

AdaDexTrack: Dynamic Modulation for Adaptive and Generalizable Dexterous Manipulation Tracking

Supplementary Material

Contents

A Additional Details on Method	1
A.1. Data Preprocessing	1
A.2. Sim-to-Real Transfer	1
A.3. Tracking Controller Training	2
A.4. Tracking Modulator Training	3
B Simulation Experiments Details	3
B.1. Environment Setup	3
B.2. Baseline Implementation	3
B.3. Qualitative Results	4
C Real-World Experiments Details	5
C.1. Environment Setup	5
C.2. Metrics	5
C.3. Textured Axially Symmetric Objects	5
C.4. Real-World Evaluation Without Object Tape	5
C.5. Real-World Evaluation with Practical Object Geometry	5
C.6. Qualitative Results	6
D Additional Analysis	6
D.1. Object Latent Smoothness	6

A. Additional Details on Method

A.1. Data Preprocessing

Language-Guided Trajectory Augmentation. We build our language-guided HOI trajectory dataset on top of the text-annotated sequences provided by DiffH2O [9]. Starting from the full set of demonstrations, we retain only sequences that (i) involve single-hand (predominantly right-hand) manipulation without bimanual interaction, (ii) exhibit no hand–robot-arm collisions, and (iii) keep the hand pose within the manipulator’s reachable workspace at every timestep, which yields 505 reference trajectories. For each retained trajectory, we then perform paraphrase-based language augmentation. Concretely, we use GPT-5 [29] to generate nine semantically equivalent descriptions that follow DiffH2O’s native annotation format and preserve the original task intent. We feed these paraphrased descriptions back into the DiffH2O text-to-motion model to synthesize additional HOI trajectories conditioned on the same object geometry. Fig. 7 shows a qualitative example of our augmentation results, where a single original trajectory is expanded into ten variants. Applying the same feasibility filters to

the augmented set results in 2,765 valid text–trajectory pairs spanning 50 objects.

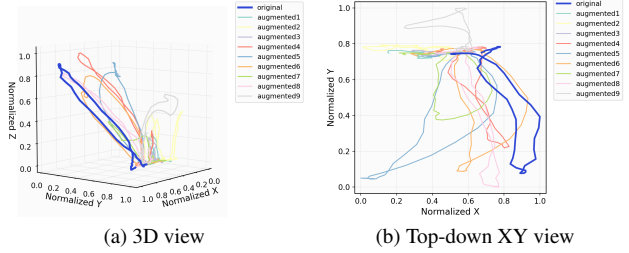


Figure 7. Language-guided HOI reference augmentation in 3D and top-down XY views.

Kinematic Retargeting. Our kinematic retargeting procedure is similar to DexTrack [21]. Given a human hand–object interaction trajectory with MANO hand poses $\{\mathbf{H}_n^{\text{human}}\}$ and object poses $\{\mathbf{O}_n\}$, we retarget it to the robot hand by matching keypoints between the MANO hand and the articulated robot hand. Let $\mathbf{K}_n^{\text{human}}$ be the MANO keypoints at timestep n and $\mathbf{K}_n(\theta_n)$ the corresponding robot hand keypoints obtained via forward kinematics from joint angles θ_n . At each timestep, we solve a least-squares problem $\min_{\theta_n} \|\mathbf{K}_n(\theta_n) - \mathbf{K}_n^{\text{human}}\|$, yielding a sequence of robot hand joint poses $\{\theta_n\}$ that serves as the kinematic reference for our controller.

Object Latent Encoding. We encode object geometry with a latent code extracted by a Chamfer-trained point-cloud autoencoder. Given an object point cloud P , the encoder produces the object latent $f_o = E_{pc}(P)$, which is used as the object representation in our tracker and modulated online during execution.

A.2. Sim-to-Real Transfer

System Identification. We perform system identification on the robot arm and hand by driving each joint with a family of sinusoidal position commands and recording the resulting joint trajectories. Specifically, we apply commands of the form $A(t) = a \sin(2\pi ft)$, where both the frequency f and the amplitude a are swept over a range, and a is chosen such that the commanded motion stays within the joint limits. We then replay the same command sequences in simulation and identify, for each joint independently, the stiffness and damping parameters by minimizing the discrepancy between the simulated and real joint trajectories. As shown in Fig. 8, the identified model produces response curves in simulation that closely match those of the real robot under identical control commands.

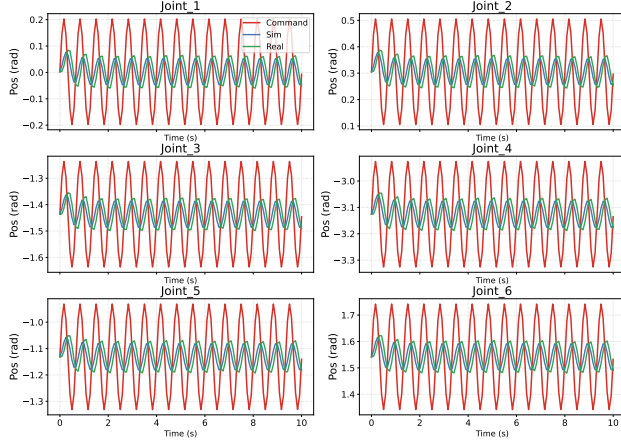


Figure 8. System identification of the arm joints. For each joint, we plot three traces: (1) the commanded input, (2) the simulated response using the identified dynamics model, and (3) the measured real response. The close agreement between (2) and (3) indicates that the identified parameters faithfully capture the joint dynamics.

Domain Randomization. To improve robustness and facilitate sim-to-real transfer, we apply domain randomization to both the system dynamics and the observations/actions during training. On the dynamics side, we randomly perturb the arm and hand parameters, including link masses, joint friction, joint stiffness, and joint damping, around their nominal values. For the manipulated objects, we use a larger range of randomization over physical parameters such as mass, friction coefficients, and restitution, and also randomize the initial object pose relative to the reference trajectory. On the sensing and control side, we inject Gaussian noise into the proprioceptive states and add small Gaussian noise to the action commands. The exact ranges used for domain randomization are summarized in Table 3.

A.3. Tracking Controller Training

A.3.1. Specialist Policy

Action Space. The control action at each timestep is defined in joint space. We use a 22-DoF arm-hand system with joint configuration $\mathbf{q} \in \mathbb{R}^{22}$. The action space $\mathcal{A} \subset \mathbb{R}^{22}$ consists of relative joint commands $\Delta\mathbf{q} \in \mathbb{R}^{22}$. Internally, we maintain a vector of joint position targets \mathbf{q}^{tar} . At every control step, the policy outputs $\Delta\mathbf{q}$ and we update the targets by $\mathbf{q}^{\text{tar}} \leftarrow \mathbf{q}^{\text{tar}} + \Delta\mathbf{q}$ (followed by clipping to the joint limits). The updated targets \mathbf{q}^{tar} are then sent to the low-level joint position controllers as setpoints.

Observation Space. Table 4 details the observation space used for training the specialist policies.

Reward implementation details. In practice, we implement the reward in Eq. (1) as a shaped combination of tracking and proximity terms. The joint and hand-pose terms are

Parameter	Type	Distribution	Initial Range
Robot			
Mass	Scaling	uniform	[0.9, 1.1]
Friction	Scaling	uniform	[0.9, 1.1]
Joint stiffness	Scaling	uniform	[0.9, 1.1]
Joint damping	Scaling	uniform	[0.9, 1.1]
Object			
Mass	Scaling	uniform	[0.7, 1.3]
Friction	Scaling	uniform	[0.7, 1.3]
Restitution	additive	uniform	[0.0, 0.4]
Scale	Scaling	uniform	[0.95, 1.05]
Initial Position (x,y)	Additive	Gaussian	[0.0, 0.05]
Initial Rotation (yaw)	Additive	Gaussian	[0.0, 0.5]
Observation			
Proprioception	Additive	Gaussian	[0.0, 0.002]
Action			
Correlated noise	Additive	Gaussian	[0.0, 0.02]
Uncorrelated noise	Additive	Gaussian	[0.0, 0.05]

Table 3. Domain randomization configuration.

realized by a tracking loss $\ell_{\text{track}}(t)$ that aggregates palm position and orientation errors, 22-DoF joint-space errors, and fingertip position errors with respect to their references; this loss enters the reward with a negative weight so that larger deviations from the reference are penalized more strongly. The hand-object proximity term is instantiated using the distances between the palm and the object, and between the fingertips and the object, which encourages the hand to move into and remain in the vicinity of the object. To avoid excessively large gradients, these distances are clipped to fixed upper bounds, and the fingertip penalty is activated only once the palm is sufficiently close to the object.

The object-pose term in Eq. (1) is implemented via a goal distance $d_{\text{goal}}(t)$ that combines translational and rotational discrepancies between the current and target object poses. We represent rotations as unit quaternions and compute the angular component of $d_{\text{goal}}(t)$ from the quaternion difference via the dot product between the normalized goal quaternion and the current quaternion. In addition, we include a sparse success bonus $r_{\text{bonus}}(t) = \mathbb{I}[d_{\text{palm}}(t) \leq \tau_{\text{palm}}, d_{\text{fing}}(t) \leq \tau_{\text{fing}}, d_{\text{goal}}(t) \leq \tau_{\text{goal}}] (1 + 10 d_{\text{goal}}(t))^{-1}$, where $\mathbb{I}[\cdot]$ is the indicator function. This bonus rewards states where the hand remains close to the object and the object pose is very close to the goal, and can be viewed as a shaped realization of the proximity component in Eq. (1).

To improve computational efficiency and obtain a smooth orientation error, we represent all orientations using unit quaternions and compute $d_R(\cdot)$ from the quaternion difference via the dot product between the normalized goal quaternion and the current quaternion.

Semantic Grouping of Specialists. We start from the text-annotated DiffH2O demonstrations and first apply a feasi-

bility filter (single-hand interaction, no collisions, reachable workspace), which leaves 505 valid human–object interaction sequences. On top of these 505 base demonstrations, we then perform paraphrase-based augmentation: for each retained sequence, we use GPT-5 to generate multiple semantically equivalent but textually different captions and feed them into DiffH2O to synthesize additional reference trajectories. This procedure yields at most ten references per base demonstration (one from the original caption and up to nine from paraphrased captions), i.e., 5050 candidates in total. We apply the same feasibility filter again and obtain 2,765 valid reference trajectories overall.

For specialist training, we group references by their underlying semantic intent: all trajectories that originate from the same base demonstration (and its paraphrased captions) are treated as one semantic group. Each such group defines a single specialist tracker π_i^{track} , which is trained jointly on all of its associated reference trajectories by running them in parallel in Isaac Gym. In this way, we obtain one specialist per semantic group, with each specialist responsible for a small cluster of semantically equivalent references rather than a single trajectory. This semantic grouping substantially reduces the number of specialist policies while still exposing each of them to multiple motion and language variants of the same underlying task.

Time consumption. Each specialist is trained in Isaac Gym [27] with 8,192 parallel environments on a single NVIDIA A10 GPU and converges in about 3 hours.

A.3.2. Generalist Policy

The observation space used for behavior cloning of the generalist policy is identical to that of the specialist policy, as summarized in Table 4.

Time consumption. The generalist policy is trained on eight NVIDIA A10 GPUs and converges in about two days.

A.4. Tracking Modulator Training.

During modulator training, we freeze the generalist policy and optimize only the modulator. The modulator uses the same observation space as in Table 4, and its reward is identical to those used in specialist training. As discussed in Sec.3.2, the modulator modifies both the inputs and outputs of the tracker: it modulates the tracking reference and the object latent in the observation, and applies a small residual modulation to the tracker’s action.

Time consumption. The modulator policy is trained in Isaac Gym [27] with 8,192 parallel environments on a single NVIDIA A10 GPU and converges in about two days.

B. Simulation Experiments Details

B.1. Environment Setup

We use the Isaac Gym simulator with a simulation frequency of 60 Hz and a control frequency of 12 Hz. For each

Index	Description
0 – 22	arm-hand joint positions
22 – 57	the palm pose and the poses of the four fingertips
57 – 64	object pose
64 – 71	distance between the current object pose and the object reference pose at the current timestep
71 – 99	object reference poses at future timesteps $\{1, 2, 4, 12\}$ relative to the current timestep
99 – 121	distance between the current and reference arm-hand joint positions at the current timestep
121 – 209	reference arm-hand joint positions at future timesteps $\{1, 2, 4, 12\}$ relative to the current timestep
209 – 273	object latent

Table 4. Observation space used for specialist policy training.

Index	Description
0 – 22	arm-hand joint positions
22 – 57	the palm pose and the poses of the four fingertips
57 – 64	object pose
64 – 71	distance between the current object pose and the object reference pose at the current timestep
71 – 141	object reference poses at future ten timesteps relative to the current timestep
141 – 147	distance between the current and reference arm joint positions at the current timestep
147 – 207	reference arm joint positions at future ten timesteps relative to the current timestep
207 – 271	object latent

Table 5. Observation space for the low-level controller in ObjDex.

reference trajectory to be tracked, we define the tracking start frame as the first frame where the minimum distance between the hand mesh and the object center exceeds 20 cm. Starting from this frame, we take the subsequent 300 frames as the full trajectory segment used for training.

B.2. Baseline Implementation

B.2.1. DexTrack

We implement a single, generalist neural tracker that takes the current robot/object state and a fixed reference as input and directly outputs actions—without any in-loop modula-

tion (no reference, object latent, or positional target modulation at test time). To ensure a fair comparison, we keep the observation/action spaces, reference format, rewards, training data, and evaluation protocol identical to those used for AdaDexTrack. The object latent is computed once at episode start and held fixed throughout execution, matching the fixed-reference, fixed-representation setting.

Training objective (IL+RL). The combined IL+RL training procedure and the homotopy-based demonstration mining described in DexTrack are not publicly available. We therefore re-implement the IL+RL scheme (without homotopy mining): we train the policy with PPO on the task reward and add a time-indexed imitation regularizer that directly matches the policy’s action at timestep t to the logged teacher action $\mathbf{a}_t^{\text{teacher}}$ for that same reference index. In other words, since tracking uses a time-indexed reference, the imitation target is keyed by t (no observation-based matching). The total objective is the standard PPO actor–critic loss plus a weighted imitation term.

We reproduce a hierarchical planner-over-controller baseline: a high-level, object-centric planner synthesizes wrist trajectories conditioned on the task goal, and a low-level RL controller tracks these trajectories with the dexterous hand. The two layers are trained with different objectives (generative trajectory modeling vs. RL) and communicate primarily via wrist-space references.

Low-level controller. We implement an RL tracker with the same action space as AdaDexTrack for fairness, but with a different observation design. Following ObjDex, the policy receives a 10-step lookahead reference containing only the object pose and arm joint positions for the next

Data augmentation loop. Following ObjDex, once the low-level controller is trained, we execute it in simulation under the planner’s guidance, collect successful motion trajectories, add them to the planner’s training set, and fine-tune the high-level planner on the augmented data.

We train a single PPO policy across all training references, using the same action space, observation space, and reward as in the specialist training (Sec. 3.1).

We provide qualitative comparisons with the baselines on the unseen trajectory set in Figure 13. Across tasks, AdaDexTrack achieves the most faithful hand-object tracking, while Vanilla RL (PPO) and DexTrack often drift or fail. ObjDex typically follows the object trajectory but, lacking hand-shape tracking, produces unnatural, non-humanlike hand motions.

Figure 9. Real-world setup. (a) Hardware. (b) Test objects. The blue box marks objects used for the real-world unseen-trajectory set; the orange box marks objects used for the real-world unseen-object set.

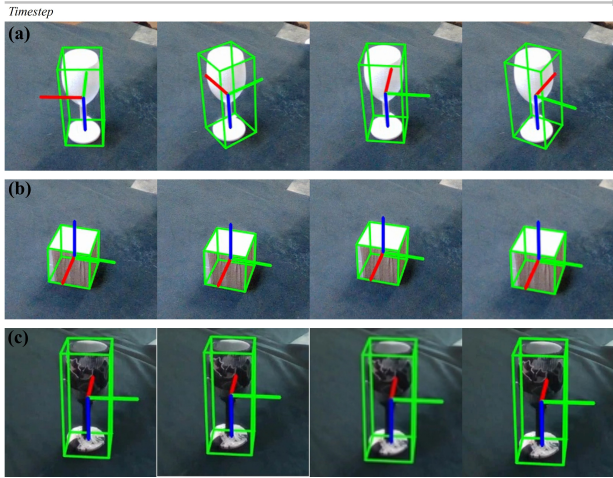


Figure 10. FoundationPose estimation behavior. (a) Wineglass: although the object is static, the estimate continuously spins around the blue symmetry axis due to axial symmetry. (b) Cube: despite discrete symmetry, the pose remains stable (no continuous spin). (c) Textured wineglass: asymmetric texture breaks the symmetry and yields a stable object frame.

C. Real-World Experiments Details

C.1. Environment Setup

Our real-world setup is shown in Fig. 11. We use an Intel RealSense D435 for RGB-D sensing and FoundationPose [45] for 6-DoF object pose estimation. In the main real-world evaluation, we exclude axially symmetric objects (e.g., wineglass) because FoundationPose exhibits continuous rotational ambiguity on them (Fig. 10a), which would confound the manipulation results. By contrast, objects with discrete symmetries (e.g., cube with 90° rotational invariance; Fig. 10b) do not induce continuous spin; edges or mild texture typically yield stable poses up to the object’s symmetry group. Accordingly, we include discretely symmetric objects when the estimator is stable under this equivalence and exclude objects that exhibit free spinning about a symmetry axis. We further study axially symmetric objects with added asymmetric texture in Sec. C.3, where the texture breaks the visual symmetry and enables a stable object frame (Fig. 10c).

C.2. Metrics

In real-world experiments, we report Success Rate as the completion metric defined in Sec. 4.1, identical to the simulation setting. All thresholds match Sec. 4.1, and we report the average completion over trajectories.

C.3. Textured Axially Symmetric Objects

To study axially symmetric objects, we add asymmetric texture to break the rotational symmetry and obtain stable FoundationPose estimates (Fig. 10c). Fig. 11 shows the 8

textured axially symmetric objects used in this study. Since FoundationPose also requires a known object mesh, we obtain the object meshes using AR Code [1]. We evaluate these 8 objects in the real world, with 5 unseen trajectories per object. Table 6 reports the average completion rate over all trials. With stable pose estimates, our full method substantially outperforms the variant without the modulator, showing that the modulator remains effective on rotationally symmetric geometries.



Figure 11. Eight textured axially symmetric objects used in our real-world experiments.

Method	Completion (% , \uparrow)
Ours (w/o Modulator)	29.72%
Ours	58.61%

Table 6. Real-world results on textured axially symmetric objects.

C.4. Real-World Evaluation Without Object Tape

To address the concern that tape on the objects may artificially simplify grasping, we conduct an additional real-world evaluation after removing all tape from the objects. We keep the rest of the setup unchanged and re-run both the unseen-trajectory and unseen-object settings. Table 7 reports the resulting completion rates.

Removing the tape does not degrade performance; instead, it leads to higher success rates for both methods. We attribute this to the fact that excessive friction can hinder the small relative motions needed for grasp adjustment, whereas untaped objects allow more natural micro-sliding during contact. Our full method still substantially outperforms the variant without the modulator, indicating that the improvement does not rely on artificial adhesion, but on more effective handling of natural contact dynamics.

Method	Unseen Traj.	Unseen Obj.
Ours (w/o Modulator)	36.42%	18.75%
Ours	65.71%	32.50%

Table 7. Real-world results without grip tape on the objects.

C.5. Real-World Evaluation with Practical Object Geometry

To further assess the practicality of our method, we replace the ground-truth object geometry used in the main

experiments with inputs obtainable from lightweight perception tools. Specifically, we use InstantMesh [48] to reconstruct object geometry from a single RGB image, and use the resulting geometry for both policy inference and FoundationPose-based pose tracking. We evaluate this setting on the unseen-trajectory and unseen-object sets from Sec. 4.4, without object tape. As shown in Table 8, using reconstructed geometry yields performance comparable to ground-truth geometry, indicating that our method does not rely on carefully prepared CAD models.

We further consider a stricter setting where the object input is a segmented partial point cloud from a single-view depth observation. This setting is more challenging due to incomplete geometry and self-occlusion. Nevertheless, our method remains robust, achieving strong performance on both evaluation sets. These results support the practical applicability of our approach under realistic geometry acquisition pipelines.

Input	Method	Unseen Traj.	Unseen Obj.
GT Pointcloud	Ours	65.71%	32.50%
Recon. Pointcloud	Ours	59.28%	34.73%
Partial Pointcloud	Ours	49.46%	25.72%

Table 8. Real-world results under practical object-geometry inputs. GT Pointcloud uses ground-truth object geometry; Recon. Pointcloud is reconstructed from a single RGB image using InstantMesh [48]; Partial Pointcloud is obtained from segmented single-view depth.

C.6. Qualitative Results

Figure 14 shows additional qualitative comparisons on the unseen-trajectory set against the tracker-only baseline, highlighting AdaDexTrack’s robustness.

D. Additional Analysis

D.1. Object Latent Smoothness

In our framework, object latent modulation is intended to act as a continuous interface for skill composition, rather than a discrete object identifier. The general tracker is conditioned on the object latent, and the modulator adjusts this latent online to recruit the most suitable behavior primitives for the current interaction phase. A key question, however, is whether such a mechanism is meaningful when the tracker is distilled from behavior cloning on a finite set of training objects. If the latent-to-action mapping were highly discontinuous, then interpolating in the latent space would not correspond to smooth interpolation between skills.

To examine this, we perform a Lipschitz-style perturbation analysis on the learned object latent. Specifically, we sample 200 reference observations from the unseen-trajectory set, and for each one we randomly select 5 timesteps, yielding 1,000 observations in total. For each observation, we perturb only the object latent by a small

amount ϵ , while keeping the robot state, object pose, and tracking reference fixed. We then measure the change in the tracker output using

$$\rho = \frac{\|\Delta a\|_2}{\epsilon},$$

where Δa denotes the induced change in the predicted action.

As shown in Fig. 12, the response ratio ρ remains bounded across samples and perturbation directions. This indicates that the tracker responds smoothly to local changes in the object latent, rather than switching erratically between unrelated behaviors. Such local smoothness is precisely the property needed for object latent modulation to function as a continuous control variable: nearby latent codes induce nearby actions, making interpolation between latent anchors meaningful at execution time.

This observation is consistent with the qualitative behavior shown in Fig. 5 of the main paper, where the object latent acts like a phase-dependent “knob” for hand openness. The modulator can therefore use small latent adjustments to continuously recruit different grasping behaviors—e.g., preferring a more open hand during approach and a tighter hand during grasp—without changing the reference or relying on large action corrections. Overall, this analysis supports our claim that object latent modulation enables smooth interpolation and switching across the skill library learned by the distilled general tracker.

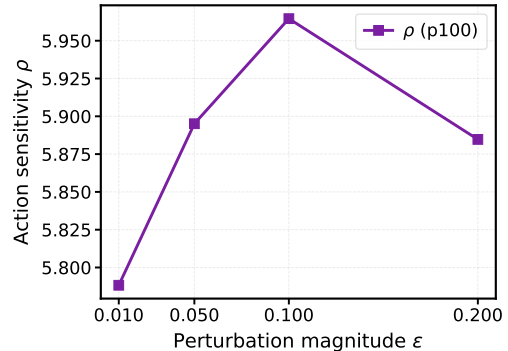


Figure 12. Lipschitz-style perturbation analysis of object latent modulation. The response ratio $\rho = \|\Delta a\|_2/\epsilon$ remains bounded under small latent perturbations, supporting a locally smooth latent-to-action mapping and meaningful interpolation between object anchors.

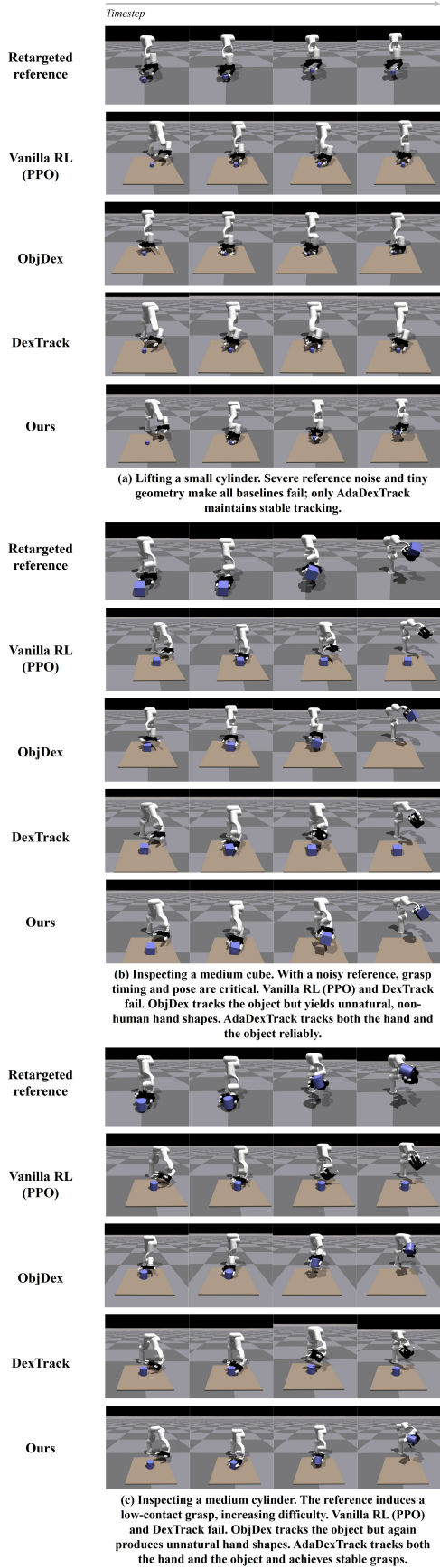


Figure 13. Qualitative results in simulation.

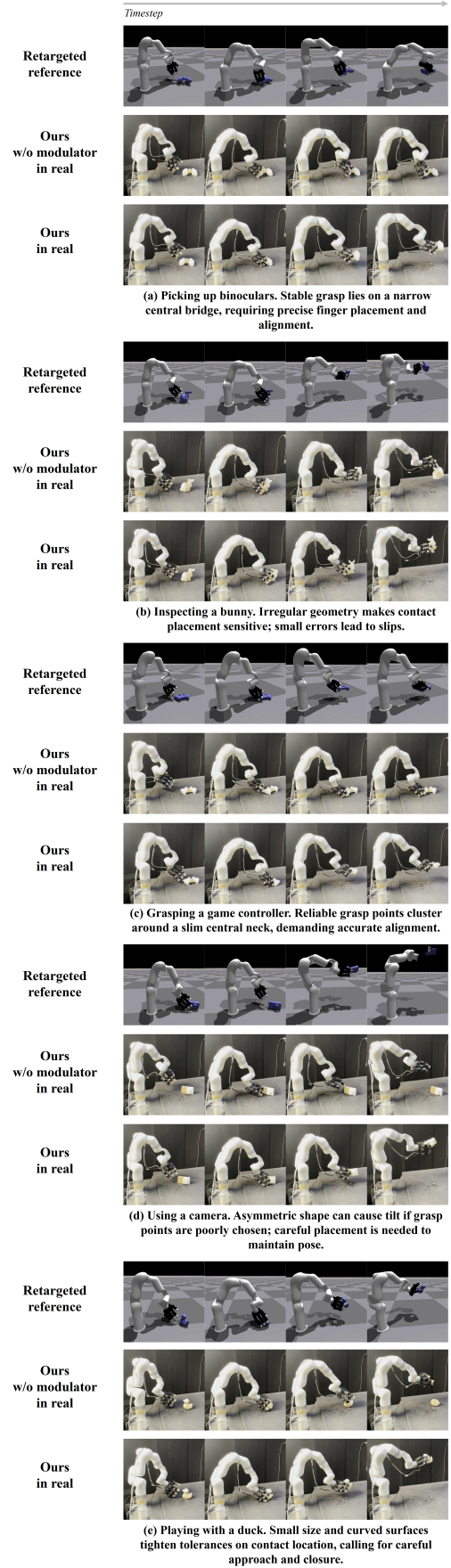


Figure 14. Additional qualitative results in real-world.