APPENDIX

A. Technical Implementation of Pos-to-Pos Non-Collision Module

Due to differences in action space and limitations in the precision of the retargeting algorithm, the configuration of a dexterous robotic hand often generates invalid self-collision configurations. These invalid configurations not only lack operational utility but also risk system failure or hardware damage. To address this issue, we propose a method for mapping invalid configurations to their closest valid counterparts, enabling recovery from self-collision scenarios.

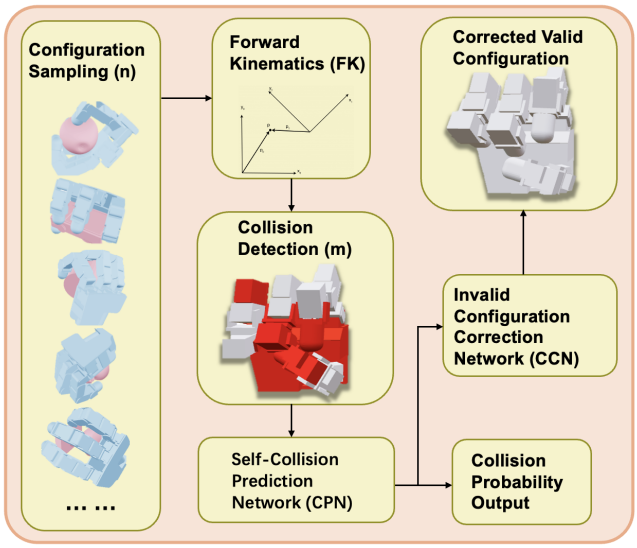


Fig. 8: Pipeline of the Non-collision Module

Fig. 8 illustrates our pos2pos implementation pipeline, which consists of several interconnected components for handling robot hand configurations.

1) Self-Collision Prediction Network (CPN): To facilitate the transformation from invalid to valid configurations, we first develop a **Self-Collision Prediction Network** (CPN). The primary objective of CPN is to predict the likelihood of self-collision for each link within a given joint configuration.

The training dataset is generated by uniformly sampling n configurations from the robot's action space. For each sampled configuration, the system employs forward kinematics (FK) to compute the robot's pose. A collision detection algorithm (e.g., geometric or physics-based) then checks the pose to derive m collision labels for the links. Each label indicates whether a link is in a collision state.

The CPN takes joint configurations as input and outputs collision probabilities for all joints. We train the network using the binary cross-entropy (BCE) loss function, defined as:

$$L_{\text{CPN}} = \frac{1}{m}\sum_{i=1}^{m} \text{BCE}(p_i, t_i), \tag{5}$$

where p_i and t_i represent the predicted and true collision probabilities, respectively.

2) Invalid Configuration Correction Network (CCN): Building on the CPN, we introduce an **Invalid Configuration Correction Network** (CCN) to map invalid configurations to valid ones. The CCN takes an invalid configuration as input and outputs a corrected configuration that minimizes collision risks while closely resembling the original input.

The CCN training process minimizes a composite loss function comprising two components:

- **Mean Squared Error (MSE) Loss**: Ensures the corrected configuration closely resembles the original configuration.

$$L_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n} (\hat{q}_i - q_i)^2, \tag{6}$$

  where q_i and $\hat{q}_i$ denote the original and corrected joint configurations, respectively.
- **Collision Probability Loss**: Leverages the CPN to compute the mean collision probability of the corrected configuration and aims to minimize this value.

$$L_{\text{Collision}} = \frac{1}{m}\sum_{i=1}^{m} p_i(\hat{q}), \tag{7}$$

  where $p_i(\hat{q})$ represents the collision probability of joint i in the corrected configuration $\hat{q}$.

We define the total loss function as:

$$L = \alpha L_{\text{MSE}} + \beta L_{\text{Collision}}, \tag{8}$$

where α and β are hyperparameters balancing the two loss components.

3) Explanation and Optimization Strategy: The loss terms in the proposed framework serve distinct roles:

- L_{MSE} ensures the corrected configuration retains continuity with the original input.
- $L_{\text{Collision}}$ minimizes the likelihood of self-collision in the corrected configuration.
- The hyperparameters α and β significantly influence the training outcomes, and we optimize their values through grid search.

The CCN employs a fully connected multi-layer perceptron (MLP) architecture. The input is the invalid joint configuration q, and the output is the corrected configuration $\hat{q}$. We train the model using the Adam optimizer with a learning rate η and monitor convergence via the collision rate on a validation dataset.

4) Summary: By integrating the CPN and CCN, we efficiently transform invalid self-collision configurations into valid ones. This approach ensures the validity and continuity of robotic configurations, laying a robust foundation for subsequent task execution.

B. Bill of Materials (BOM)

Our teleoperation system supports a modular architecture with flexible input configurations. **Not all devices shown in Fig. 2 are required simultaneously.** Instead, the system requires **one device from each of the two input groups** on the left side of the diagram:

- **Wrist pose acquisition:** RGB(-D) camera, IMU mocap suit, or similar.
- **Hand gesture acquisition:** Mocap gloves, AR tracking, or EMG-based sensing.

This design allows users to build their system using available hardware, optimizing for cost, performance, or ease of use. For example, while a VR headset can serve as a unified sensor in both categories, it is not necessary. Our system can be operated using a standard external monitor, as done in our experiments.

Experimental Setup Used in This Paper: In our experiments, we selected a configuration based on performance, generalizability, and affordability:

- **Mocap Gloves:** $500
  - Kickstarter Product Page
- **IMU-Based Motion Capture Suit:** $200
  - Rebocap Product Page
- **RGB-D Camera (Intel RealSense D435):** $300
  - Amazon Product Page
- **Total Cost:** Approximately **$1000**

Note: Since our method does not depend on depth data, the RGB-D camera can be replaced by a lower-cost RGB camera, further reducing the overall system cost.

Remarks on Reproducibility and Flexibility: The modular nature of the TelePreview architecture allows for hardware substitution based on availability or task requirements. We provide this BOM to facilitate future replication efforts and to emphasize that **our system's affordability claim refers to the teleoperation input interface only**, not the robot arm or end-effector hardware.

C. Reproduction Experience of Baseline Methods

We tested several typical vision-based teleoperation methods under our experimental setup: OpenTeach, OpenTelevision, and AnyTeleop. For methods that did not support LeapHand in the open-source code, we implemented the corresponding parts ourselves to apply these methods.

OpenTeach: Following the guidelines in the open-source repository for adding new hardware, we implemented the retargeting module for LeapHand. However, we encountered two significant issues. First, gesture recognition based on visual input is highly sensitive to occlusions. When the back of the hand is fully occluded or the side of the hand is partially blocked, the positioning of the fingers can deviate substantially—particularly with the Meta Quest3, which has less accurate hand tracking compared to the Apple Vision Pro. Second, the visual approach depends heavily on precise hand calibration, requiring careful adjustment of parameters to achieve acceptable retargeting results. Together, these two factors led to instability in the accuracy of teleoperation in some instances, ultimately impacting the success rate of task completion.

OpenTelevision: The open-source code only provides the necessary components for the 6-DoF Inspire Hand, with no guidance on how to integrate new dexterous hands. Additionally, many hand-specific hyperparameters lack proper documentation. After attempting to integrate LeapHand using the dex-retargetting code and redoing the joint mapping, we were able to align the retargeting results with the hand's movements. However, due to LeapHand's significantly higher degrees of freedom, we encountered severe self-collision issues. During deployment, motor collisions occurred, rendering effective teleoperation impossible. Such self-collision problems did not arise with the original Inspire Hand, as its 6-DoF design inherently prevents the possibility of self-collisions.

AnyTeleop: The retargeting code used here is also based on dex-retargetting. For LeapHand, a high-DOF dexterous hand with a mechanical design that has a higher likelihood of self-collision, relying solely on fingertip-based inverse kinematics (IK) resulted in frequent self-collisions. This posed significant issues during teleoperation.

D. User Study in Subjective and Practice Time Evaluation

To complement the quantitative performance evaluation, we conducted a user study measuring both subjective workload and practice time across different input conditions. Our goal was to assess how the preview system impacts perceived difficulty and actual learning effort.

1) Subjective Workload Ratings: We asked participants to rate their experience using a 5-point Likert scale (higher values indicate greater intensity) across five dimensions: *Mental Demand*, *Physical Demand*, *Total Demand*, *Frustration Level*, and *Perceived Performance*. Each participant completed all tasks under both **w/o Preview** and **w/ Preview** conditions.

As shown in Figure 9, the use of the preview system significantly reduced user-reported workload across all categories. In particular, users reported substantially lower mental and physical demand and improved overall performance perception under the **w/ Preview** condition. This indicates that the preview feature not only improves task success but also reduces cognitive and physical strain.

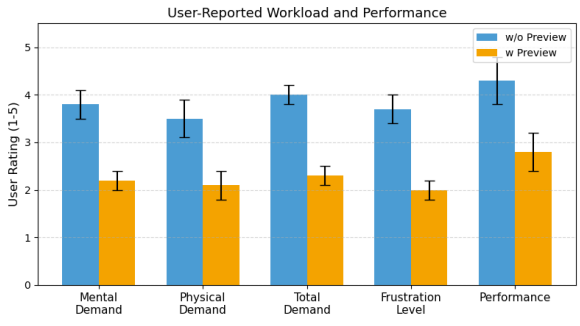


Fig. 9: User-reported workload and performance across five dimensions using in [43]. Ratings were collected using a 5-point Likert scale.

2) Practice Time to Confidence: We also recorded the time participants spent practicing each control modality until they reported being comfortable with beginning formal task trials. Three input conditions were compared: our proposed method **w/ Preview**, the same setup **w/o Preview**, and a

baseline **vision-based method** [3] commonly used in prior work.

As shown in Figure 10, participants required the least time to reach confidence using our full TelePreview setup. The vision-based baseline took the longest and also showed the largest variability across users. These results support the claim that previewed, body-mapped teleoperation enables faster and more intuitive learning.

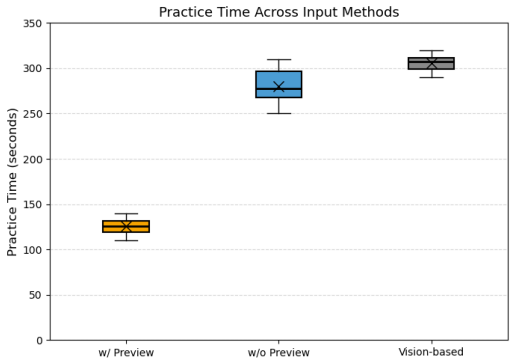
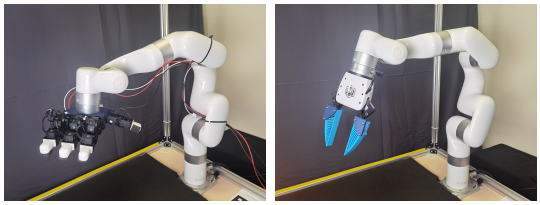


Fig. 10: Practice time. New users reached task readiness faster with our system (w/ Preview) than with other input methods.

E. End-Effector Integration Details



(a) LEAP Hand (b) Gripper

Fig. 11: Deployment on Different Robots.

To demonstrate the generalizability of our system, we deployed TelePreview on a Ufactory xArm robot equipped with three different end-effectors(See in Fig. 11):

- **LeapHand (multi-DoF anthropomorphic hand)**: Controlled via direct mapping of 16 captured joint angles from the mocap glove through our SMPL-X based retargeting pipeline. The preview and execution follow the full configuration space of the robot hand, enabling rich dexterous behaviors.
- **Parallel-jaw Gripper (single-DoF)**: We selected a representative finger joint angle from the mocap glove and used its value to control the gripper's open/close motion. This allowed the system to retain the preview feature without the need for full-hand mapping.
- **Vacuum Gripper (binary actuator)**: We applied a threshold to the same glove joint signal to produce a binary on/off activation, simulating "grasp" or "release" behavior. This minimal control model was still compatible with the preview system.

In all three configurations, only minor parameter adjustments were required (e.g., kinematic model, end-effector transform), and the core TelePreview architecture remained unchanged. These results support our claim that the system is hardware-agnostic and can be adapted to different robot platforms with minimal integration effort.

End-Effector	w/o Preview	w/ Preview	Improvement
LeapHand (16-DoF)	23.6 ± 4.7	13.6 ± 3.2	-10.0
Parallel-jaw Gripper (1-DoF)	16.8 ± 1.7	14.2 ± 1.4	-2.6
Vacuum Gripper (binary)	15.2 ± 1.9	13.7 ± 1.3	-1.5

TABLE III: Average execution time (in seconds) across different end-effectors with and without the preview feature in the Pick & Place task. TelePreview shows the most benefit on the high-DoF LeapHand.

As shown in Table III, the preview system yields the greatest execution time improvement with the LeapHand, supporting our claim that TelePreview is most beneficial for high-DoF, complex manipulation tasks.

F. User Evaluation Questionnaire

After completing the teleoperation tasks under each input condition, participants were asked to rate their experience across five categories using a 5-point Likert scale. Higher scores indicate greater intensity unless otherwise specified.

Question	Rating Scale (1–5)
How mentally demanding was the task?	1 = Very Low, 5 = Very High
How physically demanding was the task?	1 = Very Low, 5 = Very High
How frustrated did you feel while using the system?	1 = Not at all, 5 = Extremely
How much overall effort did the task require?	1 = Very Little, 5 = Very Much
How successful do you feel you were in completing the tasks?	1 = Very Unsuccessful, 5 = Very Successful

TABLE IV: Likert-scale questionnaire completed after each input condition.

Mentally demanding: *The degree of cognitive effort required (e.g., concentration, decision-making).*

Physically demanding: *The amount of physical exertion needed (e.g., hand or body movement, fatigue).*

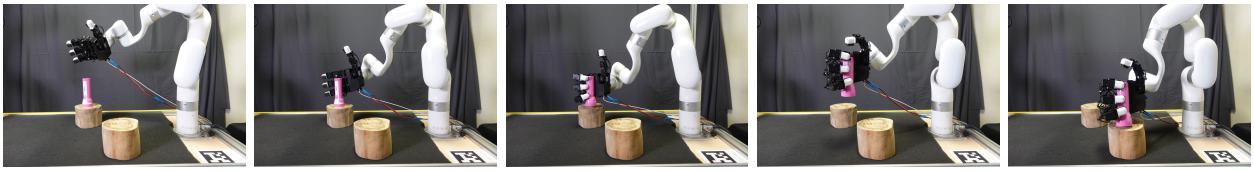
Frustration: *The extent of annoyance, stress, or irritation experienced.*

Overall effort: *The perceived total effort needed to perform the task, combining physical and mental demands.*

Participants answered this questionnaire once for each input modality they used (e.g., *w/ Preview*, *w/o Preview*).

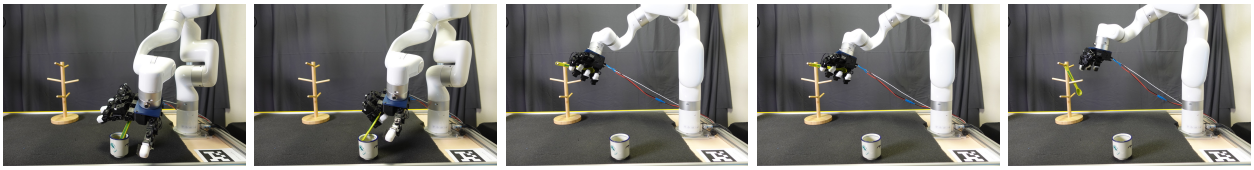
G. Visualization of our tasks

We visualize the execution process of our five manipulation tasks in Figure 12-16. They demonstrate the execution sequences of five manipulation tasks: picking and placing a cup, hanging a spoon on a peg, pouring beans between containers, rotating a box, and stacking cups.



(a) (b) (c) (d) (e)

Fig. 12: Pick&Place Visualization

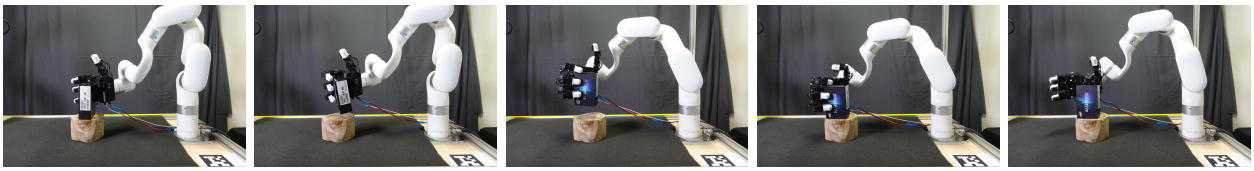


(a) (b) (c) (d) (e)

Fig. 13: Hang Visualization

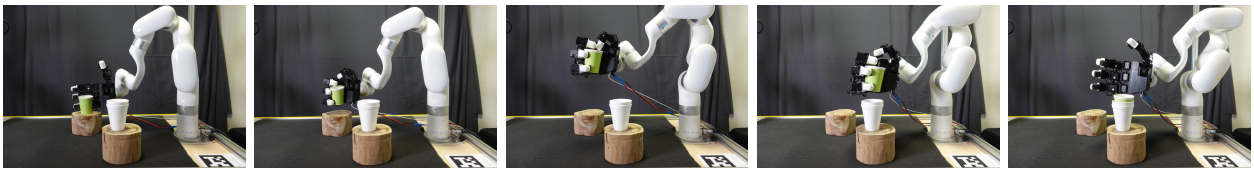
(a) (b) (c) (d) (e)

Fig. 14: Pour Visualization



(a) (b) (c) (d) (e)

Fig. 15: Box Rotation Visualization



(a) (b) (c) (d) (e)

Fig. 16: Cupstack Visualization