

# RoboPearls: Editable Video Simulation for Robot Manipulation

## Supplementary Material

### Contents

<b>A.1. Social Impact and Limitations.</b>	<b>13</b>
A.1.1. Future Work	13
A.1.2. Limitations	13
A.1.3. Scope	14
<b>A.2. Additional Related Work</b>	<b>14</b>
A.2.1. LLM Agent	14
<b>A.3. Additional Details</b>	<b>14</b>
A.3.1. More Details on Dynamic Gaussians	14
A.3.2. More Details on Editable Video Simulation	14
A.3.3. More Details on the LLM Agents	15
A.3.4. More Details on the VLM	15
A.3.5. Experimental Setting	15
A.3.6. Real World Robot Setup	15
<b>A.4. Additional Results</b>	<b>16</b>
A.4.1. More Results on Visual Metrics	16
A.4.2. More Simulation Results	16
A.4.3. More Manipulation Tasks Results	16
<b>A.5. Additional Visualizations</b>	<b>17</b>
A.5.1. Spatial-temporal Consistency	17
A.5.2. Identity Encoding Features	17

### A.1. Social Impact and Limitations.

#### A.1.1. Future Work

As this paper is the first to construct photo-realistic simulations, there remain many benefits yet to be explored.

- First, our approach inherently supports novel view synthesis. While RVT [20] leverages point clouds to generate virtual images, our method enables the rendering of realistic images from new viewpoints, potentially unlocking significant advancements for models like RVT. More recently, Act3D [19] and SAM2Act [16] have leveraged foundation models, such as CLIP and SAM, to extract image embeddings for robotic manipulation. However, these single-image foundation models lack a fundamental understanding of the 3D scene and are sensitive to view variations, leading to the generation of noisy and view-inconsistent masks or features, which in turn cause manipulation failures. In contrast, our RoboPearls distills open-world semantics into 3D space and can render view-consistent semantic features, as shown in Fig. 14, offering the potential to further improve the performance of recent sota models.

- Second, the interaction perspective remains largely unexplored. Existing methods, such as GraspSplats [30], use 3DGS to reconstruct static scenes, whereas our approach handles dynamic environments with semantic features, allowing for more effective generation of grasping proposals using explicit Gaussian primitives.
- Third, recent vision-language-action (VLA) models based on VLMs, such as Hi Robot [61] and Helix [18], have garnered significant attention for their ability to process complex instructions and generate dexterous manipulations. However, collecting and scaling vision-language data remains a significant challenge. Our RoboPearls framework allows users to generate complex simulations using natural language instructions, which enables the effective expansion of vision-language-aligned datasets and potentially enhances model performance.
- Fourth, our work advances the goal of realistic closed-loop simulation. Currently, we utilize VLM to enhance training performance in a closed-loop manner. However, our high-quality reconstruction and rendering capabilities enable robust closed-loop policy evaluation, and pave the way for developing a GS-based closed-loop reinforcement learning (RL) training paradigm to further improve robotic learning.

#### A.1.2. Limitations

Beyond these promising directions, there are also limitations that we aim to address in future work.

- The primary challenge lies in generalization, as our approach relies on scene-specific training for each environment. Fortunately, recent advancements in Generalizable Gaussian Splats [7, 10] provide a promising avenue to mitigate this issue.
- Moreover, as shown in Tab. 5, the training and processing time for high-resolution real-world scenes is still somewhat lengthy. Fortunately, recent methods [43, 91] have significantly optimized and accelerated Gaussian fitting, offering the potential for further improvements.
- Additionally, while incorporating VLM reduces reliance on human experts to some extent, handling highly complex scenes remains challenging. A practical approach is to first utilize VLM to enhance model performance and only involve human intervention when necessary. This strategy is still an improvement over previous methods that relied entirely on human analysis.

Table 5. **Processing time of each module.** The time is evaluated on Ego4d real-world scene with 120 frames on the resolution of  $1918 \times 1237$  on 1 GPU.

Module	Recon	Insert	Remove	Color	Texture	Physics
Time (min)	~70	~6	~6	~1	~5	~7

### A.1.3. Scope

RoboPearls supports various scene edits (Fig. 2 (b) and Sec. 3.3), except for action-relevant edits, more exactly, generating new manipulation trajectories, which is out of scope. We leave action editing for future work.

Altogether, we hope that our work will inspire further research and contribute to the advancement of robotic simulation and learning.

## A.2. Additional Related Work

### A.2.1. LLM Agent

Significant advances in Large Language Models (LLMs), e.g., GPT-4 [1] and DeepSeek-R1 [24], have demonstrated their remarkable capabilities across various domains. By integrating LLMs as agents, many works [46, 56, 83] have enhanced problem-solving abilities in interactive and autonomous applications. For example, AutoGen [75] leverages well-organized LLM agents to form operating procedures and code programming. ChatSim [71] adopts an LLM-agent collaboration workflow for editing 3D driving scenes, and RoboGen [70] uses generative models and LLMs to generate robotic tasks. In this work, we utilize LLM agents to decompose user simulation demands into concrete commands for specific editing functions, thereby automating and streamlining the simulation process.

## A.3. Additional Details

### A.3.1. More Details on Dynamic Gaussians

To integrate semantic information into the dynamic Gaussians, it is worth noting that the identity encoding  $e$  is independent of timestamp  $t$  and remains unchanged throughout the time series. Moreover, during densification, newly generated Gaussian primitives inherit the identity encoding of their predecessors. This ensures that Gaussian primitives associated with a specific object do not acquire the identity labels of other objects over time, thereby maintaining spatiotemporal consistency.

### A.3.2. More Details on Editable Video Simulation

**Incremental Semantic Distillation.** The pipeline is in Fig. 6. After retrieving the desired object Gaussians, we render the 2D object mask and use G-DINO to verify whether it corresponds to the desired object. If the target

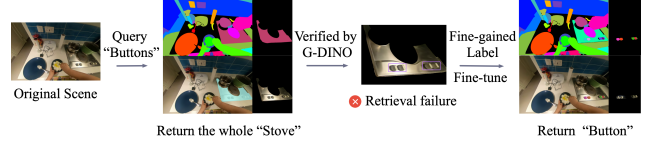


Figure 6. **The incremental semantic distillation pipeline** (zoom-in for the best of views).

object is not identified, we further use bounding boxes as prompts to SAM for a finer-grained segmentation and fine-tune the identity encoding of the retrieved object Gaussians.



Figure 7. **The object removal pipeline.**

**Object Removal.** The pipeline is shown in Fig. 7. After deleting the 3D object Gaussians, we use G-DINO to detect the “blurry hole” and apply LAMA inpainting on each view. Then, we generate new Gaussians near the deletion area and fine-tune only these newly introduced Gaussians with the inpainted views.

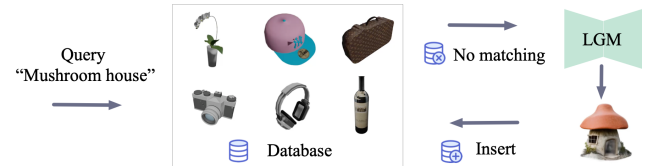


Figure 8. **The 3D asset management pipeline.**

**3D Asset Management.** The pipeline is shown in Fig. 8. The 3D Asset Management agent first retrieves desired 3D objects from the Gaussian database by matching object attributes such as color and type. If the matching is unavailable, the agent employs a Gaussian object generation model, e.g., LGM, to synthesize the desired object, and then incorporate it into the database.

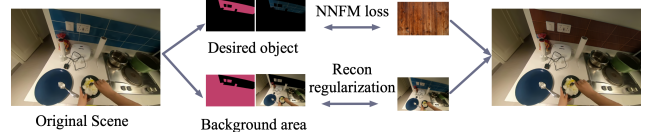


Figure 9. **The texture modification pipeline** (zoom-in for the best of views).

**Texture Modification.** The pipeline is shown in Fig. 9. We render masks of the target 3D object to optimize only the SH parameters of the object Gaussians with the NNFM loss

while preventing artifacts on regions outside the mask with the original reconstruction loss.

### A.3.3. More Details on the LLM Agents

In Fig. 10, we provide detailed prompts and examples of the LLM agents. Each agent is equipped with unique LLM prompts and tailored system functions to their specific duties. All tailored agents collaborate to execute the simulation based on user commands following a sequential pipeline. Additionally, the Simulation Manager Agent records all simulation configurations, enabling multi-round editing and further refinements as needed.

### A.3.4. More Details on the VLM

In Fig. 11, we provide detailed prompts and examples of the VLM. The VLM is prompted to analyze keyframes of failure cases and provide detailed explanations across potential failure causes, such as object position, color, and background texture. Then the VLM further generates corresponding simulation solutions in natural language, which are then fed into our simulation framework to generate targeted simulations for enhancing model training.

### A.3.5. Experimental Setting

**Datasets and Simulation Setup.** Our experiments are conducted in the popular multi-task manipulation benchmark RL-Bench [29] and the generalization evaluation benchmark COLOSSEUM [55]. RL-Bench is built on the CoppeliaSim [58] simulator, where a Franka Panda Robot is controlled to manipulate the scene. The benchmark contains 100 tasks, including picking and placing, high-accuracy peg insertions. Each task is specified by a language description and consists of 2 to 60 variations, which concern scene variability across object poses, appearance, and semantics. There are four RGB-D cameras positioned at the front, wrist, left shoulder, and right shoulder. The evaluation metric is the task completion success rate, which is the proportion of execution trajectories that achieve the goal conditions specified in the language instructions [19, 62]. COLOSSEUM is a benchmark for evaluating the generalization of robotic manipulation. It introduces 13 perturbations across 20 different tasks from the RL-Bench framework such as *close box*, and *basketball in hoop*. These perturbations include changes in color, texture, size of objects and backgrounds, lightnings, distractors, and camera poses. For simulation, due to limited computing resources, we utilize a curated subset of 6 challenging language-conditioned manipulation tasks, and follow the original COLOSSEUM setup and generate 13 environmental perturbations for each task. We also report the task completion success rate on COLOSSEUM.

**Baselines.** We compare RoboPearls with various models that have been specifically designed for 3D object manipu-

lation including RVT [20], RVT2 [21], and SAM2Act [16], which are the previous SOTA methods on RL-Bench and COLOSSEUM. Since SAM2Act [16] does not provide the code and model for COLOSSEUM, we only conducted comparisons with it on RL-Bench.

**Implementation Details.** Following RVT [20], we train RoboPearls for 100k steps, using the LAMB optimizer [86] optimizer, with an initial learning rate of  $5e-4$ . We also adopt a cosine learning rate decay schedule with warm-up in the first 2k steps. We use the SE(3) [62, 89] augmentation, i.e., translation augmentation of 12.5 cm along the  $x$ ,  $y$ , and  $z$  axis and rotation augmentation of  $45^\circ$  along the  $z$  axis, for expert demonstrations training to enhance the generalizability of policies. For visual observation, we employ RGB-D images with a resolution of  $128 \times 128$ . All the compared methods are trained on 8 NVIDIA A100 GPUs with a batch size of 24. We use 96 demonstrations per task for training and 25 unseen demonstrations for testing. Due to the randomness of the sampling-based motion planner, we evaluate each model 10 times for each task and report the average success rate and standard deviation.

**SfM details.** For the Ego4D dataset, we successfully applied Colmap in most cases, as the moving egocentric view provides sufficient multiple views. In challenging cases with sparse views or large motion, Colmap may fail; we then use DUST3R [68] as a fallback. For the fixed-view Open X-Embodiment dataset, Colmap underperforms, and we instead use DUST3R with good results.

### A.3.6. Real World Robot Setup

**Tasks Setup.** Our real-world experimental setup consists of a Kinova Gen3 ultra-lightweight robotic arm with two Realsense D435i cameras: one mounted on the wrist to provide a first-person perspective, and the other positioned opposite the robotic arm to offer a third-person view. We collect RGB-D frame-action data for three tasks, including: “Pick up”-*Pick up the yellow block*, “Place in”-*place the yellow block between the blue and red blocks*, and “Put on”-*put the yellow block to the green subregion*. For each task, we recorded 15 demonstrations, capturing visual, state, and action data. The models are tested with seen/unseen objects to further evaluate generalization capability.

**Implementation Details.** We fine-tuned our model on the collected dataset using RDT [40], enabling effective generalization across different task variations. RDT-1B, the largest imitation learning Diffusion Transformer to date, features 1 billion parameters and is pre-trained on over 1 million multi-robot episodes. It processes language instructions and RGB images from up to three viewpoints to predict the next 64 robot actions. RDT-1B is highly versatile, supporting a wide range of modern mobile manipulators, including single-arm and dual-arm systems, joint-based and







	<p><b>Simulation Manager Agent</b></p> <p><b>Role:</b> RoboPearls is an editable video simulation framework for robotic manipulation, built upon 3DGS to support various simulation operations. You are the simulation manager responsible for decomposing user simulation commands into concrete natural language instructions and dispatching tasks to the relevant simulation agents.</p> <p><b>Input:</b> User simulation commands.</p> <p><b>Task:</b> The user requires editing a simulation in a robotic scenario. Your job is to break this down into one or more supportable simulation tasks and translate them into natural language instructions for the corresponding agents. The five supportable agents include: &lt;grounding agent&gt;: retrieves the target object, &lt;scene operation agent&gt;: supports operations such as object insertion, deletion, and modification (size, position, color, texture), physics simulation; &lt;3D asset management agent&gt;: organizes and retrieves 3D assets; &lt;scene refiner agent&gt;: refines the scene; &lt;scene renderer agent&gt;: renders scene from desired perspectives.</p> <p><b>Instructions:</b> Please retain all the semantics and adjunct words from the original text. Split the tasks and natural language instructions into a dictionary, where the key is the agent, and the value is the corresponding natural language instruction. These instructions will be executed sequentially, and the tasks should be independent of each other.</p> <p>Example: Input: Insert a red cup to the right of the table, and adjust the view to focus on the cup. Output:{"&lt;grounding agent&gt;": "the right of the table", "&lt;3D asset management agent&gt;": "red cup", "&lt;scene operation agent&gt;": "insert the red cup to the right of the table", "&lt;scene refiner agent&gt;": "refine the scene", "&lt;scene renderer agent&gt;": "render the scene with the view focused on the cup"}</p> <p><b>Output Format:</b> {"&lt;xxx agent&gt;": "natural language instructions"}</p>
	<p><b>Grounding Agent</b></p> <p><b>Role:</b> You are the grounding agent, responsible for retrieving objects based on user natural language instructions.</p> <p><b>Input:</b> User grounding commands.</p> <p><b>Task:</b> Call the system's retrieve function based on the user's natural language instructions to retrieve the target object.</p> <p>Retrieve Function:Parameters: { "target": "xxx" } Return:{"object": "object category id", "3D location": "(x, y, z)", "object size": "(l, w, h)"}</p> <p><b>Instructions:</b> This agent is used to locate and ground a 3D object in the 3D space based on the user's input. Note that the 3D locations of objects can be used to calculate the distance between two objects: <math>d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}</math>.</p> <p><b>Output Format:</b> {"object": "object category id", "3D location": "(x, y, z)", "object size": "(l, w, h)"}</p>
	<p><b>Asset Manager</b></p> <p><b>Role:</b> You are the 3D asset agent, responsible for matching objects from the database or generating objects based on user natural language instructions.</p> <p><b>Input:</b> User new object commands.</p> <p><b>Task:</b> Call the system's matching function based on the user's natural language instructions to retrieve the target object from the database. If the return value is NAN, call the system's generation function.</p> <p>Matching Function: Parameters: { "target": "xxx" } Return: { "object": "object Gaussians" } or NAN; Generation Function: Parameters: { "target": "xxx" } Return: { "object": "object Gaussians" }</p> <p><b>Output Format:</b> {"object": "object Gaussians" }</p>
	<p><b>Scene Operation</b></p> <p><b>Role:</b>You are the scene operation agent, responsible for performing various editing operations based on user natural language instructions.</p> <p><b>Input:</b> User editing commands.</p> <p><b>Task:</b> Call the system's corresponding editing function based on the user's natural language instructions.</p> <p>Insert Function:Parameters: { "object Gaussians": "xxx", "position": "xxx" }; Delete Function:Parameters: { "target": "xxx" }; Modification Function:Parameters: { "target object": "xxx", "type": "xxx", "size": "xxx", "position": "xxx", "color": "xxx", "texture": "xxx" }; Physics Simulation:Parameters: { "target object": "xxx", "physics": "xxx" }.</p>
	<p><b>Scene Refiner</b></p> <p><b>Role:</b>You are the scene refiner agent, responsible for refining the edited scene.</p> <p><b>Task:</b> Call the system's refine scene function.</p>
	<p><b>Scene Renderer</b></p> <p><b>Role:</b> You are the scene renderer agent, responsible for rendering the scene from desired perspectives based on user natural language instructions.</p> <p><b>Input:</b> User rendering commands and original viewpoint.</p> <p><b>Task:</b> Generate an appropriate viewpoint based on the user's natural language instructions, and then call the system's rendering function.</p> <p>Rendering Function: Parameters: { "viewpoint": "xxx" }Return:{"simulation videos" }</p>

Figure 10. The prompts and examples of the LLM agents (zoom-in for the best of views).

end-effector control, position and velocity control, and even wheeled locomotion. For our task, we primarily adjust the RDT chunk size to 4 and fill its actions to the right-arm portion of the unified action vector, aligning with the RDT pre-training datasets. Additionally, we set the control frequency of our data to 10. We use an observation window of two frames, which are fed into the head and right wrist image inputs of RDT. The training batch size is set to 20, while all other settings remain consistent with the official RDT configuration, including a learning rate of  $1e-4$ , the AdamW optimizer, and acceleration via DeepSpeed. We train for 2000 steps.

## A.4. Additional Results

### A.4.1. More Results on Visual Metrics

In Tab. 6, we add reconstruction metric results and, following your advice, include CLIP scores for novel view simulations. Our model consistently outperforms IP2P in visual metrics (see visualizations in Fig. 13).

Method	Reconstruction (PSNR/LPIPS/SSIM)	Simulation (I2I-CLIP)			
		Remove	Insert	Texture	Color
IP2P [5]	–	77.6	82.1	66.7	69.3
<b>RoboPearls</b>	<b>40.6 / 0.08 / 0.96</b>	<b>93.6</b>	<b>97.2</b>	<b>92.5</b>	<b>92.9</b>

Table 6. **Visual metrics.** Avg. of RLBench, Open-X, and Ego4D.

### A.4.2. More Simulation Results

In Fig. 12 (a), we present detailed simulations on the RLBench. Our RoboPearls supports a comprehensive set of simulation operators.

### A.4.3. More Manipulation Tasks Results

In Fig. 12 (b), we show more task demonstrations on the RLBench. RoboPearls successfully performs multiple manipulation tasks.



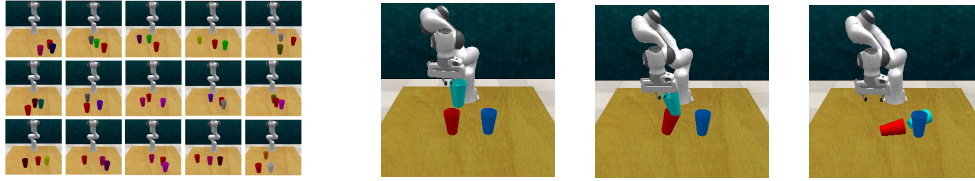
---

**Role:**

You are an expert in the field of robotic learning. A robotic model has failed a task, and you are provided with both the training data and a set of failure cases. Your objective is to analyze the issue, identify the cause of the failure, and suggest corresponding solutions.

**Input:**

1. The training data is summarized in a thumbnail that displays the learned demonstrations.
2. The failure cases are provided in several keyframe images, which visually indicate the cause of the failure.

**Task:**

Your task is to analyze the failure cases and provide a reasonable explanation for the failure, along with suggested improvements. You should focus on factors such as the desktop texture, desktop color, environmental lighting, background color, and the types, quantities, positions, and color combinations of interactive objects. These factors can significantly influence the success of the task.

**Possible Failure Reasons and Suggested Improvements:**

- 1 The placement of interactive objects is not diverse enough, causing the model to overfit and misjudge object positions.  
*Solution:* Add more diverse placement scenarios and increase the variety of training trajectories.
- 2 The colors of the objects in the training data were not varied enough, leading to poor generalization by the model.  
*Solution:* Increase the diversity of interactive object colors.
- 3 The textures of the objects in the training data were not varied enough, leading to poor generalization by the model.  
*Solution:* Increase the diversity of object textures.
- 4 The desktop texture is not varied enough, causing the model to fail in new environments.  
*Solution:* Introduce more diverse desktop textures in the training data.
- 5 The desktop color is not varied enough, causing the model to fail in new environments.  
*Solution:* Add more diverse desktop color scenarios.
- 6 The background patterns are not diverse enough, causing the model to fail in new environments.  
*Solution:* Introduce more varied background patterns in the training data.
- 7 The environmental light colors are not diverse enough, causing the model to fail in new environments.  
*Solution:* Include more diverse lighting scenarios with varying light colors.
- 8 Unintended interference in the workspace leads to failure.  
*Solution:* Add training data that includes interference objects.
- 9 Other failure causes that may be inferred.

**Instructions:**

Choose potential failure causes listed above, and provide an explanation and solution. Your judgment should be as detailed and accurate as possible, with reasoning supported by the images and training data. For example, if the training data contains several colors but the failure case features an object with a color not seen in the training set, explain how this discrepancy contributes to the failure.

**Output Format:**

1. Failure Reason: <chosen reason>
2. Explanation: <explanation of why the failure occurred, based on the images and training data provided>
3. Solution: <suggested improvement for the training data to fix the issue>



**Failure Reason:** The colors of the objects in the training data were not varied enough, leading to poor generalization by the model.

**Explanation:** In the training data provided (first image), the cups are predominantly in shades of red, purple, and green. Notably, there is significant repetition of these colors across the scenarios. In the failure case (second image), the robot is required to interact with cups in red, teal (or pale blue), and blue. The teal and blue cup does not closely resemble any of the colors seen frequently in the training data. Because the training images lack sufficient variation in color—particularly in shades like teal or blue—the model likely struggled to identify or interact with the teal cup due to its unfamiliarity.

**Solution:** To enhance the model's performance and generalization capabilities, it is essential to introduce a broader range of cup colors in the training data. Specifically, including more colors that are underrepresented, such as teal, light blue, orange, and yellow, will help the model better recognize and interact with objects in various color settings. This increased color diversity will enable the robotic system to perform more effectively in real-world scenarios or environments with different object colors.

---

Figure 11. The prompts and examples of the VLM (zoom-in for the best of views).

## A.5. Additional Visualizations

### A.5.1. Spatial-temporal Consistency

Fig. 13 demonstrates that our simulation maintains significant spatial-temporal consistency, whereas the baseline IP2P struggles.

### A.5.2. Identity Encoding Features

In Fig. 14, we adopt PCA to visualize the Identity Encoding features with the rendered semantic and can observe that the approach provides an effective way to select 3D objects in the scene.

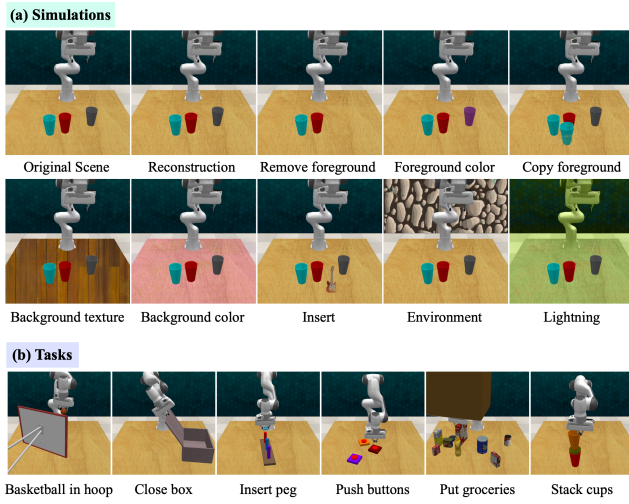


Figure 12. **The detailed simulations (a) and more tasks (b) on RLbench** (zoom-in for the best of views).

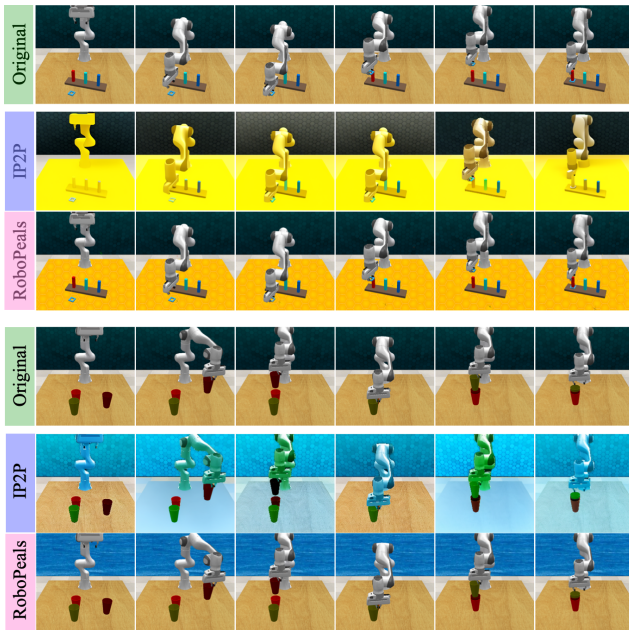


Figure 13. **The visualizations of the spatial-temporal consistency** (zoom-in for the best of views).



Figure 14. **The visualizations of the learned feature vectors.**