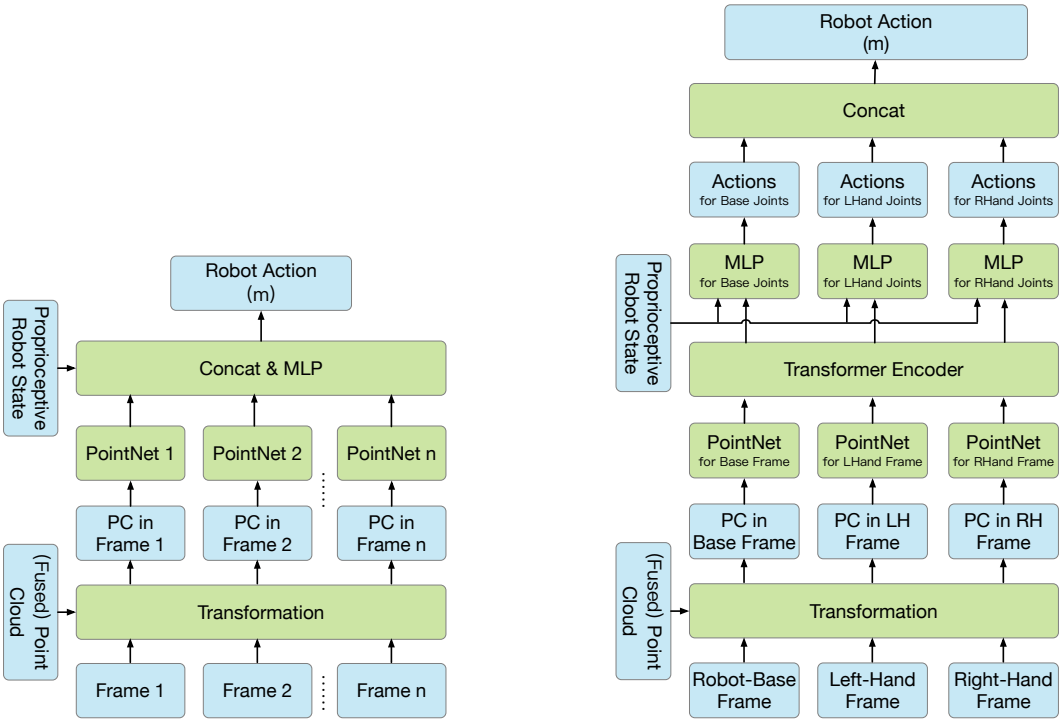
# Frame Mining: a Free Lunch for Learning Robotic Manipulation from 3D Point Clouds Supplementary Material

## S.1 Architecture of the other two FrameMiners

Figure S1 shows architectures of the other two FrameMiners, FrameMiner-FeatureConcat (FM-FC) and FrameMiner-TransformerGroup (FM-TG).




(a) FrameMiner-FeatureConcat (b) FrameMiner-TransformerGroup

Figure S1: Architectures of FrameMiner-FeatureConcat and FrameMiner-TransformerGroup.

## S.2 Additional Experiment Results and Discussions

### S.2.1 Imitation Learning

In the main text, we analyzed the profound impact of coordinate frames on point cloud-based object manipulation learning through online RL algorithms. Apart from online RL, some previous work [1] have shown that dynamic selection of coordinate frames could benefit demonstration-based manipulation learning as well. In this section, we conduct experiments on imitation learning and investigate whether our previous findings can generalize to other algorithm domains.

For each task, we use an expert RL policy to generate 100 successful demonstrations. We then perform Behavior Cloning (BC) by representing input point clouds under different coordinate frames, along with using our proposed FrameMiner-MixAction (FM-MA). We utilize the same network architectures as online RL, and we use MSE loss for training. For FM-MA, the robot-base frame and the end-effector frame(s) are fused. As shown in Table S1, we observe similar findings to Section 3 and Section 4. Specifically, the end-effector frame has much higher performance on single-arm

Submitted to the 6th Conference on Robot Learning (CoRL 2022). Do not distribute.

| | Robot-Base | End-Effector | FM-MA |
|---|---|---|---|
| OpenCabinetDoor | 50±3 | **85±3** | **83±4** |
| OpenCabinetDrawer | 72±4 | **88±2** | **88±2** |
| PushChair | 38±3 | 28±2 | **42±4** |
| MoveBucket | 76±4 | 80±2 | **91±2** |

Table S1: Behavior Cloning (BC) success rates (%) on four ManiSkill tasks. Mean and standard deviation over 5 seeds are shown.

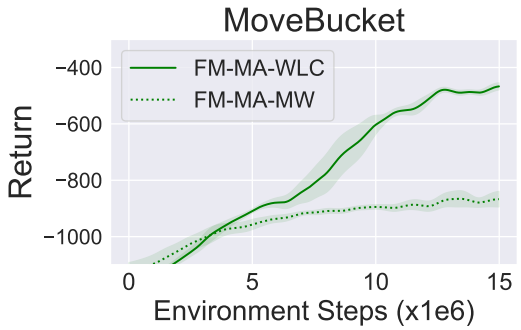



Figure S2: Comparison between FM-MA-WLC and FM-MA-MW on MoveBucket. Mean and standard deviation over 5 seeds are shown. FM-MA-WLC achieves 81±3% final success rate, while FM-MA-MW only has 9±2% final success rate.

tasks (OpenCabinetDoor/Drawer), demonstrating the benefits of end-effector alignment. Our proposed FrameMiner is capable of automatically selecting the best single frame or combining the merits from multiple frames and outperforming single-frame baselines.

### S.2.2 Alternative Designs in FM-MA (Weighted Linear Combination vs. Maximum Weight)

In the main paper, FrameMiner-MixAction (FM-MA) uses weighted linear combination to fuse action proposals from each coordinate frame (see Figure 8). For simplicity, we name this variant FM-MA-WLC. An alternative design is to choose the max-weighted action proposal for each joint (we name this variant FM-MA-MW). Formally, let $A \in \mathbb{R}^{n \times m}$, where $A_{ij}$ denotes the action proposal for the $j$-th robot joint from the $i$-th coordinate frame. Let $W \in \mathbb{R}^{n \times m}$ be the weight matrix predicted by the network. In FM-MA-MW, the output action $\mathbf{a} = (a_1, a_2, \ldots, a_m)$ satisfies $a_j = A_{kj}$ where $k = \text{argmax}_{k=1}^{n} W_{kj}$. Note that FM-MA-WLC uses SoftMax to normalize the weights; thus FM-MA-WLC can be regarded as a "soft version" of FM-MA-MW.

To compare the two designs, we conduct two experiments: (1) We train FM-MA-MW from scratch. Results are shown in Figure S2. (2) We resume from the final checkpoint of the original FM-MA-WLC. During evaluation, we use the max-weighted action proposal as the action output. Results are shown in Table S2. We observe that for both experiments, using FM-MA-MW deteriorates performance. We conjecture that FM-MA-WLC alleviates optimization difficulty, which likely comes from the fact that it is a "soft version" of FM-MA-MW with well-behaving gradients. On the other hand, since FM-MA-MW uses argmax operation over columns of $W$, there is a lack of gradient for $W$ during training, which leads to more difficult optimization.

### S.2.3 Ablation Study on Camera Placements

As a recap, the five tasks analyzed in our main paper cover both static and moving camera settings. The experiments in the main paper were conducted using default camera placements shown in Figure 2. For the four tasks with moving cameras, a panoramic camera is mounted on the robot head.

While FrameMiners do not require changing existing camera placements, camera placements could still matter, since different camera placements affect the point clouds being captured (due to different occlusion and sparsity patterns). Therefore, we perform an experiment where we move the

| | FM-MA (WLC eval) | FM-MA (MW eval) |
|---|---|---|
| OpenCabinetDoor | 84±2 | 45±5 |
| OpenCabinetDrawer | 93±1 | 93±2 |
| PushChair | 36±4 | 20±3 |
| MoveBucket | 81±3 | 14±3 |

Table S2: Success rate (%) comparison between the same FM-MA checkpoint evaluated using weighted linear combination of actions (WLC) and using maximum-weighted action (MW) on four ManiSkill tasks. Mean and standard deviation over 5 seeds are shown.

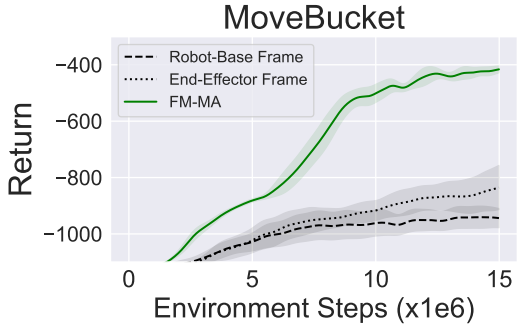



Figure S3: Results on MoveBucket with a panoramic camera mounted on the robot base. The "Robot-Base Frame" and the "End-Effector Frame" indicate the coordinate frames used to represent captured input point clouds. FM-MA fuses the two end-effector frames (left and right arms) and the robot-base frame. Mean and standard deviation over 5 seeds are shown.

panoramic camera from the robot head to the robot base. As shown in Figure S3, we observe similar phenomena as in Figure 10. Specifically, fusing multiple coordinate frames with our FrameMiners still leads to better sample efficiency and final performance, demonstrating that FrameMiners are robust under different camera placements.

#### S.2.4 Learning Adaptive Frame Transformations from Observations

In our paper, we use known transformations (e.g., end-effector pose in robot state) to align input point clouds in different coordinate frames and propose FrameMiners to fuse merits of multiple coordinate frames. A potential baseline is to learn a transformation adaptively based on input point clouds. To examine the effectiveness of this baseline, we add an additional network before the PointNet backbone to learn an adaptive $\mathbb{SE}(3)$ transformation based on the input point cloud. This transformation is then applied to the input point cloud before passing it through the PointNet backbone (note that we remove spatial transformation layers from the original PointNet in all of our experiments). However, as shown in Figure S4, adding this $\mathbb{SE}(3)$ transformation layer barely improves performance.

We conjecture that it's very difficult to predict a $\mathbb{SE}(3)$ transformation for aligning the input point cloud across time due to the large search space (where most transformations are ineffective) and weak supervision from RL training loss. Moreover, in many challenging tasks, we may need to fuse information simultaneously from multiple coordinate frames (e.g., left-hand and right-hand frames). This is not achievable through learning a single transformation. In contrast, for FrameMiners, we take advantage of easily-accessible frame information (e.g. end-effector poses) without relying on transformation prediction. We then fuse the merits of multiple candidate coordinate frames.

#### S.2.5 $\mathbb{SO}(3)$ and $\mathbb{SE}(3)$ Equivariant Point Cloud Backbones

Recently, there have been several works on designing $\mathbb{SO}(3)$ and $\mathbb{SE}(3)$ equivariant/invariant backbone networks for point cloud learning [2, 3]. While they are of great benefit for analysis within each object (e.g., shape classification, part segmentation, and 6D pose estimation), our robot-object interaction setting is a bit different.

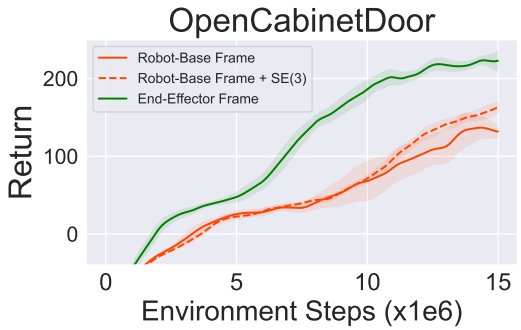



Figure S4: Ablation study on adding an adaptive $\mathbb{SE}(3)$ transformation prediction layer. When the input point cloud is represented in the robot-base frame, adding such transformation layer barely improves performance, while representing the point cloud in the end-effector frame significantly improves performance.

In robot manipulation scenarios, a particular challenge comes from inferring the relations between two object parts (e.g., relative pose between the end-effector and the cabinet handle). This binary relation inference task is challenging under the weak RL loss supervision, even using $\mathbb{SO}(3)$ and $\mathbb{SE}(3)$ equivariant/invariant backbones. FrameMiners explicitly approach this challenge by aligning point clouds (across multiple time steps) with the known transformation matrices (e.g., the end-effector pose). This reduces many binary relation inference tasks to single-subject location tasks, which has much lower difficulty. For example, when using the end-effector frame in the OpenCabinet task, the network only needs to copy the handle pose to infer the relative pose between the handle and the end-effector, as the end-effector is always at the frame origin.

### S.3 More Details of Manipulation Tasks

**Task Descriptions:**

- In OpenCabinetDoor, a single-arm mobile agent needs to approach a cabinet, use the handle to fully open the designated cabinet door, and then keep the door static for a while.
- In OpenCabinetDrawer, a single-arm mobile agent needs to approach a cabinet, use the handle to fully open the designated cabinet drawer, and then keep the drawer static for a while.
- In PushChair, a dual-arm mobile agent needs to approach the chair, push the chair to a target location, and then keep the chair static for a while.
- In MoveBucket, a dual-arm mobile agent needs to approach the bucket, move the bucket to a target platform, place the bucket onto the platform, and then keep the bucket static for a while.
- In PickObject, a single-arm fixed-base agent needs to grasp an object from the table, lift it up to a certain target height, and keep it static for a while.

Simulations are fully physical. For OpenCabinetDoor, OpenCabinetDrawer, PushChair, and MoveBucket, there are 66, 49, 26, and 29 different objects (designated parts) during training, respectively.

**Observations and Actions:**

For all ManiSkill tasks, the proprioceptive robot state includes:

- Positions of all (two if single-arm, four if dual-arm) fingers
- Velocities of all (two or four) fingers
- x, y position of the mobile robot base
- Mobile robot base's rotation around the z-axis
- x, y velocity of the mobile robot base
- Angular velocity of the mobile robot base around the z-axis
- Joint angles of the robot, excluding the joints in the mobile base
- Joint velocities of the robot, excluding the joints in the mobile base

- Indicator of whether each joint receives an external torque

The action space includes:

- x, y velocity of the mobile robot base
- Angular velocity of the mobile robot base around the z-axis
- Height of the robot body
- Joint velocities of the robot, excluding joints of the mobile base and the gripper fingers
- Joint positions of the gripper fingers

Joint positions of the gripper fingers are controlled by position PID. All other action components are controlled by velocity PID.

For the PickObject task, the proprioceptive robot state includes:

- Joint angles of the robot,
- Joint velocities of the robot,
- 1D gripper joint position,
- Target xyz positions of object.

The action space includes 3 DoF end-effector position and 1 DoF gripper joint position.

For all tasks, input point cloud features include xyz coordinates, RGB colors, and one-hot segmentation masks for each part category.

**Motivations for Our Task Choice**

We aim to cover a wide range of factors that may influence the selection of point cloud coordinate frames. Specifically, the tasks are chosen to cover various robot mobilities, numbers of robot arms, and camera settings, as demonstrated in Figure 1.

Different robot mobility results in differences in world frame and robot base frame. These two frames are aligned in static robots but not in mobile robots. The robot's mobility can also change the focus of tasks (e.g., navigation or object interaction), which may place different requirements on the choice of point cloud frame.

We cover both single-arm and dual-arm environments, as they pose different requirements for point cloud frame selection. In single-arm environments, using the only end-effector frame may already be able to achieve good performance. However, in dual-arm environments, there are two end-effector frames, and these tasks require precise coordination between the two robot arms, which pose significant challenges for manipulation learning. As each end-effector may have a preferred frame, the necessity of frame fusion becomes more pronounced.

Last but not least, camera placements determine sources of point clouds, which may potentially influence the selection of coordinate frames. In our experiments, we cover both static camera settings and moving camera settings (mounted on robots).

### S.4 Detailed Experimental Settings and Hyperparameters

For our visual backbones, our PointNets are implemented with a three-layer MLP with dimensions $[64, 128, 300]$ followed by a max-pooling layer. We do not apply any spatial transformation to the inputs. Our SparseConvNets are implemented as a SparseResNet10 using TorchSparse [4]. SparseResNet10 has a 4-stage pipeline with kernel size 3 and hidden channels $[64, 128, 256, 512]$ respectively. We use kernel size 3 and stride 2 for downsampling. Initial voxel size is 0.05. Final features in the final-stage voxels are maxpooled as output visual feature.

All of our agents are trained with PPO (hyperparameters in Table S3). Each policy MLP that outputs actions has dimensions $[192, 128, \text{action\_dim}]$. For FM-MA that uses input-dependent joint-specific weights to fuse action proposals from different frames, the MLP has dimension $[192, n \times m]$, where $n$ is the number of frames and $m$ is the dimension of action space. For FM-TG that uses Transformer to fuse features from different frames, the Transformer has 3 layers with hidden dimension 300 and feed-forward dimension 1024. For all network variants, the value head takes the concatenation of all

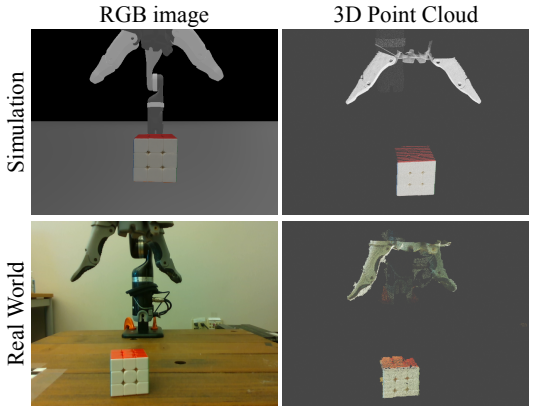



Figure S5: RGB images and 3D point clouds captured in both simulation and the real world. Colored point clouds for better illustration.

| Hyperparameters | Value |
|---|---|
| Optimizer | Adam |
| Discount ($\gamma$) | 0.95 |
| $\lambda$ in GAE | 0.95 |
| PPO clip range | 0.2 |
| Coefficient of the entropy loss term of PPO $c_{ent}$ | 0.0 |
| Advantage normalization | True |
| Reward normalization | True |
| Number of threads for collecting samples | 5 |
| Number of samples per PPO update | 40000 |
| Number of epochs per PPO update | 2 |
| Number of samples per minibatch | 330 |
| Gradient norm clipping | 0.5 |
| Max KL | 0.2 |
| Policy learning rate | 3e-4 (non FM-TG); 1e-4 (FM-TG) |
| Value learning rate | 3e-4 |
| Action MLP Last Layer Initialization | Zero-init |

Table S3: Hyperparameters for PPO.

visual features from all frames as input and passes through an MLP with dimensions $[192, 128, 1]$ to output value prediction.

In addition, we found that zero-initializing the last layer of MLP before action output along with the joint-specific weights in FM-MA to be very helpful for stabilizing agent training.

For each task, we train an agent for a fixed number of environment steps. Specifically, for OpenCabinetDoor, OpenCabinetDrawer, and MoveBucket, we train for 15 million steps. For PushChair, we train for 20 million steps. For PickObject, we train for 4 million steps. Success rates are calculated among 300 evaluation trajectories.

### S.5 More Details of Real-World Experiments

Figure S5 shows the captured RGB images and point clouds in both simulation and the real world (by RealSense camera). For both simulation and the real-world environment, the ground points are removed using z-coordinate threshold or RANSAC, and the distant points are clipped. To reduce the sim-to-real gap, we only use xyz coordinates as our input point cloud feature, and we discard RGB colors.

## References

[1] B. Wen, W. Lian, K. Bekris, and S. Schaal. You only demonstrate once: Category-level manipulation from single visual demonstration. In *Proceedings of Robotics: Science and Systems*, 2022.

[2] C. Deng, O. Litany, Y. Duan, A. Poulenard, A. Tagliasacchi, and L. J. Guibas. Vector neurons: A general framework for so (3)-equivariant networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12200–12209, 2021.

[3] A. Simeonov, Y. Du, A. Tagliasacchi, J. B. Tenenbaum, A. Rodriguez, P. Agrawal, and V. Sitzmann. Neural descriptor fields: Se(3)-equivariant object representations for manipulation. 2022.

[4] H. Tang, Z. Liu, S. Zhao, Y. Lin, J. Lin, H. Wang, and S. Han. Searching efficient 3d architectures with sparse point-voxel convolution. In *European Conference on Computer Vision*, 2020.