6 Appendix

6.1 Noise schedulers for Local Trajectory Diffuser

We model local trajectory optimization as a discrete-time diffusion process, which we implement using the DDPM sampler [20]. DDPM uses a non-parametric time-dependent noise variance scheduler β_k, which defines how much noise is added at each time step. We adopt the cosine schedule of GLIDE [79] as our choice for β_k:

$$\beta_k = \frac{1 - \cos\left(\frac{k+1.008}{1.008} * \frac{\pi}{2}\right)^2}{\cos\left(\frac{k+0.008}{1.008} * \frac{\pi}{2}\right)^2} \tag{6}$$

By defining $\alpha_k = 1 - \beta_k$, and $\bar{\alpha}_k = \prod_{i=1}^{k} \alpha_i$, we can now obtain the analytical form of $\lambda_k, \gamma_k, \sigma_k$ in Equation 1 as follows:

$$\lambda_k = \frac{1}{\sqrt{\alpha_k}} \tag{7}$$

$$\gamma_k = \frac{1 - \alpha_k}{\sqrt{1 - \bar{\alpha}_k}} \tag{8}$$

$$\sigma_k = \frac{1 - \bar{\alpha}_{k+1}}{1 - \bar{\alpha}_k} \beta_k \tag{9}$$

where k is the diffusion denoising timestep.

6.2 Act3D Background and Implementation Details

Act3D is a language-conditioned end-effector 6-DoF keypose predictor that learns 3D perceptual representations of arbitrary spatial resolution via recurrent coarse-to-fine 3D point sampling and featurization. Act3D featurizes multi-view RGB images with a pre-trained 2D backbone and lifts them in 3D using depth to obtain a multi-scale 3D scene feature cloud. It then iteratively predicts 3D foci of attention in the empty 3D workspace, samples 3D point grids in their vicinity, and featurizes the sampled 3D points using relative cross-attention to the physical scene feature cloud, language tokens, and proprioception. Act3D detects the 3D point that corresponds to the next best end-effector position using a detection Transformer head, and regresses the rotation, end-effector opening, and planner collision avoidance from the decoder's parametric query.

We extract two feature maps per 256×256 input image view: 32×32 coarse visual tokens and 64×64 fine visual tokens. We use three ghost point sampling stages: first across the entire workspace (roughly 1 meter cube), then in a 16 centimeter diameter ball, and finally in a 4 centimeter diameter ball. The coarsest ghost points attend to a global coarse scene feature cloud ($32 \times 32 \times n_{\text{cam}}$ coarse visual tokens) whereas finer ghost points attend to a local fine scene feature cloud (the closest $32 \times 32 \times n_{\text{cam}}$ out of the total $64 \times 64 \times m_{\text{cam}}$ fine visual tokens). During training, we sample 1000 ghost points in total split equally across the three stages. At inference time, we trade-off extra performance for additional compute by sampling more ghost points than the model ever saw at training time $(20,000)$. We use 2 layers of cross-attention and an embedding size 60 for single-task experiments and 120 for multi-task experiments. Training samples are augmented with random crops of RGB-D images and ± 45.0 yaw rotation perturbations (only in the real world as this degrades performance in simulation).

6.3 Simulation Setup in RLBench

The RLBench simulation environment uses a Franka Panda robotic arm on a table-top setting. We consider $m = 4$ camera inputs: *left_shoulder*, *right_shoulder*, *wrist*, and *front*, as shown in Figure 3. The *wrist* camera is attached to the robot's end-effector and moves together with the robot. The other 3 are static. To ensure a fair comparison, when comparing with PerAct, we use all 4 cameras following PerAct setting, and use the first 3 cameras when compared with other baselines.

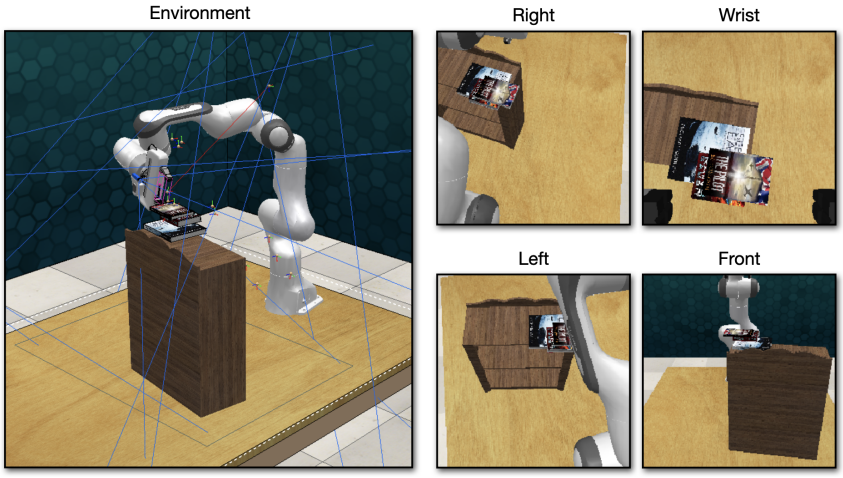

Figure 3: Simulation setup.

6.4 Real-world Setup

Our real-robot setup contains a real Franka Panda robotic arm equipped with a parallel jaw gripper, as shown in Figure 4. We use a single Azure Kinect sensor to provide RGB-D input signal from the front view at 30Hz. The image input is of resolution 1280×720, and we crop and downsample it to 256×256 before feeding it to our model. We calibrate the extrinsics of the camera with respect to the robot base using the `easy_handeye`[1] ROS package. Our full model generates dense trajectories, thus we do not use low-level motion planners. We collect 6-DoF human demonstrations by tele-operating the robot using a SpaceMouse[2] at 30Hz, following [16]. We use the same strategy for keyframe extraction as in simulation. Our real-world multi-task policy is trained on 4 A100 GPUs for 3 days. Inference is done on a desktop with a single RTX4090 GPU, running Ubuntu 20.04 and ROS Noetic. For robot control, we use the open-source `frankapy`[3] package to send real-time position-control commands to the robot.

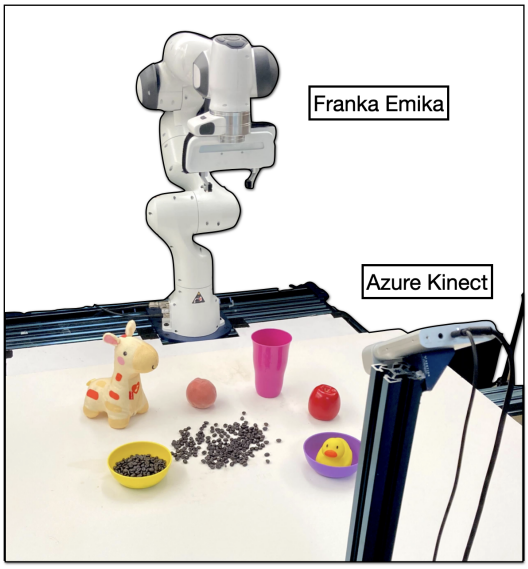

Figure 4: Real-world setup.

[1] https://github.com/IFL-CAMP/easy_handeye
[2] https://3dconnexion.com/us/product/spacemouse-compact/
[3] https://github.com/iamlab-cmu/frankapy