

DexUMI: Using Human Hand as the Universal Manipulation Interface for Dexterous Manipulation

Anonymous Author(s)

Affiliation

Address

email

A Additional Experiment Results

We show the processed visual observation by the software adaptation layer in policy training data in Fig. 1. Our software adaptation bridges the visual gap by replacing the human hand and exoskeleton in visual observations recorded by the wrist camera with high-fidelity robot hand inpainting. Though the overall inpainting quality is good, we found there are still some deficiencies in the output caused by:

- **Imperfect Segmentation from SAM2:** In most cases, SAM2 can segment the human hand and exoskeleton effectively. However, we notice SAM2 sometimes misses some small areas on the exoskeleton.
- **Quality of inpainting method:** We use flow-based inpainting to replace the human and exoskeleton pixels with background pixels. Though the overall quality is high, some areas remain blurry. We add Gaussian blur augmentation to the images during policy training to make the policy less sensitive to this blurriness.
- **Robot hand hardware limitations:** Throughout our experiments, we found that both the Inspire Hand and XHand lack sufficient precision due to backlash and friction. For example, the fingertip location of the Inspire Hand differs when moving from 1000 to 500 motor units compared to moving from 0 to 500 motor units. Consequently, when fitting regression models between encoder and hand motor values, we can typically ensure precision in only "one direction"—either when closing the hand or opening it. This inevitably causes minor discrepancies in the inpainting and action mapping processes.
- **Inconsistent illumination:** Similar to prior work [1], we found that illumination on the robot hand might be inconsistent with what the robot experiences during deployment. Therefore, we add image augmentation including color jitter and random grayscale during policy training to make the learned policy less sensitive to lighting conditions.
- **3D-printed exoskeleton deformation:** The human hand is powerful and can sometimes cause the 3D-printed exoskeleton to deform during operation. In such cases, the encoder value fails to reflect this deformation. Consequently, the robot finger location might not align with the exoskeleton's actual finger position.

B Evaluation Details

B.1 Initial State Selection

For each task, we manually select a set of initial states for the environment. Objects are placed as diversely as possible within the environment. This set of initial states is shared across all methods. We achieve consistency by placing an additional side camera to record images of all selected initial states. When starting a new evaluation episode, we visualize an image overlay between the recorded pre-selected initial state and the current initial state. We carefully adjust the current setup until it matches the pre-selected initial state with near pixel-perfect alignment.



Figure 1: **Inpainting Results.** The visual observations in the original collected dataset contain exoskeletons and human hands. The software adaptation layer replaces these pixels with corresponding robot hand images while preserving the natural occlusion relationships during hand-object interactions. Please see anonymous project website <https://dexumi.github.io/DexUMI/> for details.

37 Note that due to differences in wrist camera placement relative to the robot flange between the
 38 XHand and Inspire Hand, some initial states viable for the Inspire Hand cannot be completed by the
 39 XHand. For example, if the tea cup is positioned more than 45° to the left of the tea pot (image
 40 space), the XHand’s wrist camera cannot capture the tea cup after grasping the tea due to its camera
 41 positioning (the XHand thumb has a larger range of motion, requiring us to rotate the wrist camera
 42 more toward the thumb direction to obtain clearer visual observations). Consequently, the XHand
 43 and Inspire Hand do not strictly share the same set of initial states for the Tea Picking Using Tool
 44 task. Nevertheless, we ensure their initial states remain within similar distributions and maintain as
 45 much diversity as possible.

46 For the kitchen task, the large workspace presents challenges for a fixed-base single UR5 to cover
 47 diverse initial states, particularly regarding the seasoning bowl location, as the stove and knob posi-

48 tions are fixed. Despite these constraints, we maximize the diversity of bowl placement within the
49 kinematically feasible workspace.

50 B.2 Success Criteria

51 **Cube Picking:** The robot must pick up the red cube and place it into the yellow cup. If the cup falls
52 over after the cube is already placed in it, we still count the episode as successful.

53 **Egg Carton:** We define task success as when the lid is lifted up with its box at an angle greater than
54 30° and the egg box remains stable on the shelf.

55 **Tea Picking Using Tool:** This task consists of two sub-tasks. We define tool picking success as the
56 robot’s ability to steadily hold the tweezers and move them to the tea pot. We define leaf picking
57 success as the robot’s ability to use tweezers to 1) grasp at least one tea leaf from the pot and 2)
58 transfer at least half of the grasped tea leaf into the cup. Subsequent sub-tasks automatically count
59 as failures if the previous sub-task fails, even if the robot can successfully complete the later sub-
60 tasks.

61 **Kitchen Manipulation:** This task consists of three sub-tasks. We define knob closing success as the
62 robot hand rotating the knob by at least 60° from its initial position. We define pan moving success
63 as the robot moving the pan from the stove to the counter without dropping it during transfer. We
64 define the salt task success as the robot 1) grasping some seasoning from the bowl and 2) sprinkling
65 it inside the pan. Subsequent sub-tasks automatically count as failures if the previous sub-task fails,
66 even if the robot can successfully complete the later sub-tasks.

67 B.3 Policy Execution

68 The learned policy predicts 16 steps of future actions, but the robot only executes the first 8 steps
69 and discards the rest. The policy executes at 10 Hz, while the UR5 executes commands at 125 Hz.
70 The Inspire Hand executes at 10 Hz, and the XHand executes at 60 Hz. The 10 Hz policy commands
71 are linearly interpolated to match the desired hardware execution frequency.

72 The action output by the policy contains two components: relative UR5 end-effector action and hand
73 action. The relative end-effector action from the learned policy is converted to absolute by adding
74 the relative action to the current UR5 absolute position in the UR5 base frame. For hand actions, if
75 the action type is absolute, the desired motor value is sent directly to the robot hand for execution. If
76 the hand action type is relative, we first read the current hand motor position, add the relative hand
77 action to it, and then send the result for execution.

78 For the XHand, we found that creating a virtual current hand motor position improves performance
79 compared to reading the current position directly from hardware. Unlike the Inspire Hand motor,
80 which is self-locking, the XHand finger position slightly drifts after encountering external forces
81 (such as the restoring force of tweezers). The 10 Hz policy isn’t reactive enough to adjust for this
82 real-time drifting. Consider the following scenario: the robot hand attempts to close the tweezers
83 to grasp tea leaves. The current motor value obtained by calling the hardware API might already
84 be outdated due to the restoring force of the tweezers (causing fingers to spread wider) when robot
85 execution begins. To address this issue, we initialize a virtual current hand motor position by reading
86 the actual motor position at the beginning of the evaluation. Once the evaluation begins, we update
87 this virtual hand motor position by adding the executed relative hand actions. With this virtual hand
88 motor position approach, finger actions become less impacted by physical drifting, resulting in more
89 precise and reliable grasping operations.

C Exoskeleton Design Details

C.1 Inspire Hand

Underactuated hands like the Inspire Hand typically incorporate closed-loop kinematics, such as four-bar linkages, which cannot be directly represented in URDF. As a result, we cannot initialize the exoskeleton design for the Inspire Hand directly from its URDF model. Instead, our approach is to capture the finger kinematic behavior—specifically, the fingertip poses—and use equivalent general linkage designs with the same degrees of freedom (DoFs) as an initial template for the finger mechanisms. This allows the optimization process to identify parameters that best match the observed kinematics.

To achieve this, we employed a motion capture system (see Fig. 2) to record the fingertip poses in SE(3) space. We 3D-printed marker mounting components for each finger and flange and installed them on the Inspire Hand. For the index, middle, ring, and pinky fingers, each of which has a single DoF, we uniformly sampled 16 motor command values from the lower limit (0) to the upper limit (1000), sent the commands to the fingers, and recorded the corresponding fingertip poses.

For the thumb, which has 2 DoFs—swing and bend—we first fixed the swing value and then uniformly sampled the bend motor values. For example, as shown in Fig. 2d, we set the swing motor to 400 and recorded the fingertip poses by varying the bend motor command. We repeated this procedure for swing values of 0, 200, 400, 600, 800, and 1000.

After obtaining the fingertip poses in the flange coordinate system, we applied the same bi-level optimization formulation defined in Equation 1 in main paper to determine design parameters for each finger. For all five fingers, we employed four-bar linkages as the linkage designs. For each sampled design parameter, We simulate the fingertip poses using PlaCo [2]. For thumb, we minimized the overall loss across all swing motor values, since the thumb’s structural configuration should remain consistent regardless of the swing motor value.

From the optimized design parameters to the physical implementation, we apply three additional steps to ensure that the exoskeleton mask consistently covers the real Inspire Hand. First, we extend the length of the last link of each finger in the exoskeleton design by 3 mm beyond the optimized value. This guarantees that the exoskeleton mask always fully covers the last link of the actual Inspire Hand. Second, we increase the width of the thumb’s four-bar linkage to eliminate any hollow regions in the camera’s field of view, thereby maintaining the visual integrity of a continuous exoskeleton mask. Third, we conservatively tighten the joint limits by 5° at each joint to ensure the mask continues to cover the real Inspire Hand even when structural deformation occurs due to the limited strength of the 3D-printed PLA-CF material.

C.2 XHand

Since the URDF file of the XHand is well-organized, with each joint origin defined at the location of its corresponding rotary joint, we can directly extract link lengths from the URDF structure. In cases where the exact values are not specified, we can perform reverse modeling using the STL meshes from URDF file to recover geometric features near each joint and manually measure link lengths in CAD software.

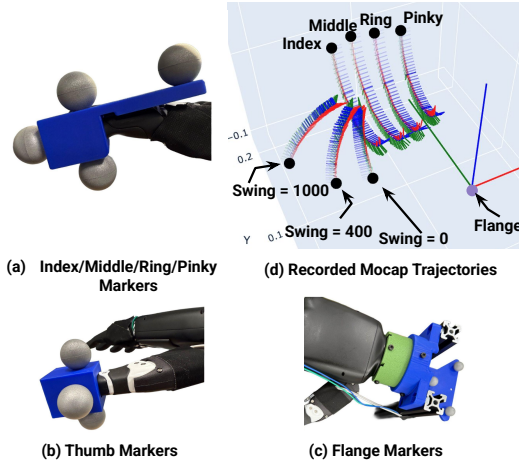


Figure 2: **Inspire Mocap:** We use motion capture system to record fingertips trajectories in the flange coordinate. We attached marker on fingers and flange to capture the fingertip pose in flange coordinate.

Joint limits are also specified in the URDF file and are implemented in the exoskeleton design by physically constraining the link motion to prevent rotation beyond the specified range. Similar to the Inspire Hand exoskeleton design, we adopt a conservative strategy when applying these limits setting slightly tighter bounds on each joints. For example, if the actual joint rotation range is -110° to 20° , the corresponding exoskeleton limit is set to -105° to 15° . This precaution accounts for possible deformation of the 3D-printed exoskeleton links under human-applied torque, which can introduce unintended joint deflection. Without this buffer, the exoskeleton might deform beyond the physical limits of the XHand, leading to an embodiment gap.

When converting the link lengths to the actual exoskeleton design, two primary constraints must be considered. The first is *wearability*. To ensure that the human operator can comfortably wear the exoskeleton, the structure must be hollowed out as much as possible, allowing the finger to pass through unobstructed. The second constraint is *material strength*. Through empirical testing, we determined that the optimal minimum structural width for 3D-printed PLA-CF material is 4 mm. Therefore, any part expected to experience significant stress is reinforced to be at least 4 mm thick in the final design.

D Sensor Details

D.1 Joint Encoder

Our exoskeleton uses Alps RDC506018A rotary sensors as encoders at every joints. These are resistive sensors whose resistance varies approximately linearly with absolute angular position.

As shown in Fig. 3, when the joint rotates, the voltage on the ADC line changes proportionally. This analog voltage signal is then sampled by an Analog-to-Digital Converter (ADC) on a microcontroller unit (MCU). Then the joint angle α_{joint} can be estimated as:

$$\alpha_{\text{joint}} = \frac{V_{\text{ADC}}}{3.3 \text{ V}} \times 360^\circ$$

However, this simple voltage divider circuit has a significant failure mode: if the power supply (3.3 V in our case) is unstable due to temperature drift in semiconductor components or ripple from DC-DC converters and LDOs, the joint angle reading will drift accordingly. To mitigate this issue, we simultaneously measure the supply voltage through another ADC channel. Instead of dividing by a fixed 3.3 V, we normalize the sensor voltage using the measured supply voltage when computing the joint angle:

$$\alpha_{\text{joint}} = \frac{V_{\text{ADC}}}{V_{\text{supply}}} \times 360^\circ$$

This voltage normalization runs in real time on the MCU. After computing the joint angles, the MCU packs all joint values into a single data packet with a fixed 2-byte header and a checksum tail. The header simplifies decoding by allowing the receiver to locate a known keyword in variable-length data streams, while the checksum ensures packet integrity. The final data packet is transmitted to the host computer via a Universal Asynchronous Receiver-Transmitter (UART) interface.

D.2 Tactile

For commercial dexterous hands without built-in tactile sensors (e.g., the Inspire Hand in our evaluation), we use a simple and low-cost Force-Sensitive Resistor (FSR) as the tactile sensor. When no force is applied, the FSR exhibits a resistance of several megaohms, while under significant force,

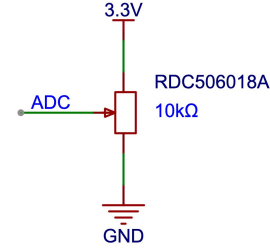


Figure 3: **Joint Encoder Circuit:** The rotary sensor acts as a variable resistor with three output pins. As it rotates with the joint, the voltage on the ADC line changes approximately linearly.

the resistance drops to the kilohms range. As shown in Fig. 4, the FSR is incorporated into a simple voltage divider circuit to produce an analog voltage signal. The divider resistor R_1 is selected to be comparable to the minimum resistance of the FSR. Since the FSR resistance is approximately inversely proportional to the applied force, we can express the force using a constant scale factor k as:

$$F = k \left(\frac{V_{\text{supply}}}{V_{\text{ADC}}} - 1 \right)$$

In our experimental setup, the same FSR sensor is mounted on both the dexterous hand and the exoskeleton. For simplicity, we directly use the V_{ADC} reading as a proxy for tactile input.

For hands equipped with onboard tactile sensors (e.g., the XHand), we install the same type of sensor as used in the hand. In our setup, this sensor is a magnet-based tactile array capable of measuring three-dimensional forces across 120 points on its surface. The force data is output via an SPI communication interface using a proprietary protocol. By configuring this interface on our embedded system, the force array can be successfully transmitted to the host machine.

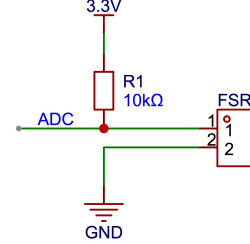


Figure 4: **Voltage Divider Circuit:** This simple voltage divider circuit converts the resistance change of the FSR sensor into an analog voltage on the ADC line.

E Data Collection and Policy Training details

E.1 Data Collection

We collected 310 trajectories for Cube Picking task policy training, 175 trajectories for Egg Carton Opening task policy training, and 400 trajectories for Tea Picking Using Tools policy training (for both Inspire Hand and XHand). For the kitchen task, we collected 370 trajectories covering all four sub-tasks, plus an additional 100 trajectories focused solely on knob closing.

For the Inspire Hand, all data types—including wrist position from ARKit, policy visual observations from the wrist-mounted camera, joint angles from encoders, and tactile feedback—were recorded at 45 FPS. For the XHand, we recorded at 30 FPS, as the tactile sensor readings became unstable at higher recording frequencies. For each data type, we recorded the receive timestamp t_{receive} when the data arrived at the recording buffer.

We wear green gloves when collecting data with exoskeleton as we use green PLA-CF to 3D-printed the exoskeleton. We found consistent color helps SAM2 to yield better segmentation results.

E.2 Training Data Latency Management

There is an inherent latency between the time when sensors capture data and when that data actually arrives in the recording buffer. To ensure our imitation learning policy receives properly aligned observations (visual observations, tactile sensor readings) and actions (joint encoder readings), we calculate the actual data capture time using $t_{\text{capture}} = t_{\text{receive}} - l_{\text{sensor}}$, where l_{sensor} refers to the latency from capture to receive for a particular sensor. We measure the iPhone and OAK camera latency by reading a rolling QR code displayed on a computer monitor showing the current computer system time, as proposed in UMI [3]. The camera and iPhone latency is calculated as $l_{\text{camera}} = t_{\text{receive}} - t_{\text{display}} - l_{\text{display}}$, where l_{display} represents the monitor refresh rate.

The encoder latency is adjusted by examining the overlay image between the recorded exoskeleton image and the corresponding robot hand image from action replay. If the encoder latency is set too high, the robot hand fingers will execute future actions and lead in the overlay image. If the encoder latency is set too low, the robot hand fingers will lag behind the exoskeleton fingers in the overlay image. We tune the encoder latency until the exoskeleton fingers and robot fingers are

226 perfectly aligned. Once all data timestamps are adjusted, we linearly interpolate the joint angles
227 and tactile readings to obtain data points properly aligned with the camera timestamps. Finally, We
228 downsample the data by a factor of 3 to reduce the policy training time.

229 **E.3 Policy Training**

230 We process the visual observations with pretrained DINO-V2 [4]. Before passing the visual observa-
231 tions into DINO-V2, we augment it with random crop, color jitter, random grayscale and Gaussian
232 Blur. We concatenate the CLS token from DINO-V2 with tactile sensor readings as input to the
233 diffusion policy [5]. The policy predicts 16 steps of robot actions, which contain both 6-DoF robot
234 end-effector relative actions and hand actions (6-DoF for Inspire Hand and 12-DoF for XHand). We
235 train the models for 400 epochs across all tasks for both types of hands. The pretrain DINO-V2 is
236 not frozen and updated during the policy training.

References

- [1] L. Y. Chen, C. Xu, K. Dharmarajan, M. Z. Irshad, R. Cheng, K. Keutzer, M. Tomizuka, Q. Vuong, and K. Goldberg. Rovi-aug: Robot and viewpoint augmentation for cross-embodiment robot learning. arXiv preprint arXiv:2409.03403, 2024.
- [2] P. Contributors. Placo: A python library for planning and control, 2025. URL <https://placo.readthedocs.io/>. Accessed: May 8, 2025.
- [3] C. Chi, Z. Xu, C. Pan, E. Cousineau, B. Burchfiel, S. Feng, R. Tedrake, and S. Song. Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots. arXiv preprint arXiv:2402.10329, 2024.
- [4] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, M. Assran, N. Ballas, W. Galuba, R. Howes, P.-Y. Huang, S.-W. Li, I. Misra, M. Rabbat, V. Sharma, G. Synnaeve, H. Xu, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski. Dinov2: Learning robust visual features without supervision, 2024. URL <https://arxiv.org/abs/2304.07193>.
- [5] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. In Robotics: Science and Systems, 2023.