LALA: Learning Object Assembly via Learning to Attend

Hao-Yu Hsu
School of Computer Science
UIUC
haoyuyh3@illinois.edu

I. Abstract

In this project, we tackle on the problem of goal-oriented object assembly. More specifically, our objective is to train a robot to execute the object assembly task by predicting a sequence of pick and place positions based on the specified goal configurations for the target objects. Most goal-oriented object assembly methods require ground-truth states of every object in the workspace and train a graph-based policy. Although that information (e.g., positions and orientations) can be easily acquired from a simulator, it cannot be accurately captured in real-world scenarios, making graph-based policy infeasible. In this work, we utilize multi-view observations of the workspace as input and train a vision-based policy for goal-oriented object assembly. Our model LALA is built upon the Transporter Network, incorporating additional top-down orthographic feature maps and a forward branch for goal observation. Furthermore, we designed customized assembly tasks for this work, including the stacking of a single tower and the palletizing of boxes. The experiment results indicate that LALA performs worse than CLIPort in the stacking tower task and also achieve suboptimal performance in palletizing boxes.

II. Introduction

Object assembly refers to a task that involves constructing or arranging various components to form a complete object. This process is applicable in numerous fields, including manufacturing and robotics automation. Goal-oriented object assembly is a more specialized process, focusing on achieving specific objectives, such as creating a predetermined complete object. Envision a future where robots can assemble furniture from pieces based on user-specified inputs. These inputs could be natural language instructions, such as 'Give me a yellow chair,' a step-by-step manual, or even a series of screenshots of the finished object. Therefore, goal-oriented object assembly represents an exciting direction for robotics research.

Most current methods, as referenced in [4], [3], [11], [10], and [2], rely on input from each component's state within a simulator, training a graph-based policy using graph neural networks. While these approaches have shown promising results in generalizing to unseen targets, they depend on the precise state of each component, which is often impractical in real-world scenarios. In this work, we pivot to using visual observations as input, training a vision-based policy with 2D convolutional neural networks. This approach eliminates the need to extract object-centric representations, such as poses, keypoints, and descriptors, from the scene. Furthermore, it more closely mirrors human assembly processes, which rely on visual observations as the cues for object assembly.

In this work, two customized tasks have been developed: one involves stacking a single tower with five blocks, and the other involves packing boxes onto a wooden pallet. The objective is to execute a series of pick-and-place actions to ensure that the assembled object matches the goal object. Regarding the model architecture, our approach is built upon the Transporter Network [15], enhanced with a top-down feature fusion pipeline and an additional branch for processing goal observations. Our learning paradigm primarily relies on imitation learning, such as behavior cloning, where the model learns from ground truth data for each step of the pick-and-place actions. Experimental results have shown that our method does not outperform CLIPort [12] in stacking the tower and also exhibits suboptimal performance in palletizing boxes. We suggest several potential improvements for this baseline method for future research.

III. Related Work

A. Goal-Oriented Object Assembly Task

[4], [3], [11], [10] and [2] learn graph-based policies for object assembly. This way, all those methods require ground-truth state (e.g. positions, rotated angles) of each object in the scene from the simulator to further form the complete graph representation. In contrast, our method relies fully on RGB-D observations from multiple angles, thus it does not require extracting object-centric representations.

[9] and [6] formulate the object assembly problem to be a much harder robot manipulation task. They aim to solve tasks such as assembling household furniture and learn a more complex action like fastening the screws. In this project, we do not consider this type of problem since it would be too difficult to learn a policy from visual observations.

B. Multi-View Robot Manipulation

[1] learns visuomotor policies from multi-view expert demonstrations and demonstrates that policies learned from multiview data have spatially correlated visual features. [8] employs contrastive learning to establish a shared latent space among multiple viewpoints for robot control tasks. [7] attends

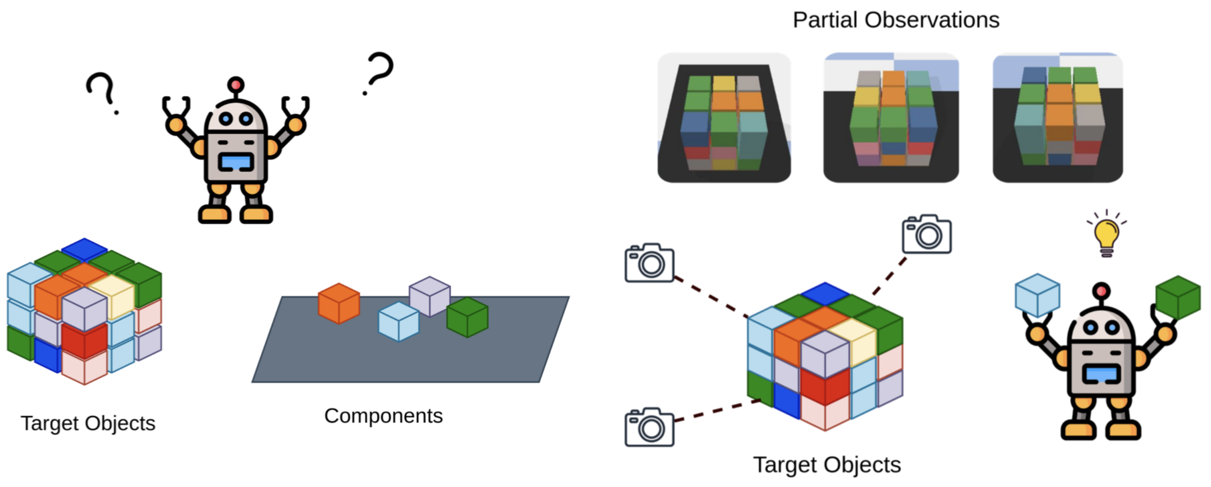

Fig. 1: **An overview of LALA**. Our model takes current observation and targegt observationfrom the workspace as inputs, and learns a policy, denoted as π, by predicting a series of pick-and-place actions to assemble the objects.

to both egocentric and third-person views using a transformer for robot manipulation.

RVT [5] first renders multiple virtual images from unprojected point clouds in the workspace, then uses Transformer to aggregate those virtual views as input for downstream robot manipulation. MIRA [14] instead renders virtual images from pre-trained NeRF in the workspace. Our input & output format would be similar to both RVT and MIRA, but the main difference lies in the target conditions. Our method is conditioned on goal configuration of target objects, while MIRA does not have a specific condition and RVT is conditioned on text instructions.

C. Transporter Network

TransporterNet [15] has exhibited exceptional spatial precision in robotic manipulation by predicting heatmap distributions from top-down orthographic views of the current workspace. Moreover, this 2D prior allows for data-efficient learning, enabling the model to generalize from as few as 10 training datasets. CLIPort [12], on the other hand, integrates the pre-trained CLIP model with TransporterNet, to achieve end-to-end learning for fine-grained manipulation with multi-goal and multi-task generalizaiton capabilities of vision-language grounding systems. Aiming for better learning efficiency and generalizability in our model, we based our design on the TransporterNet and further restrict our action primitives to be top-down pick-and-place.

IV. Methods

LALA is an imitation-learning agent based on three key principles: (1) Manipulation through a two-step primitive where each action involves a start and final end-effector pose. (2) Visual representations of actions that are equivariant to translations and rotations. (3) Two separate pathways for current and goal workspace observation. Combining (1) and (2) from Transporter with (3) allows us to achieve generalizable policies that adapt to different goal configurations. See Figure-1 for an overview of our problem formulation.

Section A., Section B. and Section C. describes the problem formulation, gives an overview of Transporter [15], and presents our goal-oriented model. Section B. provides details on the training approach.

A. Goal-Oriented Manipulation

We consider the problem of learning a goal-conditioned policy π that outputs actions $\mathbf{a}_t$ given input $\gamma_t = (\mathbf{o}_t, \mathbf{g}_t)$ consisting of a current visual observation $\mathbf{o}_t$ and a goal visual observation $\mathbf{g}_t$:

$$\pi(\gamma_t) = \pi(o_t, g_t) \rightarrow a_t = (\mathcal{T}_{\text{pick}}, \mathcal{T}_{\text{place}}) \in \mathcal{A} \qquad (1)$$

The actions $\mathbf{a} = (\mathcal{T}_{\text{pick}}, \mathcal{T}_{\text{place}})$ specify the end-effector pose for picking and placing, respectively. We consider tabletop tasks where $\mathcal{T}_{\text{pick}}, \mathcal{T}_{\text{place}} \in \mathbf{SE}(2)$. The visual observation $\mathbf{o}_t$ and $\mathbf{g}_t$ are top-down orthographic RGB-D reconstruction of the current scene and goal scene where each pixel corresponds to a point in 3D space. To further address visual occlusion in top-down RGB-D reconstruction, we enhance the visual observation $\mathbf{o}_t$ and $\mathbf{g}_t$ by incorporating additional tabletop feature maps. These maps are generated by fusing back-projected pixel-level visual features from each viewpoint into a three-dimensional space. See Figure-2 for more details on feature fusion.

B. Transporter for Pick-and-Place

The policy π is trained with Transporter [15] to perform spatial manipulation. The model first (1) attends to a local region to decide where to pick, then (2) computes a placement location by finding the best match through cross-correlation of deep visual features.

Following Transporter, the policy π consists of two-action-value modules (i.e., Q-functions): The pick module $\mathcal{Q}_{\text{pick}}$ decides where to pick, and conditioned on this pick action, the

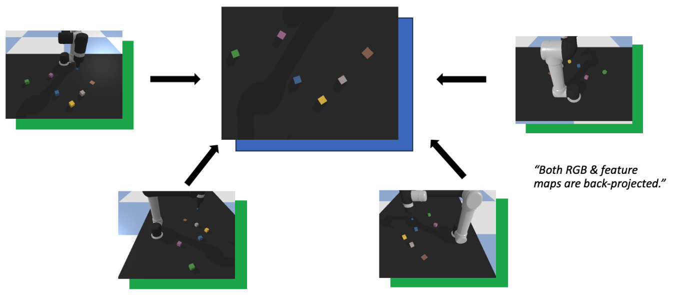

Fig. 2: **Feature fusion proposed in this project.** Essentially, we forward input images from each viewpoint into a pre-trained ResNet50 to obtain the feature maps. Then, we cast those features into 3D space using known camera parameters. Finally, we fuse those unprojected features into top-down orthographic feature maps.

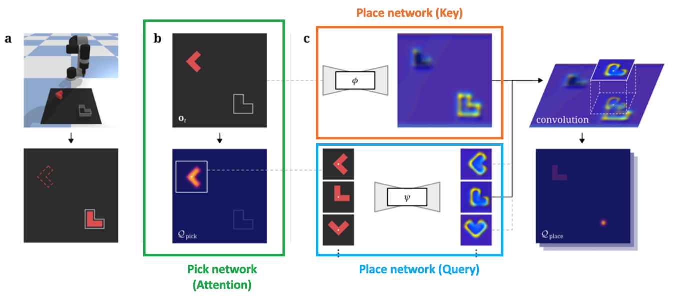

Fig. 3: An overview of original Transporter model architecture.

place module Q_{place} decides where to place. These modules are implemented with Fully-Convolutional-Networks (FCNs) that are translationally equivariant by design. Details on how we extend these networks to handle goal observation will be covered later.

The pick FCN f_{pick} takes input $\gamma_t = (\mathbf{o}_t, \mathbf{g}_t)$ and outputs a dense pixelwise prediction $Q_{\text{pick}} \in R^{H\times W}$ of action-values, where are used to predict the pick action T_{pick}:

$$T_{\text{pick}} = \underset{(u,v)}{\operatorname{argmax}}\ Q_{\text{pick}}((u,v) \mid \gamma_t) \tag{2}$$

Since $\mathbf{o}_t$ and $\mathbf{g}_t$ are obtained from orthographic projections, each pixel location (u, v) can be directly mapped to a picking location in 3D using the known camera parameters. f_{pick} is trained in a supervised manner to predict the pick action T_{pick} that imitates the expert demonstration on picking objects at timestep t.

The second FCN f_{query} takes in $\gamma_t[T_{\text{pick}}]$, which is a $c \times c$ crop of $\mathbf{o}_t$ and $\mathbf{g}_t$ centered at T_{pick}, and outputs a query feature embedding of shape $c \times c \times d$. The third FCN f_{key} consumes the full input γ_t and outputs a key feature embedding of shape $H \times W \times d$. The place action-values Q_{place} are then computed by cross-correlating the query and key features:

$$Q_{\text{place}}(\tau|\gamma_t, T_{\text{pick}}) = (f_{\text{query}}(\gamma_t[T_{\text{pick}}]) * f_{\text{key}}(\gamma_t))[\tau] \tag{3}$$

where $\tau \in \mathbf{SE}(2)$ represents a possible placement pose. Since $\mathbf{o}_t$ is from orthographic projection, rotations in the placement pose τ can be captured by batching k discrete angle rotations of the crop before passing it through the query network f_{query}. Then $T_{\text{place}} = \operatorname{argmax}_\tau Q_{\text{place}}(\tau|\gamma_t, T_{\text{pick}})$, where the place module is trained to imitate the placements in the expert demonstrations. See Figure-3 for an overview of the original Transporter architecture. More details could be found in the original paper [15].

C. *Goal-Oriented Transporter*

In LALA, we extend the network architecture of all three FCNs f_{pick}, f_{query} and f_{key} from Transporter [15] to allow for additional goal inputs. We duplicate the FCNs into two streams, the current stream processes the current observation, while the goal stream handles the goal observation. Both streams are identical to the ResNet architecture in Transporter – a tabula rasa network that takes in RGB-D and fused visual features input $\mathbf{o}_t$ and outputs dense features through an hourglass encoder-decoder model. See Figure-4 for an overview of our model architecture.

To better capture the relations between the current and the goal observations, we add lateral connections from the current stream to the goal stream. These connections involve concatenating two feature tensors together and applying a 1×1 convolution layer to reduce the dimension to be compatible to the next layer. For the final fusion of dense features between two streams, addition for f_{pick} and 1×1 convolution fusion for f_{query} and f_{key}.

D. *Learning from Demonstrations*

We assume access to a dataset $\mathcal{D} = \{\xi_1, \xi_2, \ldots, \xi_n\}$ of n expert demonstrations with associated discrete-time input-action pairs $\xi_i = \{(\mathbf{o}_1, \mathbf{g}, \mathbf{a}_1), (\mathbf{o}_2, \mathbf{g}, \mathbf{a}_2), ...\}$ where $\mathbf{a}_t = (T_{\text{pick}}, T_{\text{place}})$ corresponds to expert pick-and-place coordinates at timestep t, and the goal observation $\mathbf{g} = \mathbf{o}_T$ where T represents the last timestep in this demonstration. These expert demonstrations are used to supervise the policy π.

During training, we randomly sample an input-action pair from the dataset and supervise the model end-to-end with one-hot pixel encodings of demonstration actions Y_{pick} and Y_{place} with k discrete rotations. The model is trained with cross-entropy loss:

$$V_{\text{pick}} = \text{softmax}(Q_{\text{pick}}((u, v) \mid \gamma_t)) \tag{4}$$

$$V_{\text{place}} = \text{softmax}(Q_{\text{place}}((u', v', w') \mid \gamma_t, T_{\text{pick}})) \tag{5}$$

$$L = -E_{Y_{\text{pick}}}[\log V_{\text{pick}}] - E_{Y_{\text{place}}}[\log V_{\text{place}}] \tag{6}$$

Compared to the original Transporter models that were trained for 40K iterations, we train our models for only 20K iterations with data augmentation such as randomized colors. The models are trained on a single 48GB A6000 GPU for 1.5 days with batch size of 1.

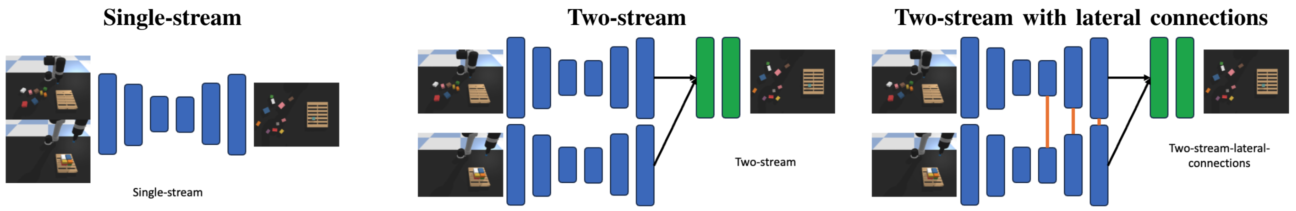
Single-stream
Single-stream
Two-stream
Two-stream
Two-stream with lateral connections
Two-stream-lateral-connections

Fig. 4: **Goal-oriented transporter.** We duplicate the FCNs into two streams to accommodate additional goal input. Each stream processes the current observation and the goal observation, respectively, followed by a feature fusion layer in the final stage.

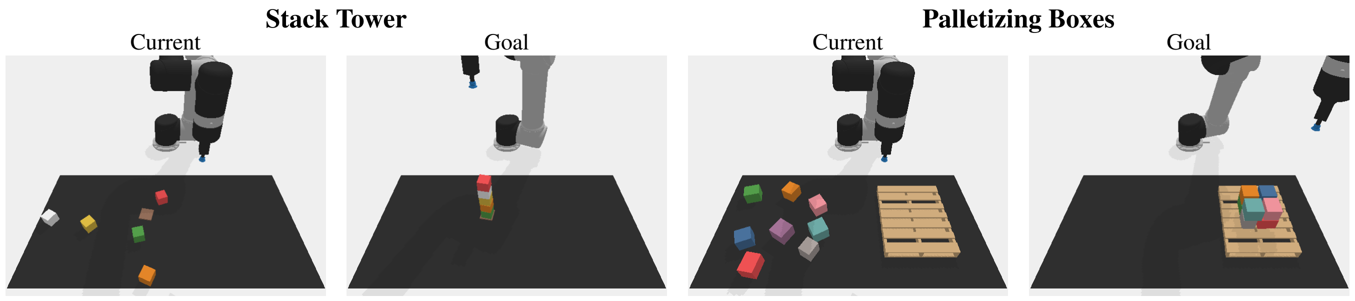

Fig. 5: Visualizations of simulated environment for stacking-tower and palletizing-boxes task.

V. Simulation Environment Setup

A. Environment

The environment codebase primarily follows CLIPort implementation framework (https://github.com/cliport/cliport). All simulated experiments are based on a Universal Robot UR5e with a suction gripper. The basic input observation is a top-down RGB-D reconstruction from 4 cameras positioned around a rectangular workspace: one in the front, one on the left shoulder, one on the right shoulder and one from the back, all pointing towards the center. The configurations of the virtual cameras are aligned with the RealSense D415 settings, featuring an image resolution of 640×480.

B. Goal-Oriented Object Assembly Tasks

We customized two object assembly manipulation tasks from the Ravens benchmark in PyBullet: one task involves stacking a tower, and the other involves packing boxes onto a wooden pallet. Each task instance is constructed by sampling a set of block attributes, including poses, colors, and sizes. See Figure-5 for sampled visualizations of each task.

For the **stacking-tower** task, we begin by sampling a location in the workspace to set the base plate. We then create five blocks with distinct colors, sampled from the color set T_{color}, which includes $\{$blue, red, green, orange, yellow, purple, pink, cyan, brown, white, gray$\}$. Next, we stack these blocks onto the base sequentially to construct a tower. Finally, the blocks are repositioned and reoriented in the free space of the workspace in a top-to-bottom order.

For the **palletizing-boxes** task, we start by placing a wooden pallet on the right side of the workspace to serve as the base. We then generate multiple blocks around a fixed-size region with size $= (0.15, 0.15, 0.10)$ centered at the middle of the wooden pallet. The generation of boxes is conducted by running a KD-Tree splitting algorithm, with the minimum split block size in each dimension set to 0.5. Note that the color of each box is sampled from the same color set as before. Finally, in the same manner as the previous task, we reset the position and orientation of each box based on the height of the box.

C. Evaluation Metric

We adopt the scoring range from 0 (fail) to 100 (success) as proposed in the Ravens benchmark [15]. This score awards partial credit based on task completion, for example, packing 3 out of 5 specified objects in the goal observation is scored as $3/5 \rightarrow 60.0$. In the Ravens benchmark, the palletizing-boxes task is evaluated by computing the overlap of the occupied regions, regardless of the block attributes. Consequently, we have slightly modified the evaluation metric for this task. A single placement will be deemed successful only if the block is positioned at the exact same location as in the goal configurations, within the designated error threshold for both position and rotation. During an evaluation episode, an agent continues interacting with the scene until an oracle agent indicates task completion. We report scores based on 100 evaluation runs for agents that have been trained with 500 demonstrations.

VI. Experimental Results

The evaluation results are presented in Table-I. We evaluated three versions of our methods: the single-stream method, the two-stream method, and the two-stream method with lateral connections. The single-stream method shares the same model architecture as the Transporter, but it includes an additional

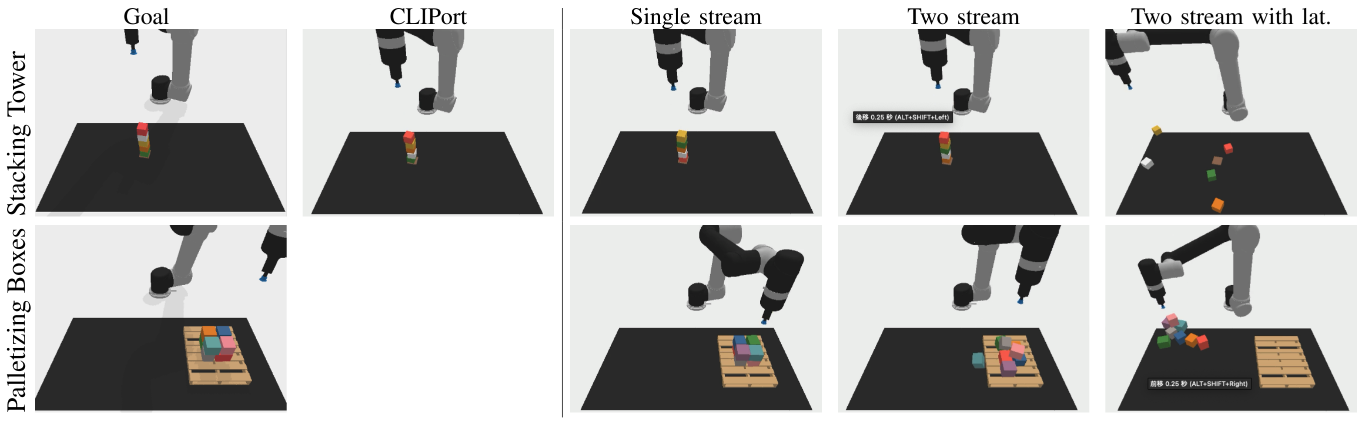

Fig. 6: Comparisons of robot manipulation results on stacking tower and palletizing boxes.

goal observation as input. We derive the final input by subtracting the goal input (both RGB-D and fused features) from the current input. In contrast, the two-stream method employs two separate networks to process the current input and the goal input, respectively. The two-stream method with lateral connections further incorporates a lateral connection branch to integrate the goal features into the current features during the decoder phase of the hourglass ResNet architecture.

A. *Stacking Tower*

We also compare our method with CLIPort [12] in the stacking tower task. For a fair comparison, per-step instructions such as *'Put the green block onto the yellow block'* are not provided. Instead, we use goal instructions to represent the entire object stacking process. An example instruction is *'Make the tower with yellow, brown, green, blue, red blocks'*. The evaluation results indicate that CLIPort outperforms all three of our methods. This outcome is reasonable, considering that CLIPort utilizes a pre-trained CLIP feature to encode the RGB image and language instructions. In terms of our methods, the two-stream architecture slightly outperforms the single-stream method. However, the two-stream method with lateral connections fails to even complete a correct placement.

B. *Palletizing Boxes*

In the palletizing-boxes task, our method demonstrates limited effectiveness. The mean reward achieved is approximately 0.06, which falls short of the average for correctly placing one box in each round (calculated as $1/8 = 0.125$). The two-stream method still slightly outperforms the single-stream method, yet it fails to achieve even a single correct placement when augmented with the additional lateral connections design.

C. *Discussions*

From the visualization of the assembly result in Figure-6, we observe that both the single-stream and two-stream methods can form a tower successfully, but they do not follow the goal configuration. I propose that this is due to

Method	Stacking Tower	Palletizing Boxes
CLIPort [12]	27.0	x
Single stream	13.4	6.0
Two stream	15.8	6.3
Two stream with lat.	0.0	0.0

TABLE I: **Quantitative evaluation on two custom tasks – stacking tower and palletizing boxes.** The average reward of all 100 evaluation runs is reported. The scoring range from 0 (fail) to 100 (success).

two primary reasons: firstly, the networks do not utilize pre-trained weights. A potential solution could involve training a self-supervised reconstruction task to predict a patch in one view from visual observation cues in another view. Secondly, there is the issue of the ambiguity in the placement action. In the palletizing boxes task, after completing the first layer of palletization, any box could be selected for the next step. However, during training, only one box receives supervision. While the randomness in box selection might be mitigated with numerous demonstrations, this does not fully address the ambiguity in the choice of the next box. A potential solution could be to heavily supervise the model with all possible actions it could take.

Regarding why the lateral connections negatively impact task performance, I suggest that it is because our networks are trained from scratch. Consequently, the lateral connections could overcomplicate the networks and bring more instability, especially in producing spatially-consistent top-down feature maps.

VII. Conclusion

In this project, we propose a novel problem setting, termed **Multi-view Goal-oriented Object Assembly for Robot Manipulation**. To address this challenge, we introduce a goal-oriented transporter network. Additionally, we have designed two customized tasks-stacking a tower and palletizing boxes—to evaluate the efficacy of our model. Although our method currently does not surpass CLIPort in stacking tower tasks and shows suboptimal performance in palletizing boxes, several potential enhancements have been identified. Firstly, we could implement pre-training by introducing several auxiliary losses prior to the imitation learning phase to effectively 'warm up' the parameters. Secondly, the incorporation of the RVT (Robotic Vision Transformer) framework could be beneficial to facilitate attention across local patches from multiple views. Finally, following the UniSim [13] approach, we could simulate rollouts of the object assembly process, thereby enabling downstream action planning.

References

[1] Trevor Ablett, Yifan Zhai, and Jonathan Kelly. Seeing all the angles: Learning multiview manipulation policies for contact-rich tasks from demonstrations. 2021.

[2] Hyunsoo Chung, Jungtaek Kim, Boris Knyazev, Jinhwi Lee, Graham W. Taylor, Jaesik Park, and Minsu Cho. Brick-by-brick: Combinatorial construction with deep reinforcement learning, 2021.

[3] Niklas Funk, Georgia Chalvatzaki, Boris Belousov, and Jan Peters. Learn2assemble with structured representations and search for robotic architectural construction. *CoRL*, 2021.

[4] Seyed Kamyar Seyed Ghasemipour, Daniel Freeman, Byron David, Shixiang Shane Gu, Satoshi Kataoka, and Igor Mordatch. Blocks assemble! learning to assemble with large-scale structured reinforcement learning. *ICML*, 2022.

[5] Ankit Goyal, Jie Xu, Yijie Guo, Valts Blukis, Yu-Wei Chao, and Dieter Fox. Rvt: Robotic view transformer for 3d object manipulation. *CoRL*, 2023.

[6] Minho Heo, Youngwoon Lee, Doohyun Lee, and Joseph J. Lim. Furniturebench: Reproducible real-world benchmark for long-horizon complex manipulation. In *Robotics: Science and Systems*, 2023.

[7] Rishabh Jangir, Nicklas Hansen, Sambaran Ghosal, Mohit Jain, and Xiaolong Wang. Look closer: Bridging egocentric and third-person views with transformers for robotic manipulation, 2022.

[8] Akira Kinose, Masashi Okada, Ryo Okumura, and Tadahiro Taniguchi. Multi-view dreaming: Multi-view world model with contrastive learning, 2022.

[9] Youngwoon Lee, Edward S Hu, and Joseph J Lim. IKEA furniture assembly environment for long-horizon complex manipulation tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021. URL https://clvrai.com/furniture.

[10] Richard Li, Allan Jabri, Trevor Darrell, and Pulkit Agrawal. Towards practical multi-object manipulation using relational reinforcement learning, 2020.

[11] Yixin Lin, Austin S. Wang, Eric Undersander, and Akshara Rai. Efficient and interpretable robot manipulation with graph neural networks. *ICRA*, 2022.

[12] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. In *Proceedings of the 5th Conference on Robot Learning (CoRL)*, 2021.

[13] Mengjiao Yang, Yilun Du, Kamyar Ghasemipour, Jonathan Tompson, Dale Schuurmans, and Pieter Abbeel. Learning interactive real-world simulators. *arXiv preprint arXiv:2310.06114*, 2023.

[14] Lin Yen-Chen, Pete Florence, Andy Zeng, Jonathan T. Barron, Yilun Du, Wei-Chiu Ma, Anthony Simeonov, Alberto Rodriguez Garcia, and Phillip Isola. MIRA: Mental imagery for robotic affordances. In *Conference on Robot Learning (CoRL)*, 2022.

[15] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Ayzaan Wahid, Vikas Sindhwani, and Johnny Lee. Transporter networks: Rearranging the visual world for robotic manipulation, 2020.