4	63.0
Ours	**74.1**	**89.3**	**87.6**	**95.3**	**49.7**	**75.4**	**80.1**	**97.6**

Table A.1: Evaluation of SOTA VIS methods on the unseen object instance segmentation task. "syn" indicates evaluation on a synthetic tabletop dataset; "real" denotes evaluation on a real-world tabletop dataset. The evaluation metrics include "video" and "image" for assessing performance in different contexts (as described in subsection A.2.1). We see that MinVIS exhibits superior performance in detection, while Mask2Former Video and VITA excel in matching. Remarkably, our proposed method harnesses the strengths of both approaches, surpassing all evaluated methods in overall performance.

A.3 Detailed Analysis

A.3.1 Sim-to-real Gap

Results are shown in Table A.1. The experiment is conducted in the tabletop environment with the same setting as in main manuscript. We see the following from

(1) The Image AP results from both the synthetic validation set and real test set demonstrate that MinVIS outperforms Mask2Former Video and VITA. This indicates that the approach of Mask2Former Video and VITA, which utilizes a single object token to predict object masks across an entire sequence, is less effective than using distinct object tokens for each frame. Consequently, it is difficult for object tokens to efficiently manage discrete frames with considerable movement and appearance variations.

(2) A significant decline is apparent in the Image AP and Video AP results on the real test set for MinVIS. This suggests a suboptimal tracking performance when applying object tokens directly.

(3) Notably, our method experiences a less dramatic drop in performance, as evidenced by the Video AP results on both the synthetic validation set and the real test set. This suggests that our method is more adept at managing the simulation-to-reality gap.

A.4 Failure Cases

The efficacy of our method is sometimes compromised in scenarios characterized by crowded scenes or significant alterations in object appearance, as illustrated in Figure A.5. We categorize these shortcomings into two principal groups: segmentation failures and tracking failures.

Segmentation failures arise from issues such as:

- *Under-segmentation*: This occurs when objects have similar colors or lack clear borders, leading to a blending of distinct entities.

- *Over-segmentation*: In this case, a single object is erroneously identified as multiple entities due to recognition failures.
- *Detection failure*: Here, an object is entirely missed, leading to its absence in the segmented output.

Tracking failures, on the other hand, include:

- *Mismatch*: This involves incorrect associations between objects across frames or confusion arising from similar-looking distractors.
- *Mistrack*: In these instances, the algorithm fails to consistently identify the same object, resulting in tracking inconsistencies.

In both categories of failure, the complexity of scene compositions and variations in object appearances are pivotal factors that undermine the performance of our tracking method. We are devoted to exploring advanced strategies to mitigate these limitations, aiming for enhanced robustness in diverse and dynamic environments.

Figure A.3: Results from different methods on the tabletop dataset. Methods ordered from top to bottom: MinVIS, Mask2Former-Video, VITA, and Ours (STOW).

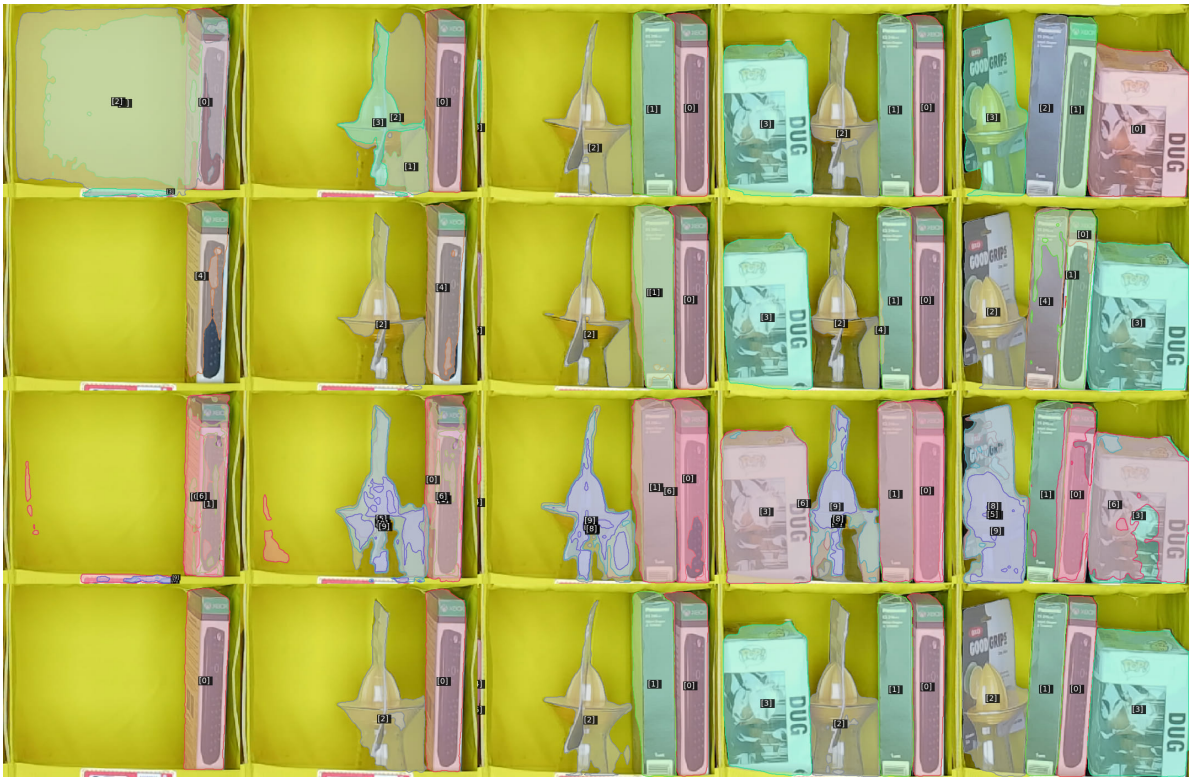


Figure A.4: Results from different methods on the bin dataset. Methods ordered from top to bottom: MinVIS, Mask2Former-Video, VITA, and Ours (STOW).

Figure A.5: Failure Cases Illustrated. In the Tabletop settings (top 2 rows), we observe under-segmentation (top left), over-segmentation (top right), mismatch (bottom left), and mistrack (bottom right). In the Shelf settings (bottom two rows), the failures include a combination of under-segmentation and failure to detect new objects (top left), failure to track and mismatch (top right), failure to detect (bottom left), and failure to segment (bottom right).

Figure B.1: Examples of various robot tasks and environments that HAMSTER can handle. See more details in our teaser video at https://hamster-robot.github.io/.

Appendix B

Hamster

B.1 VLM Finetuning Dataset Details

Pixel Point Pred Data. Our point prediction dataset comes from Robopoint [Yuan et al., 2024b]. 770k samples in our point prediction dataset contain labels given as a set of unordered points such as $p^o = [(0.25, 0.11), (0.22, 0.19), (0.53, 0.23)]$, or bounding boxes in $[(cx, cy, w, h)]$ style. Other than that, following Robopoint [Yuan et al., 2024b], we use the VQA dataset Liu et al. [2024c] with 660k samples which answer VQA queries in natural language such as "What is the person feeding the cat?" We keep these data as is because these VQA queries are likely to benefit a VLM's semantic reasoning and visual generalization capabilities; we fine-tune HAMSTER's VLM on the entire Robopoint dataset as given.

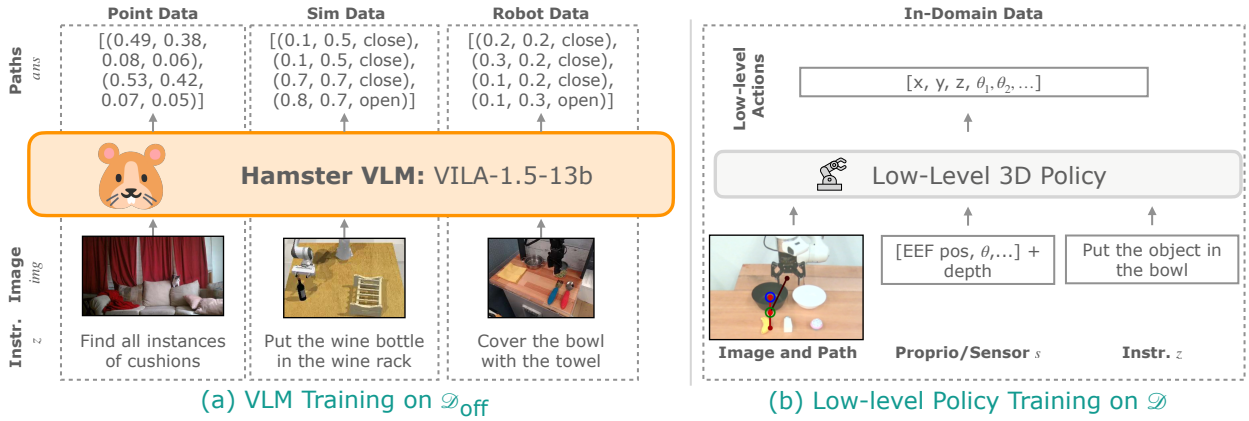


Figure B.2: (a): Examples of training data in $\mathcal{D}_{\text{off}}$ used to train HAMSTER's VLM. (b): The data used to train HAMSTER's low-level policies.

Simulation Data. We selected 81 RLBench tasks out of 103 to generate data by removing tasks with poor visibility on the `front_cam` view in RLBench. We use the first image in each episode combined with each language instruction. The final dataset contains around 320k trajectories.

Real Robot Data. For the Bridge [Walke et al., 2023] dataset, which only provides RGB images, we extract trajectories by iteratively estimating the extrinsic matrix for each episode. In each scene, we randomly sample a few frames and manually label the center of the gripper fingers. Using the corresponding end-effector poses, we compute the 3D-2D projection matrix with a PnP (Perspective-n-Point) approach. We then apply this projection matrix to the episodes and manually check for any misalignments between the projected gripper and the actual gripper. Episodes exhibiting significant deviations are filtered out, and a new round is started to estimate their extrinsic matrix.

For DROID [Khazatsky et al., 2024], a large portion of the dataset contains noisy camera extrinsics information that do not result in good depth alignment. Therefore, we filter out trajectories with poor-quality extrinsics as measured by the alignment between the projected depth images and the RGB images. This results in ∼45k trajectories (∼22k unique trajectories as trajectories each have 2 different camera viewpoints) which we use for constructing the VLM dataset $\mathcal{D}_{\text{off}}$ as described in Section 4.4.1.

B.2 Implementation and Architecture Details

B.2.1 VLM Implementation Details

VLM Prompt. We list the prompt for both fine-tuning on sim and real robot data and evaluation in Figure B.3. We condition the model on an image and the prompt, except when training on Pixel Point Prediction data (i.e., from Robopoint [Yuan et al., 2024b]) where we used the given prompts

HAMSTER Prompt

In the image, please execute the command described in ⟨quest⟩{quest}⟨/quest⟩.
Provide a sequence of points denoting the trajectory of a robot gripper to achieve the goal.
Format your answer as a list of tuples enclosed by ⟨ans⟩ and ⟨/ans⟩ tags. For example:
⟨ans⟩`[(0.25, 0.32), (0.32, 0.17), (0.13, 0.24),` ⟨action⟩`Open Gripper`⟨/action⟩`, (0.74, 0.21),` ⟨action⟩`Close Gripper`⟨/action⟩`, ...]`⟨/ans⟩
The tuple denotes the x and y location of the end effector of the gripper in the image. The action tags indicate the gripper action.
The coordinates should be floats ranging between 0 and 1, indicating the relative locations of the points in the image.

Figure B.3: The full text prompt we use to train HAMSTER with on simulation and real robot data (Section 4.4.1). We also use this prompt for inference.

from the dataset. Note that we ask the model to output gripper changes as separate language tokens, i.e., `Open Gripper/Close Gripper`, as opposed to as a numerical value as shown in simplified depictions like Figure 4.2.

VLM Trajectory Processing. As mentioned in Section 4.4.1, one problem with directly training on the path labels p^o is that many paths may be extremely long. Therefore, we simplify the paths p^o with the Ramer-Douglas-Peucker algorithm [Ramer, 1972; Douglas and Peucker, 1973] that reduces curves composed of line segments to similar curves composed of fewer points. We run this algorithm on paths produced by simulation and real robot data to generate the labels p^o for $\mathcal{D}_{\text{off}}$. We use tolerance $\epsilon = 0.05$, resulting in paths that are around 2-5 points for each short horizon task.

VLM Training Details. We train our VLM, VILA1.5-13B Lin et al. [2024], on a node equipped with eight NVIDIA A100 GPUs, each utilizing approximately 65 GB of memory. The training process takes about 30 hours to complete. We use an effective batch size of 256 and a learning rate of 1×10^{-5}. During fine-tuning, the entire model—including the vision encoder—is updated.

B.2.2 Low-level Policy Training Details

We train RVT2 [Goyal et al., 2024] and 3D-DA [Ke et al., 2024] as our lower-level policies. We keep overall architecture and training hyperparameters the same as paper settings. Specific details about how the inputs were modified other than the 2D path projection follow.

For low-level policy training, we train the policies on ground truth paths constructed by projecting trajectory end-effector points to the camera image. In order to also ensure the policies are robust to possible error introduced by HAMSTER VLM predictions during evaluation, we add a small amount of random noise ($N(0, 0.01)$) to the 2D path (x, y) image points during training to obtain slightly noisy path drawings. No noise was added to the gripper opening/closing indicator values.

RT-Trajectory GPT-4o Prompt

In the image, please execute the command described in '{quest}'.
Provide a sequence of keypoints denoting a trajectory of a robot gripper to achieve the goal. Keep in mind these are keypoints, so you do not need to provide too many points.
Format your answer as a list of tuples enclosed by `<ans>` and `</ans>` tags. For example:
`<ans>[(0.25, 0.32), (0.32, 0.17), (0.13, 0.24), <action>Open Gripper</action>, (0.74, 0.21), <action>Close Gripper</action>, ...]</ans>`
The tuple denotes point x and y location of the end effector of the gripper in the image. The action tags indicate the gripper action.
The coordinates should be floats ranging between 0 and 1, indicating the relative locations of the points in the image.
The current position of the robot gripper is: {current_position}. Do not include this point in your answer.

Figure B.4: The full text prompt we use to prompt RT-Trajectory with GPT4-o.

RVT2 [Goyal et al., 2024]. We remove the language instruction for RVT-2 when conditioning on HAMSTER 2D paths.

3D-DA [Ke et al., 2024]. In simulated experiments in Colosseum, no changes were needed. In fact, we saw a performance drop for HAMSTER+3D-DA when removing language for Colosseum tasks and a small drop in performance when using simplified language instructions. This is likely due to 3D-DA's visual attention mechanism which cross attends CLIP language token embeddings with CLIP visual features, therefore detailed language instructions are beneficial.

In real-world experiments, we simplify the language instruction in the same way as for RVT2 when conditioning on HAMSTER 2D paths to encourage following the trajectory more closely with limited data. In addition, we reduced the embedding dimension of the transformer to 60 from 120, removed proprioception information from past timesteps, and reduced the number of transformer heads to 6 from 12 in order to prevent overfitting.

B.3 Real World Experiment Details

B.3.1 Training Tasks and Data Collection

For our real-world experiments, we collected all data using a Franka Panda arm through human teleoperation, following the setup described in Khazatsky et al. [2024]. Below, we describe the training tasks:

Pick and place. We collected 220 episodes using 10 toy objects. In most of the training data, 2 bowls were placed closer to the robot base, while 3 objects were positioned nearer to the camera. The language goal for training consistently followed the format: `pick up the {object} and put it in the {container}`.

RT-Trajectory Code as Policies Prompt

Task Instruction: {task_instruction}
Robot Constraints:

- The robot arm takes as input 2D poses with gripper open/closing status of the form $(x, y, \text{gripper\_open} == 1)$
- The gripper can open and close with only binary values
- The workspace is a 1×1 square centered at $(0.5, 0.5)$
- The x-axis points rightward and y-axis points downward.

Please write Python code that generates a list of 2D poses and gripper statuses for the robot to follow. Include Python comments explaining each step. Assume you can use `numpy` or standard Python libraries, just make sure to import them.
Enclose the start and end of the code block with `<code>` and `</code>` so that it can be parsed. Make sure that it is a self-contained script such that when executing the code string, there is a variable named `robot_poses` which is a list of poses of the form: `[(x, y, gripper), (x, y, gripper), ...]`.
Scene Description:

```
<code>
{scene_description}
</code>
```

Figure B.5: The full text prompt we use for RT-Trajectory with Code-as-Policies on top of GPT4-o. The scene description at the bottom comes from an open-vocabulary object detector describing each detected object and its bounding box in the image based on the task instruction.

Knock down objects. We collected 50 episodes with various objects of different sizes. Typically, 3 objects were arranged in a row, and one was knocked down. The language goal for training followed the format: `push down the {object}`.

Press button. We collected 50 episodes with 4 colored buttons. In each episode, the gripper was teleoperated to press one of the buttons. The language goal followed the format: `press the {color} button`.

When training RVT2, which requires keyframes as labels, in addition to labeling frames where the gripper performs the `open gripper` and `close gripper` actions, we also included frames that capture the intermediate motion as the gripper moves toward these keyframes.

B.3.2 Baseline Training Details

OpenVLA [Kim et al., 2024]. Following Kim et al. [2024], we only utilize parameter efficient fine-tuning (LoRA) for all of our experiments, since they showed that it matches full fine-tuning performance while being much more efficient. We follow the recommended default rank of r=32. We opt for the resolution of 360 x 360 to match all of the baseline model's resolutions. We also follow the recommended practice of training the model until it surpasses 95% token accuracy. However, for some fine-tuning datasets, token accuracy converged near 90%. We selected the model checkpoints when we observed that the token accuracy converged, which usually required 3,000 to 10,000 steps using a global batch size of either 16 or 32. Training was conducted with 1 or 2 A6000 gpus (which determined the global batch size of 16 or 32). Emprically, we observed that checkpoints that have converged showed very similar performance in the real world. For example, when we evaluate checkpoint that was trained for 3,000 steps and showed convergence, evaluating on a checkpoint trained for 5,000 steps of the same run resulted in a very similar performance.

RT-Trajectory [Gu et al., 2023]. We implement two versions of RT-Trajectory for the comparison in Table B.2. The first (0-shot GPT-4o) directly uses GPT-4o to generate 2D paths with a prompt very similar to the one we use for HAMSTER, displayed in Figure B.4.

The second version implements RT-Trajectory on top of a Code-as-Policies [Liang et al., 2023], as described in RT-Trajectory. We use OWLv2 [Minderer et al., 2023] to perform open-vocabulary object detection on the image to generate a list of objects as the scene description and then prompt RT-Trajectory with the prompt shown in Figure B.5. We also use GPT-4o as the backbone for this method.

B.3.3 Evaluation Tasks

We evaluate our method on the tasks of `pick and place`, `knock down object`, and `press button` across various generalization challenges, as illustrated in Figure 4.4. Detailed results are available in Table B.1. Following [Kim et al., 2024], we assign points for each successful sub-action. For VLM, human experts are employed to assess the correctness of the predicted trajectories.

Category	Task	OpenVLA	RVT2	RVT2+Sketch	3DDA	3DDA+Sketch
Basic	pick up the corn and put it in the black bowl	1	1	1	0	0.25
Basic	pick up the grape and put it in the white bowl	1	0.75	1	0	1
Basic	pick up the milk and put it in the white bowl	0	1	1	0	0.25
Basic	pick up the salt bottle and put it in the white bowl	0.75	0.5	1	0	0
Basic	pick up the shrimp and put it in the red bowl	0.75	0.5	1	0	1
Basic	pick up the cupcake and put it in the red bowl	0	0.5	0.5	0.25	1
Basic	press down the red button	0.5	0	1	0	1
Basic	press down the green button	0	1	0	0	0.25
Basic	press down the yellow button	0	0	1	0	1
Basic	press down the blue button	0.5	0	1	0	0.5
Basic	push down the green bottle	0.5	0	0.5	0	1
Basic	push down the pocky	0	1	1	0	0.5
Basic	push down the red bag	0.5	0.5	0	0	0.5
Basic	push down the bird toy	0	0	0	0	0.5
Basic	push down the yellow box	1	0	1	0	0.5
Object and Goal	pick up the salt bottle and put it in the white bowl	1	1	1	0.5	1
Object and Goal	pick up the banana and put it in the black bowl	0.25	0.25	1	0.5	1
Object and Goal	pick up the grape and put it in the black bowl	1	0.25	0.5	1	1
Object and Goal	pick up the carrot and put it in the red bowl	0.75	0	1	0.5	1
Object and Goal	pick up the milk and put it in the white bowl	0.25	0	1	0	0.25
Object and Goal	pick up the shrimp and put it in the white bowl	0.25	0.75	0.5	0.25	1
Object and Goal	pick up the cupcake and put it in the black bowl	0.25	0	1	0.5	0.75
Object and Goal	pick up the icecream and put it in the black bowl	0.25	0	0.5	0.5	1
Object and Goal	pick up the corn and put it in the red bowl	1	0	1	1	1
Object and Goal	pick up the green pepper and put it in the red bowl	0.75	0	0.5	0	0.25
Object and Goal	pick up the orange and put it in the white bowl	0.25	0	0	0	0
Visual(Table Texture)	pick up the salt bottle and put it in the white bowl	1	1	1	0	1
Visual(Table Texture)	pick up the banana and put it in the black bowl	0.25	0.25	0.75	0.5	0.75
Visual(lighting)	pick up the grape and put it in the black bowl	0.25	0	0.5	0.25	0
Visual(lighting)	pick up the carrot and put it in the red bowl	0.75	0	1	0	0.75
VIsual(clutter)	pick up the milk and put it in the white bowl	0.75	0.25	1	0.25	1
VIsual(clutter)	pick up the shrimp and put it in the red bowl	0.75	0.5	0	0	0.5
Visual(mix)	pick up the green pepper and put it in the red bowl	0.25	0	1	0	0.25
Visual(mix)	pick up the salt bottle and put it in the white bowl	0.25	0	0.25	0.25	1
Visual(appearance change)	pick up the green pepper and put it in the black bowl	1	0	0.5	0	1
Visual(appearance change)	pick up the salt bottle and put it in the black bowl	1	1	1	0	1
Visual(Table Texture)	press down the red button	1	1	0	0	0.5
Visual(lighting)	press down the green button	1	0	0.5	0	0.5
VIsual(clutter)	press down the yellow button	0	0	0.5	0	0.5
Visual(mix)	press down the blue button	0	0	0	0	0.5
Visual(Table Texture)	push down the pocky	0	1	0	0	0
VIsual(clutter)	push down the green bottle	1	0.5	1	0	1
VIsual(clutter)	push down the chocolate box	1	0	0	0	1
Visual(mix)	push down the green bottle	0	0	0.5	0	1
Language	pick up the sweet object and put it in the red bowl	1	1	1	0	1
Language	pick up the spicy object and put it in the red bowl	1	0	1	0	0.75
Language	pick up the salty object and put it in the red bowl	0	0	1	0	1
Language	pick up the object with color of cucumber and put it in the red bowl	0	0	1	0.25	0.75
Language	pick up the object with color of lavender and put it in the black bowl	0	0	1	0	1
Language	pick up the object with the color of sky and and put it in the container with the color of coal	1	0	0	0.25	1
Language	pick up the block with the color of sunflower and put it in the container with the color of enthusiasm	0	0.25	1	0	1
Language	press the button with the color of fire	0.5	0	1	0	0.5
Language	press the button with the color of cucumber	0	0	1	0	0.5
Language	press the button with the color of sky	0	0	0	0.5	1
Language	press the button with the color of banana	0	0	0	0	0.5
Language	push down the object with color of leaf	0	1	1	0	0
Language	push down the box contains cruchy biscuit	0	0	0	0	1
Language	push down the bag with color of fire	0	0	1	0	0.5
Language	push down the object with feather	0.5	0	1	0	1
Spatial	pick up the left object and put it in the left bowl	0	1	1	0.25	1
Spatial	pick up the middle object and put it in the left bowl	0	0	1	0	1
Spatial	pick up the right object and put it in the left bowl	1	0	0.5	0.25	0.5
Spatial	pick up the left object and put it in the right bowl	0.25	0.25	1	0.25	1
Spatial	pick up the middle object and put it in the right bowl	0	0	1	0	1
Spatial	pick up the right object and put it in the right bowl	0.5	0	1	0	1
Spatial	press down the left button	0.5	0	0	0	0.5
Spatial	press down the middle button	0	0	1	1	0.5
Spatial	press down the right button	0	0	1	1	1
Spatial	push down the left object	0.5	0	0	0	0
Spatial	push down the middle object	1	0.5	0	0	1
Spatial	push down the right object	0.5	0	0.5	0.5	1
Novel Object	pick up the "R" and put it in the red bowl	0	0	1	0	1

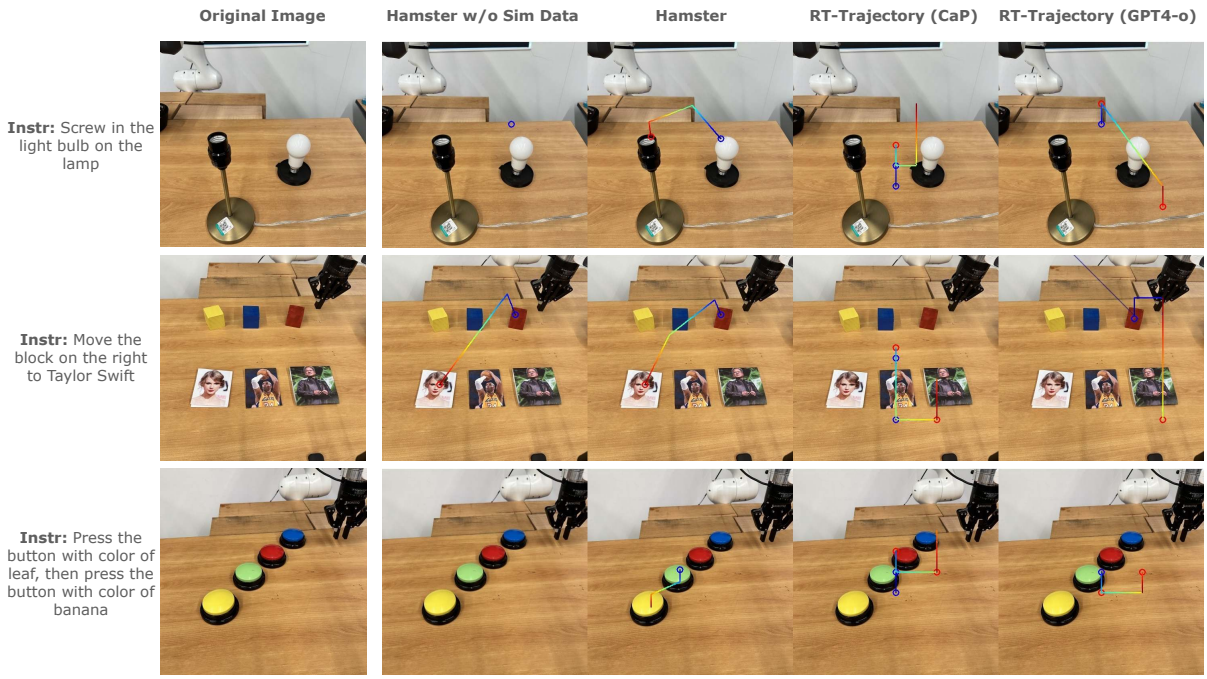


Figure B.6: Human VLM evaluation example images and instructions along with corresponding trajectories from HAMSTER without any finetuning on (RLBench) simulation data, HAMSTER finetuned on all the data in Section 4.4.1, RT-Trajectory [Gu et al., 2023] with Code-as-Policies [Liang et al., 2023] powered by GPT-4o [Achiam et al., 2023], and RT-Trjaectory powered by GPT-4o directly.

B.4 Extended Results

B.4.1 Impact of Design Decisions on VLM performance

To better understand the transfer and generalization performance of the proposed hierarchical VLA model, we analyze the impact of various decisions involved in training the high-level VLM. We conduct a human evaluation of different variants of a trained high-level VLM on a randomly collected dataset of real-world test images, as shown in Figure 4.7. We ask each model to generate 2D path traces corresponding to instructions such as "move the block on the right to Taylor Swift" or "screw the light bulb in the lamp" (the full set is in Appendix B.4.2). We then provide the paths generated by each method to human evaluators who have not previously seen any of the models' predictions. The human evaluators then rank the predictions for each method; we report the average rank across the samples in Table B.2.

We evaluate the following VLM models: (1) zero-shot state-of-the-art closed-source models such as GPT-4o using a similar prompt to ours (shown in Figure B.4), (2) zero-shot state-of-the-art closed-source models such as GPT-4o but using Code-as-Policies [Liang et al., 2023] to generate paths as described in Gu et al. [2023] (prompt in Figure B.5), (3) finetuned open-source models (VILA-1.5-13b) on the data sources described in Section 4.4.1, but excluding the simulation tra-

Method	VLM	Finetuning Data	Rank Exc. Real RLB.	Rank Real RLB.	Rank All
RT-Traj.	0-shot GPT-4o	-	3.40	3.63	3.47
RT-Traj.	CaP GPT-4o	-	3.57	3.36	3.41
HAMSTER	VILA	Our Exc. Sim RLB.	1.78	2.39	2.13
HAMSTER	VILA	Our	**1.59**	**1.28**	**1.40**

Table B.2: Ranking-based human evaluation of different VLMs, averaged across various real-world evaluation tasks. Results indicate that HAMSTER including simulation data is most effective since it captures both spatial and semantic information across diverse tasks from RLBench. This significantly outperforms zero-shot VLM-based trajectory generation, as described in Gu et al. [2023]

jectories from the RLBench dataset, (4) finetuned open-source models (VILA-1.5-13b) on the data sources described in Section 4.4.1, including path sketches from the RLBench dataset. The purpose of these evaluations is to first compare with closely related work that generates 2D trajectories using pretrained closed source VLMs Gu et al. [2023] (Comparison (1) and (2)). The comparison between (3) and (4) (our complete method) is meant to isolate the impact of including the simulation path sketches from the RLBench dataset. In doing so, we analyze the ability of the VLM to predict intermediate paths to transfer across significantly varying domains (from RLBench to the real world).

The results suggest that: (1) zero-shot path generation, even from closed-source VLMs Gu et al. [2023] such as GPT-4o with additional help through Code-as-Policies [Liang et al., 2023], underperforms VLMs finetuned on cross-domain data as in HAMSTER; (2) inclusion of significantly different training data such as low-fidelity simulation during finetuning improves the real-world performance of the VLM. This highlights the transferability displayed by HAMSTER across widely varying domains. These results emphasize that the hierarchical VLA approach described in HAMSTER can effectively utilize diverse sources of cheap prior data for 2D path predictions, despite considerable perceptual differences.

B.4.2 VLM Real World Generalization Study

The full list of task descriptions for this study is below (see Appendix B.4.1 for the main experiment details). Duplicates indicate different images for the same task. We plot some additional comparison examples in Figure B.6. Note that the path drawing convention in images for this experiment differ from what is given to the lower-level policies as described in Section 4.4.2 as this multi-colored line is easier for human evaluators to see.

1. screw in the light bulb on the lamp
2. screw in the light bulb on the lamp
3. screw in the light bulb on the lamp

4. screw out the light bulb and place it on the holder
5. screw out the light bulb and place it on the holder
6. screw in the light bulb
7. screw in the light bulb on the lamp
8. move the blue block on Taylor Swift
9. pick up the left block and put it on Jensen Huang
10. move the block on the right to Taylor Swift
11. place the yellow block on Kobe
12. pick up the blue block and place it on Jensen Huang
13. move the red block to Kobe
14. press the button on the wall
15. press the button to open the left door
16. press the button to open the right door
17. open the middle drawer
18. open the bottom drawer
19. open the top drawer
20. open the middle drawer
21. open the bottom drawer
22. press the button
23. press the button
24. press the orange button
25. press the orange button with black base
26. press the button
27. pick up the SPAM and put it into the drawer
28. pick up the orange juice and put it behind the red box
29. pick up the tomato soup and put it into the drawer

30. pick up the peach and put it into the drawer
31. move the mayo to the drawer
32. move the dessert to the drawer
33. pick up the object on the left and place it on the left
34. pick up the fruit on the left and put it on the plate
35. pick up the milk and put it on the plate
36. press the button with the color of cucumber, then press the button with color of fire
37. press the button with color of banana
38. press the button with color of leaf
39. press the button with color of leaf, then press the one with color of banana
40. press left button
41. pick up the left block on the bottom and stack it on the middle block on top
42. make I on top of C
43. put number 2 over number 5
44. stack block with lion over block with earth
45. pick up the left block on the bottom and stack it on the middle block on top
46. stack the leftest block on the rightest block
47. stack the block 25 over block L
48. put the left block on first stair

B.4.3 Human Ranking

Due to the variety of possible trajectories that accomplish the same task, we use human rankings to compare how likely produced trajectories are to solve the task instead of quantitative metrics such as MSE. To do that, we generate trajectories for 48 image-question pairs with HAMSTER w/o RLBench, HAMSTER, Code-as-Policy [Liang et al., 2023], and GPT4o [Achiam et al., 2023]. See Figure B.7 for an example.

We recruit 5 human evaluators, who are robot learning researchers that have not seen the path outputs of HAMSTER, to grade these 4 VLMs based on the instruction: *"Provide a rank for each method (1 for best and 4 for worst). In your opinion, which robot trajectory is most likely to succeed. Traj goes from blue to red, blue circle means close gripper, red circle means open gripper."* The evaluators are

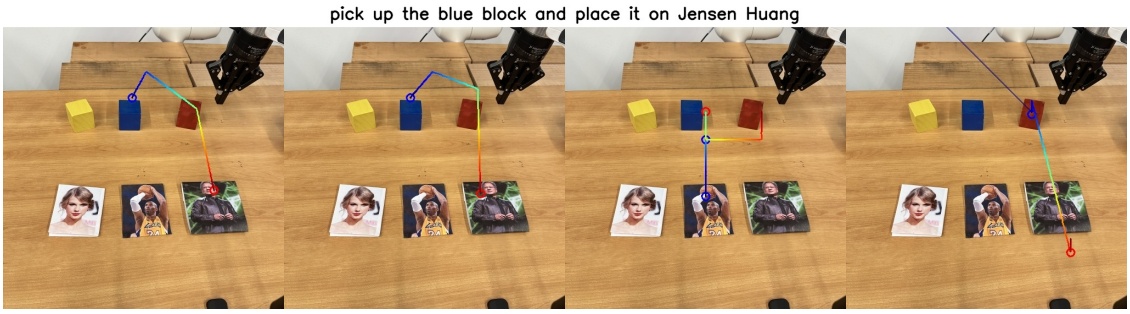


Figure B.7: An example of results for human ranking. The trajectory is from blue to red with blue circle and red circle denotes gripper close point and open point respectively. The grader is asked to provide a rank to these trajectory about which trajectory has highest chance to succeed.

allowed to give multiple trajectories the same score if they believe those trajectories are tied. As they are robot learning researchers, they are familiar with the types of trajectories that are more likely to succeed. Therefore, these rankings act as a meaningful trajectory quality metric.

B.5 Failure Analysis

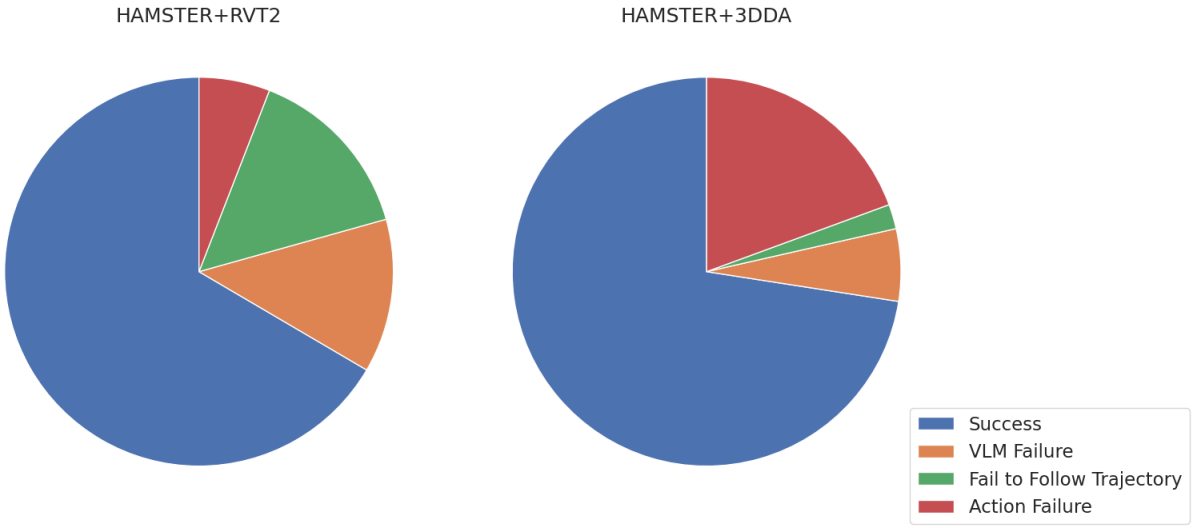


Figure B.8: Performance Distribution of RVT2+Sketch and 3DDA+Sketch

This section outlines the failure modes observed during our experiments and provides a detailed breakdown of the causes. Failures can be attributed to issues in **trajectory prediction**, **trajectory adherence**, and **action execution**.

B.5.1 Different Failure Modes

Trajectory Prediction Failures The Vision-Language Model (VLM) may fail to predict the correct trajectory due to several factors:

- *Failure to understand the language goal:* Although the VLM demonstrates strong capabilities in handling diverse task descriptions, it struggles when the training set lacks similar tasks. This can cause the model to misunderstand the goal and make inaccurate predictions.

- *Incorrect trajectory prediction:* In some cases, the VLM predicts an incorrect trajectory, either by interacting with the wrong objects or misinterpreting the direction of the affordance.

- *Dynamic changes in the environment:* Since trajectories are generated at the beginning of a task, significant environmental changes during execution can lead to failure. The model lacks the ability to dynamically adjust the trajectory or reidentify the object initially referenced.

Trajectory Adherence Failures Failures in adhering to the predicted trajectory arise primarily due to:

- *3D ambiguity:* The use of 2D trajectory predictions introduces ambiguities, such as determining whether a point is positioned above or behind an object, leading to execution errors.

- *Incorrect object interaction:* The low-level action model is not explicitly constrained to strictly follow the predicted trajectory. As a result, it may deviate, interacting with the wrong object and causing task failures.

Action Execution Failures Even when the trajectory is correctly predicted and adhered to, action execution may still fail due to:

- *Execution-specific issues:* Despite training on a diverse set of actions, the model may fail during execution. For example, in grasping tasks, an incorrect grasp angle can cause the object to slip, resulting in a failed grasp.

B.5.2 Failure Analysis

Our analysis in Figure B.8 reveals distinct failure tendencies across methods.

For RVT, 72% of failures stemmed from the low-level model failing to follow the trajectory, while 28% were due to execution failures. In contrast, for 3DDA, only 10% of failures were related to trajectory adherence, with 90% attributed to execution failures.

We hypothesize that this discrepancy arises because RVT incorporates a re-projection step, complicating trajectory adherence. In contrast, 3DDA leverages a vision tower that processes the original 2D image, simplifying trajectory interpretation.

B.6 Simulation Experiment Details

Our simulation experiments are performed on Colosseum [Pumacay et al., 2024], a simulator built upon RLBench [James et al., 2020] containing a large number of visual and task variations to test the generalization performance of robot manipulation policies (see Figure B.9 for a visualization of a subset of the variations). We use the `front_camera` and remove all tasks in which the camera does not provide a clear view of the objects in the task, resulting in 14 out of 20 colosseum tasks (we remove `basketball_in_hoop`, `empty_drawer`, `get_ice_from_fridge`, `move_hanger`, `open_drawer`, `turn_oven_on`).

Colosseum contains 100 training episodes for each task, without any visual variations, and evaluates on 25 evaluation episodes for each variation. We follow the same procedure other than using just the `front_camera` instead of multiple cameras. We report results in Table 4.3 after removing variations with no visual variations (e.g., object friction).

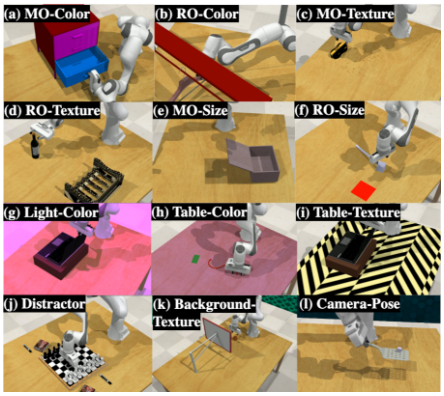


Figure B.9: Colosseum benchmark variations. Figure from Pumacay et al. [2024], taken with permission.

Task	RVT2	3DDA	OpenVLA	HAMSTER+RVT2	HAMSTER+3DDA
pick and place	0.28	0.19	0.46	0.79	0.78
press button	0.13	0.16	0.25	0.50	0.63
knock down	0.17	0.03	0.41	0.47	0.66

Table B.3: Real world average success rates grouped by task type.

B.7 Different ways of representing 2D Paths

To investigate the effect of the number of points on the 2D path, we train the VLM to predict 1. paths simplified using RDP algorithm, which simplify paths in short horizon tasks to 3-5 points and is what we used in the paper. We denote these paths as RDP in the following; 2. Paths represented with 20 points sampled on the path with same step size, denoted as 20p in the following. We keep points where the gripper is executing operation of open or close in both methods.

We train the network on RLBench 80 tasks with 1000 episodes for each task and test it on 25 episodes on the task of close jar. We tried both VILA1.5-3B (denoted as 3B) and VILA1.5-13B (denoted as 13B) as our backbone. Thus we have in total 4 combinations over 2 backbones and 2 designs of path representations. We visualize the result in this Figure B.10.

From this result we can see that when using smaller models, like VILA1.5-3B, paths represented using points extracted using RDP algorithm outperforms paths represented with a fixed number of 20 points significantly. When the network becomes larger to the level of 13B, the VLM is able to handle the representation using 20 points and both two path representations work per-

fectly. We believe that is because when points are simplified using the RDP algorithm, we usually need less points to represent the path and helps the model to pay more attention to predict the accurate position for the gripper open/close points.

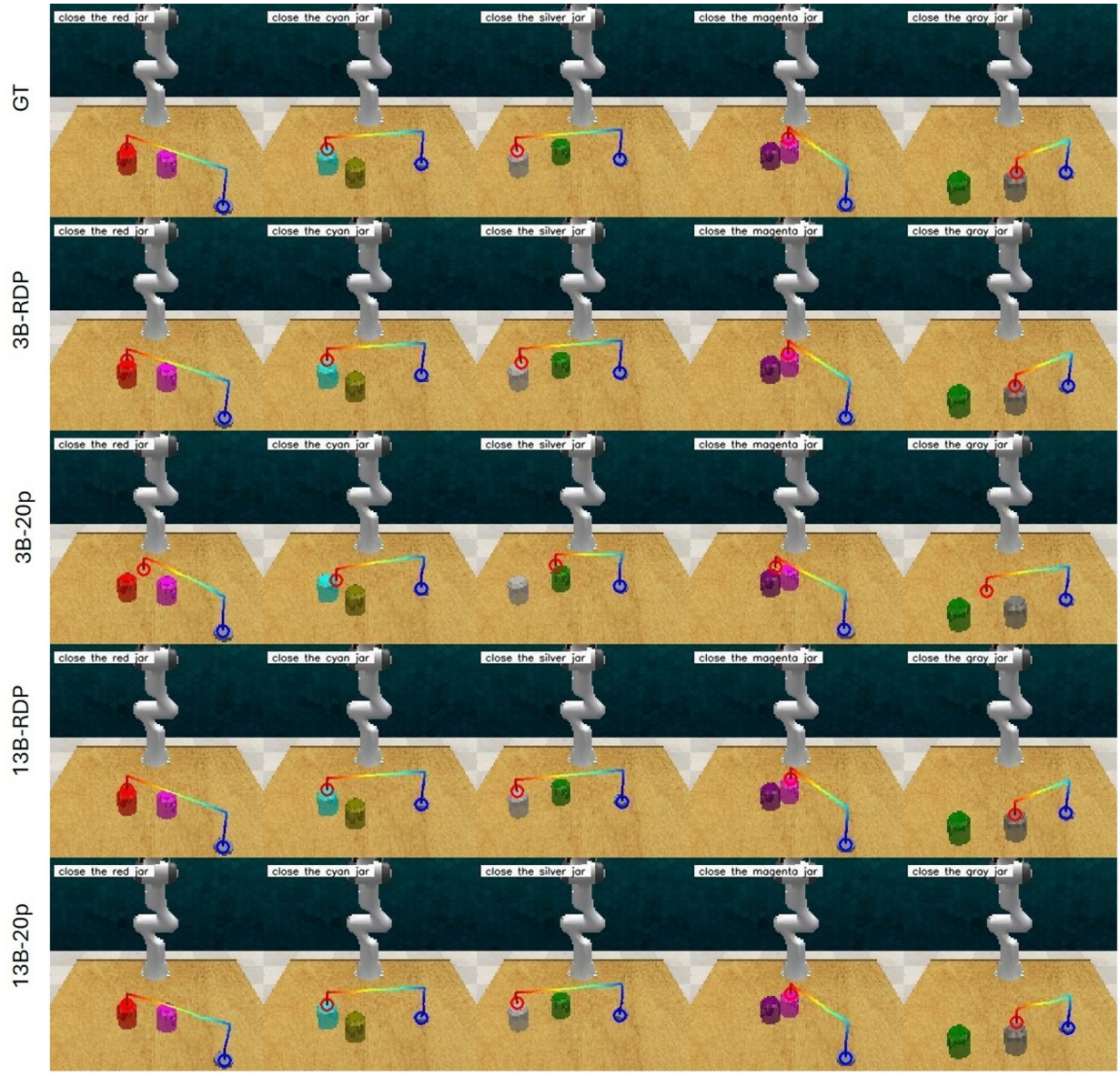


Figure B.10: The task is to pick up the lid and close it on the jar with correct color. Task description is located on the top-left corner of each image. The trajectory goes from blue to red where blue circles denotes where the gripper should close and red circles denotes where the gripper should open. GT denotes ground truth, 3B and 13B denotes VILA1.5-3B and VILA1.5-13B, RDP denotes paths simplified using Ramer–Douglas–Peucker algorithm while 20p denotes paths reprensented using 20 points.