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.

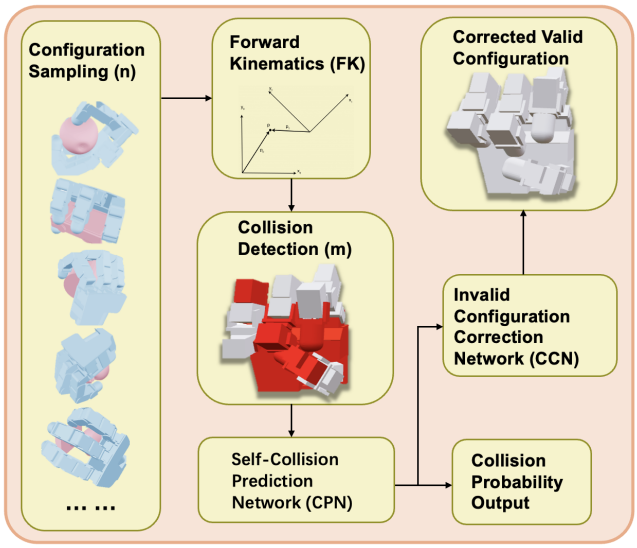


Fig. 8: Pipeline of the Non-collision Module

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:

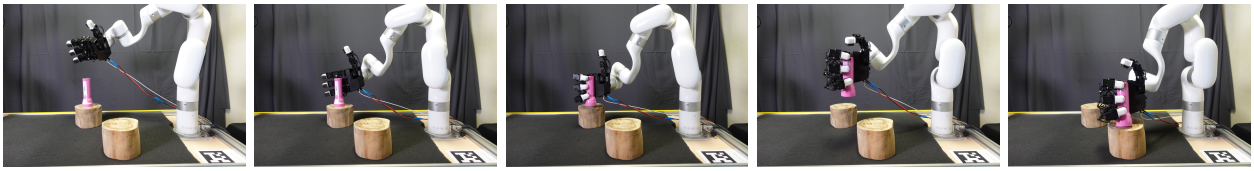
- L_{MSE} ensures the corrected configuration retains continuity with the original input.
- $L_{\text{Collision}}$ minimizes the likelihood of self-collision in the corrected configuration.
- The hyperparameters α and β significantly influence the training outcomes, and we optimize their values through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

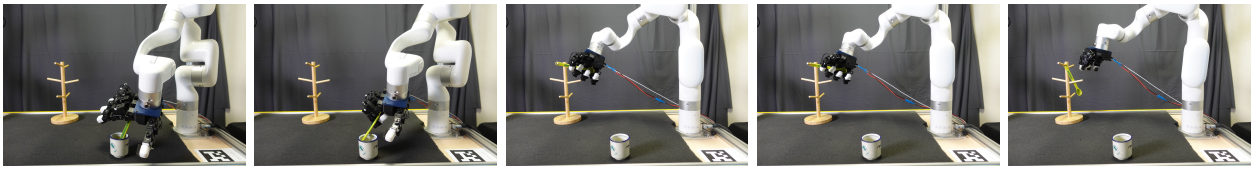
B. Visualization of our tasks

We visualize the execution process of our five manipulation tasks in Figure 9-13. 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. 9: Pick&Place Visualization

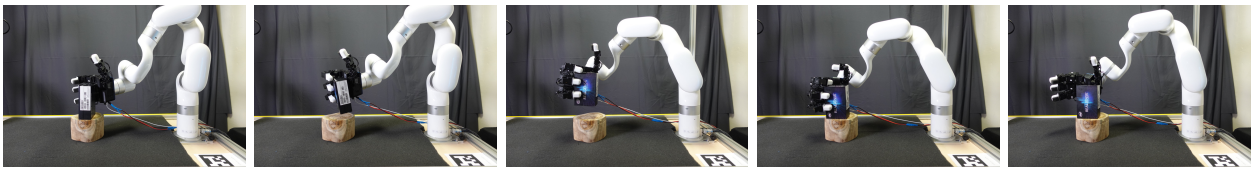


(a) (b) (c) (d) (e)

Fig. 10: Hang Visualization

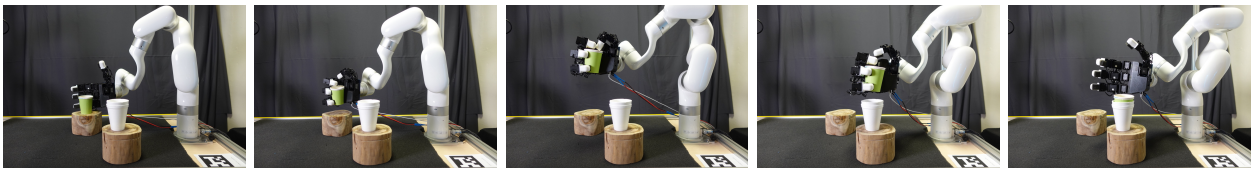
(a) (b) (c) (d) (e)

Fig. 11: Pour Visualization



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

Fig. 12: Box Rotation Visualization



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

Fig. 13: Cupstack Visualization