Robotic Telekinesis: Learning a Robotic Hand Imitator by Watching Humans on Youtube

Anonymous CVPR submission

Paper ID 8108

Abstract

We build a system that enables a human to control a robot hand and arm, simply by demonstrating motions with their own hand. The robot observes the human operator via a single RGB camera and imitates their actions ***in real-time****. Human hands and robot hands differ in shape, size, and joint structure, and performing this translation from a single uncalibrated camera is a highly underconstrained problem. Moreover, the retargeted trajectories must effectively execute tasks on a physical robot, which requires them to be temporally smooth and free of self-collisions. Our key insight is that while paired human-robot correspondence data is expensive to collect, the internet contains a massive corpus of rich and diverse human hand videos. We leverage this data in order to train a system that understands human hands, and retargets a human video stream into a robot hand-arm trajectory that is smooth, swift, safe, and semantically similar to the guiding demonstration. We demonstrate that it enables previously untrained people to teleoperate a robot on various dexterous manipulation tasks. We hope this work makes robot teaching more accessible, and that in the future, our system can be used to aid robots that learn to act autonomously in the real world.*

1. Introduction

Building robots that mimic human behavior has been a central component of robotics research for decades. This paradigm, known as teleoperation, was historically used to enable robots to perform tasks that were unsafe or impossible for humans to perform, such as handling nuclear materials [32] or deactivating explosives [11]. More recently, teleoperation has been used to enable the robotic automation of tasks that are easy for humans to demonstrate but difficult to program. In industrial robots, for example, teleoperation can be used to collect a single trajectory (e.g. picking a box from a conveyor belt) that the robot repeats verbatim for months or years thereafter. These robots, however, *overfit* to a single type of motion in a single environment; performing any other task (or even variants of the same task) would require additional teleoperated demonstrations. A recent line of research has therefore focused on using teleoperation as a means to collect a large dataset of demonstrations, which can then be used to learn a policy that *generalizes* to new tasks in unseen environments [25, 26].

Figure 1. Our system leverages passive data from the internet to enable robotic real-time imitation in-the-wild.

In this paper, we study the problem of teleoperation for dexterous robotic manipulation. Most current techniques can be grouped into four broad categories. (a) *Kinesthetic Control*: a human physically holds a robot, and moves its joints to desired positions. This approach is laborious for demonstrators, and is limited to small robots that can easily be maneuevered by a human. (b) *Virtual Reality*: Devices like the Oculus Rift allow demonstrators to use wearable headsets or joysticks to control a robot, but these methods cannot scale to high-dimensional robot hands, and are limited to parallel jaw grippers at best. (c) *Haptic Gloves* [2] embed sensors into wearable gloves that track human hand motion, and relay haptic feedback to the demonstrator. (d) *MoCap*: motion capture markers are placed on several points on the demonstrator's hand, and are used to track the hand over time. [36]. All of these setups involve expensive hardware, specialized engineering, and expert operators, and may impede the natural fluid motion of the demonstrator's hand.

The challenge of overcoming these shortcomings grows

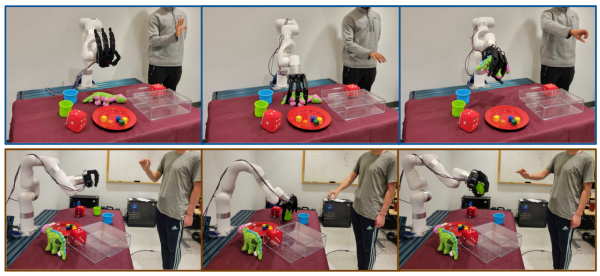

Figure 2. Operators completing a stuffed dinosaur pickup task and cup pickup task. Videos are in the supplementary.

exponentially with the complexity of the robot to be controlled; multi-fingered hands are far more difficult to teleoperate than two-finger grippers. Despite recent advancements, building an easy-to-use, performant and low-cost teleoperation system for high-DoF dexterous manipulation has remained elusive. Handa *et al.* recently proposed DexPilot [14], a low cost system for vision-based teleoperation that is free of markers or hand-held devices. DexPilot significantly lowers the cost and usability barrier, but relies on a custom setup with multiple calibrated depth cameras, and uses neural networks trained on images collected in this controlled environment, which limits its use to a specific lab setting.

The objective of this paper is to enable teleoperation of a dexterous robotic hand, *in the wild*. Our system must work for any operator, on any task, in any environment. We call our system *Robotic Telekinesis*, as it provides a human the ability to control a dexterous robot from a distance with a single camera and no physical interaction. Unfortunately, this poses a chicken-and-egg problem: to train a teleoperation system that can work in the wild, we need a rich and diverse dataset of paired human-robot correspondences. However, to collect this kind of data, we need an in-the-wild teleoperation system.

While we lack paired human-robot data, there is no shortage of rich and diverse human data, and our key insight is to leverage the massive corpus of human videos available on the internet. This data is *passive*, in the sense that it is readily available for free without any collection effort, as opposed to data actively gathered in our lab. We propose a method that uses this passive data, and draws on the latest advancements in 3D human pose estimation, to train a system that understands human hand behavior and retargets human motion into robot actions. Our system allows an operator to control a four-finger 16-DoF hand, mounted on a robotic arm, simply by moving their own hand and arm, as illustrated in Figure 1. Our system is low-cost, glove-free and marker-free, and it requires only a single uncalibrated color camera with which to view the operator. We demonstrate the usability and versatility of our system on 10 challenging dexterous manipulation tasks. We further demonstrate the generality and robustness of our system by performing a systematic study across ten previously untrained human operators.

2. Related Work

Understanding Human Hands and Bodies. Modeling and estimating human poses is a widely studied problem, with applications in graphics, virtual reality and robotics. The recent research most relevant to our work can roughly be divided into four sub-areas. (1) *Hand and body modeling.* MANO [27] is a low-dimensional parametric model of a human hand, and SMPL [22] is an analogous model for the human body. SMPL-X [23] is a single unified model of both the body and hands. (2) *Monocular human hand and body pose estimation*. Recent works in human pose estimation typically estimate 2D quantities like bounding boxes [29] or skeletons [6], or perform a full 3D reconstruction [12, 18, 34]. Rong *et al.* [28] propose a method for integrated 3D reconstruction of human hands and bodies. (3) *Dataset curation*: The advances in human pose estimation crucially rely on large datasets of human hand and body poses. FreiHand [37], Human3.6M [17] and the CMU Mocap Database [1] are examples of densely-annotated datasets in clean indoor settings. On the other end of the spectrum, the 100 Days of Hands [29] and Epic Kitchens [10] datasets are massive collections of raw videos that span a rich and diverse set of hand poses and motions, but don't contain pose annotations. (4) *Understanding human hand function*. Brahmbhatt *et al.* [4] use thermal imaging to capture impact heatmaps that reveal patterns in the ways human hands interact with everyday objects. Taheri *et al.* [30] study the problem of how human hands and bodies behave while grasping and manipulating objects, and propose a method to generate plausible human grasps for novel objects. Hasson *et al.* [15] and Cao *et al.* [7] propose methods for joint hand and object reconstruction, and Hampali et al. [13] present a dataset of hand-object interactions with 3D annotations. Liu *et al.* [21] builds a taxonomy of human grasps to understand the cognitive patterns of human hand behavior [20].

Kinematic Retargeting and Visual Teleoperation. Human pose tracking only solves half of the visual teleoperation problem. Mapping human poses to robot poses is itself a difficult challenge, because humans and robots have very different kinematic structures. Li *et al* [19] train a deep network to map human hand depth images to joint angles in the robotic Shadow Hand, and Antotsiou *et al* [3] combine inverse kinematics and Particle Swarm Optimization to retarget human hand poses to a high-dimensional robot hand model. Our system follows the method of DexPilot [14], which minimizes an cost function that captures the functional similarity between a human and a robot hand.

The general problem of kinematically retargeting motion in one morphology into another is also studied outside of robotic manipulation. Villegas *et al.* [33] propose a cycle consistency objective to transform motion between animated humanoid characters of different body shapes. Peng

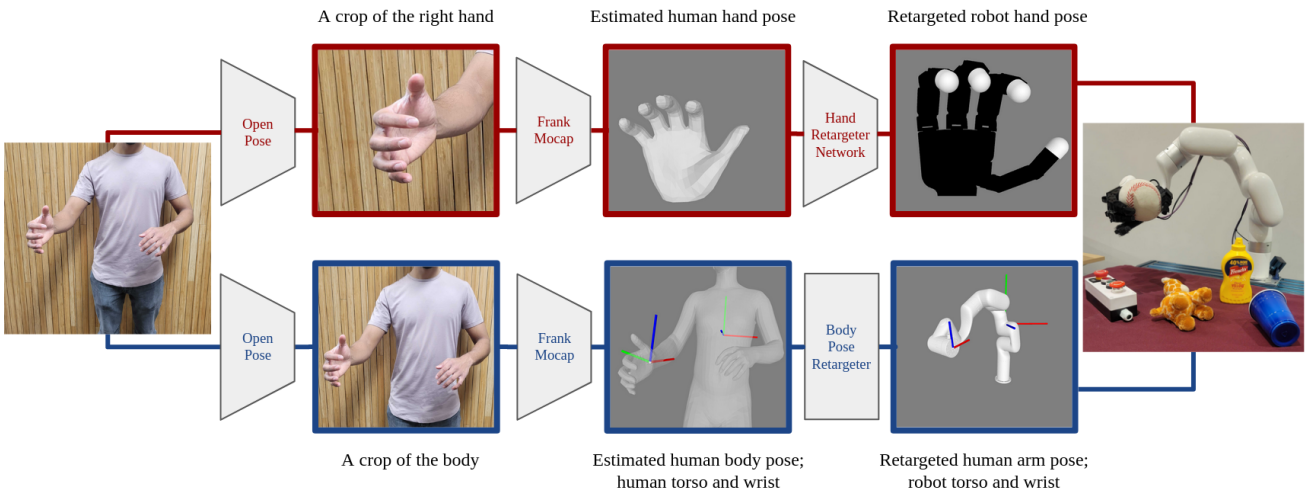

Figure 3. A graphical description of our visual teleoperation pipeline. A color camera captures an image of the operator. To command the robot hand, we pass a crop of the operator's hand to a hand pose estimator, then run our hand retargeting network to map the estimated human hand pose to a robot hand pose. To command the robot arm, we pass a crop of the operator's body to a body pose estimator, and use cross-body correspondences to determine the desired pose of the robot's end-effector from the estimated body pose. We finally send commands to both the robot hand and arm.

et al [24] use an approach based on keypoint matching to learn robotic locomotion behaviors from demonstration of walking dogs. Zakka *et al.* [35] learn a visual reward function that allows reinforcement learning agents to learn from demonstrators with different embodiments.

3. Realtime Human-to-Robot Retargeting

Our system consists of an xArm6 robot arm, a 16-DoF Allegro robot hand and a single RGB camera that captures a stream of images of the human operator. The camera can be placed anywhere as long as the operator is within the camera's field of view. Each captured image at 30hz is *retargeted* into a pair of robot commands (one for the Allegro hand and one for the xArm) which place the robot hand and arm in poses that match the hand-arm poses of the human operator. This capture-retarget-command loop allows the operator to guide the robot to complete tasks while watching the robot mimic them in real-time. Figure 2 illustrates operators using the system to solve a pair of grasping tasks.

In the rest of this section, we describe the implementation of our visual teleoperation pipeline (see Figure 3 for a graphical description). One stream estimates the pose of the operator's hand, and retargets it into a robot hand pose, and a second stream estimates the pose of the operator's body in order to determine the pose of the robot's arm.

3.1. From Human Hand Poses to Robot Hand Poses

We represent a human hand as a vector of parameters for the MANO model [27] and use a Convolutional Neural Network (CNN) (from [28]) to regress these hand shape and pose parameters from an RGB image. The Allegro hand has a known kinematic structure defined by its URDF, and its pose is entirely determined by the angles of its 16 joints (four in each finger). The two hands differ greatly in shape, size and joint structure, which makes it difficult to design a meaningful similarity metric between them. Inspired by [14], we posit that for any given human hand pose, the optimal corresponding robot hand pose is the one that best mimics the *functional intent* of the human. Consider a human performing a pinch grasp with the tips of their thumb and index finger 1cm apart, while simultaneously pressing a button with their outstretched ring fingertip positioned 15cm away from the center of their palm. In order for the robot hand to effectively mimic this action, the robot's thumb and index fingertips should also be 1cm apart, and the robot's ring fingertip should be 15cm away from its palm center. The joint angle values don't need to match those of the human's hand, but we must preserve the relative vectors between a set of functionally meaningful keypoints so the robot hand has the same effect on the world.

3.1.1 Retargeting as Inverse Kinematics

The notion of functional intent is formalized with an *energy function* that captures the degree of dissimilarity between a human hand pose and a robot hand pose. (Per convention, a high energy means the two poses are highly *dis*similar.) Following [14], we define a set of five hand keypoints: the tips of the index, middle, ring and thumb fingers, and the center of the palm. We define a set of ten keyvectors, each of which connects a pair of keypoints (see Figure 4). Each keyvector has one endpoint designated as the origin, and the

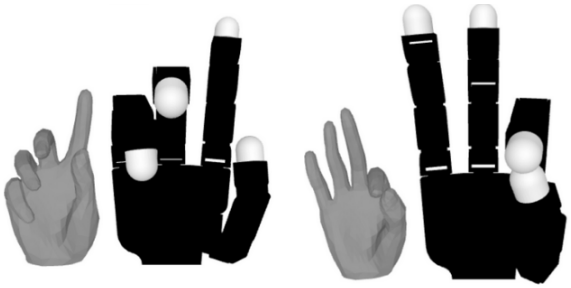

Figure 4. **Human-to-robot Translations**. The inputs and outputs of our hand retargeting network. Each of the pairs depicts a human hand pose, and the retargeted Allegro hand pose.

other as the tip, and the vector is expressed in the coordinate system of the origin keypoint. Our energy function is a weighted sum of ten cost terms, where the i-th cost term measures how much the i-th keyvector on the posed human hand differs from the i-th key vector on the posed robot hand.

The problem of mapping a human hand pose to a robot hand pose now reduces to the problem of finding the configuration of robot hand joint angles that minimizes this energy. We can view this search problem as a particular instantiation of the general Inverse Kinematics (IK) problem, which aims to find the joint angle configuration that places the end of a kinematic chain at a desired position and orientation relative to the start of the chain. While the prototypical IK problem aims to achieve a single end-effector target relative to a fixed base, here we must jointly optimize for several end-effector constraints, all relative to each other.

3.1.2 Retargeting with Online Gradient Descent

To find solutions, our first key insight is that for a fixed human hand pose, the energy function is a differentiable function of the Allegro joint angles, because the forward kinematics is a differentiable operation. One natural approach is to use the energy function as an objective for an optimization problem, and solve it using online gradient descent at inference time. At each timestep, for an observed hand pose x, this approach would iteratively search for the robot pose y^* that is minimizes the energy E with respect to x:

$$y^* = \underset{y}{\operatorname{argmin}}\, E(x, y), \tag{1}$$

With proper tuning and accelerated code, this approach can be quite effective [14]; however, in our experiments, we found inference-time optimization to be prohibitively slow and difficult to tune. The low-latency requirements of our system imposes a budget on the number of gradient steps we can afford to run at each timestep, and this budget was often insufficient for gradient descent to reach a satisfactory solution. This lag would compound and as the operator's hand moved, the robot would fall further behind. Seeding the search from the result of the previous helped, but had the adverse effect of magnifying problematic edge cases. Due to the non-convexity of the energy function, we occasionally observed strange local minima, and because we seed each search from the previous result, these minima were often difficult to escape. This resulted in the robot's hand staying stuck in an erroneous pose far after the operator had moved on. We address this issue by leveraging internet human data.

3.1.3 Retargeting with Neural Networks

Instead of optimizing at inference, we exploit the differentiability of our energy function to train a neural network that learns to map human hand poses to robot hand poses. Our hand retargeter network is a Multi-Layer Perceptron (MLP) that takes as input a human hand pose and outputs Allegro joint angles. At each training iteration, we compute the energy function from the input human hand pose and the output robot hand pose, and use this as our loss function. Training a neural network effectively amortizes the cost of optimization over the course of training, and each gradient step minimizes the energy function on an entire minibatch of human hand poses at a time. To train the network, we use the FreiHand dataset [37], which contains ground-truth MANO model parameters for a diverse collection of human hand poses. We additionally curate a collection of about 20 million images from the Epic Kitchens [10] and 100 Days of Hands [29] video datasets, and use the hand pose estimator from [28] to generate training samples for the hand retargeting network. These videos capture contain millions of hand poses used in day-to-day tasks.

It is important to note that our network does not regress to a ground-truth target. In most neural network regression problems, the goal is to learn a mapping f from a source domain $\mathcal{X}$ to a target domain $\mathcal{Y}$. At training time, the network has access to *paired* examples $(x \in \mathcal{X}, y \in \mathcal{Y})$. The loss function is typically a measure of how much the network's predicted $\widehat{y}$ deviates from the ground-truth target y. Here, our source domain $\mathcal{X}$ is the set of all human hand poses and our target domain $\mathcal{Y}$ is the set of all robot hand poses, *but we only have training data from the source domain.* For any given human hand pose x, there is no ground-truth robot hand pose y; our loss function is the energy function that compares the predicted robot hand pose $\widehat{y}$ to the network's input x. The neural network optimizes the following objective:

$$\underset{f}{\operatorname{argmin}}\, \mathbb{E}_{x \in \mathcal{X}} \Big[E(x, f(x)) \Big], \tag{2}$$

where $E(x, y)$ denotes the energy function that captures the dissimilarity between a human hand pose x and a robot hand pose y. By minimizing the *expected loss* over an entire dataset of human hand poses, we learn a retargeter that may

produce occasional errors on uncommon hand poses, but performs well on the types of natural hand poses commonly seen during teleoperation.

During deployment, for each image we capture, we estimate the operator's hand pose and directly compute the corresponding Allegro joint angles via a single forward pass through the network (see Figure 3). The network runs extremely fast and takes about 3ms per forward pass.

3.1.4 Collision Avoidance via Adversarial Training

Using a neural network to perform human-to-robot hand retargeting has another subtler advantage: terms that are slow to compute can be added to our energy function. When training a neural network, we can run expensive operations in order to compute the loss at each iteration. This allows us to use any energy function, as long as it's differentiable. We simply absorb the computation cost during training instead of incurring it at deployment time.

We exploit this idea to address the problem of self-collisions. Minimizing the keyvector-similarity energy function described above can sometimes yield robot hand joint configurations that place the hand such that fingers collide with each other or with the palm. It is difficult to add a term to the energy function that penalizes such configurations as self-collision-ness is not a differentiable function of the robot's joint angles. To address this, we first train a classifier that takes in a robot joint angle vector, and tries to classify whether or not this joint configuration results in self-collision. This classifier is an MLP, and we generate its training data by repeatedly sampling a joint angle vector, and generating a ground-truth self-collision label by querying a (non-differentiable) self-collision checker.

Once our self-collision classifier is trained, we use it as a "discriminator" to train our retargeter network. At every training iteration, we pass the predicted robot joint angle vectors to the self-collision classifier. The predicted self-collision probability is used as a term in the loss function, and its gradient can be backpropagated through the self-collision classifier, and used to update the weights of the retargeter network, as shown in Figure 5. Our self-collision classifier is akin to the discriminator in a Generative Adversarial Network (GAN), though in our case, we pretrain and freeze the self-collision classifier so we don't suffer the notorious instability of jointly optimizing a discriminator and a generator.

3.2. From Human Body Poses to Robot Arm Poses

A hand that can flex its fingers but cannot move around will not be able to solve many useful tasks. In order for a robot to effectively mimic the actions of a human operator, we additionally need a module that observes an image of the operator and determines how the robot's arm should move

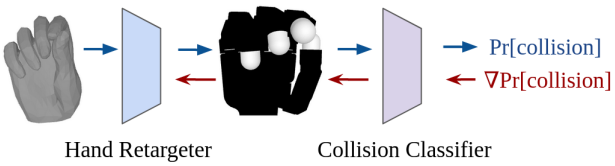

Figure 5. A trained self-collision classifier is used as an adversarial loss that penalizes self-colliding joint configurations. The blue arrows denote the forward pass, and the red arrows denote the flow of gradients during the backward pass.

to place its hand at the proper location and orientation.

A common approach is to rigidly mount a camera with a fixed known transformation relative to the robot's base frame, localize the position and orientation of the operator's wrist in the 3D camera coordinate frame, and use the known camera-to-robot transformation (extrinsics) to determine how the robot's wrist should be positioned and oriented in the robot's base coordinate frame. However we aim to build a system that operates from a single color camera, without depth sensors, and without intrinsic or extrinsic camera calibration. This means we lack a common "ground" 3D coordinate frame with which to relate the operator's and robot's worlds.

We solve the lack-of-grounding problem by defining pairs of corresponding coordinate frames between human and robot "bodies". We use the human's torso as the origin of an anchor coordinate frame, and choose a suitable point to serve as the "robot's torso". We similarly correspond the human's right hand wrist with the robot's wrist link. The axes of these torso and wrist coordinate frames are aligned such that if the operator's hand were, say, 10cm in front of their torso, the corresponding robot's hand would be 10cm in front of its "torso". If the operator rotated their wrist so that the palm faced upward, then the robot would rotate its wrist to make its palm face upward.

At runtime, when we capture an image of the operator, we first estimate the pose of the operator's body. We model the human body using the parametric SMPL-X model, and use a CNN [28] to estimate its parameters. One of these parameters is a set of 24 rotation matrices, each of which represents the relative rotation at one of the joints on the SMPL-X human skeleton model. One of these 24 skeleton joints is the torso, and one is the right hand wrist. By traversing the kinematic chain from the torso joint to the right hand wrist joint, we compute the relative position and orientation between the human's right hand wrist coordinate frame and their torso coordinate frame (see Figure 3). Using our pre-specified human-to-robot correspondences, we compute the pose of the robot's end-effector relative to its base frame, such that the relative transformation between the robot's wrist and its torso is preserved. We then use an IK solver [5] to compute arm joint angles that place the robot's end-effector at this desired pose, and we send these desired

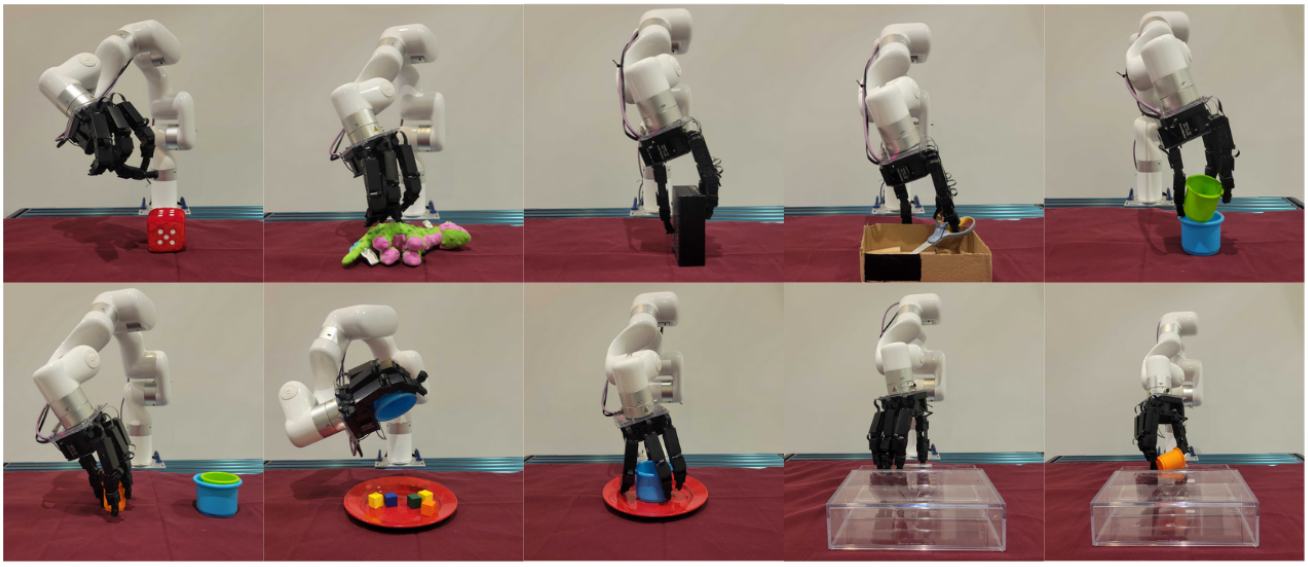

Figure 6. A graphic showing the robot completing tasks with the expert operator. These tasks from left to right, from the first row are Pickup Dice Toy, Pickup Dinosaur Doll, Box Rotation, Scissor Pickup, Cup Stack. On the second row, two cup stacking, pouring cubes onto plate, cup into plate, open drawer and open drawer and pickup cup. Please see videos in the supplementary.

Task	Success Rate	Seconds for completion (std dev.)	Description of task
Pickup Dice Toy	0.9	8.6 (2.65)	Pickup Plush dice from table.
Pickup Dinosaur Doll	0.9	8.2 (3.49)	Pickup Plush dinosaur from table.
Box Rotation	0.6	37.2 (12.6)	Rotate box 90 degrees onto the smaller side.
Scissor Pickup	0.7	28.6 (9.4)	Remove Scissors from the box using fingers
Cup Stack	0.6	21.5 (7.6)	Smaller green YCB cup must be placed inside the large blue cup.
Two Cup Stacking	0.7	27.3 (11.0)	Small orange YCB cup placed into green cup into a large blue cup.
Pouring Cubes onto Plate	0.7	36.8 (17.7)	Five 1” wooden cubes in a blue cup must be poured onto a plate.
Cup Into Plate	0.8	10.6 (4.4)	Place cup on the plate.
Open Drawer	0.9	23.6 (12.3)	Open clear drawer.
Open Drawer and Pickup Cup	0.6	33.7 (8.1)	Open clear drawer and pickup orange YCB cup inside.

Table 1. Success rate and completion of expert human pilot completing a variety of tasks, using our method.

joint angles to the robot arm controller.

This simple correspondence trick works surprisingly well in practice, and provides a natural user experience for the operator. To handle minor errors in human body pose estimation, we simply reject flagrant outliers, and maintain an exponentially decaying average of estimated wrist poses to serve as a lowpass filter. This ensures the commands we send the robot arm evolve smoothly over time. Smooth robot arm motion is crucial for safety and ease of teleoperation.

4. Experiments

We evaluate the strengths and limitations of our system through experiments on a diverse suite of dexterous manipulation tasks with an expert operator. We also demonstrate the usability and generality of the system through a smaller set of tasks on a group of ten previously untrained operators with videos in supplementary. Finally, we present ablation experiments that analyze two design decisions of our system.

4.1. Trained Operator Study

A trained operator attempted a diverse set of tasks to test the horizon of our system’s capabilities. These tasks are shown in Figure 6. They span a diverse spectrum of arm and hand motions and involved interacting with a variety of different objects. Each of the ten tasks was run for ten trials with a timeout period of one minute. This rigorously tested the system’s capabilities and limitations. These videos will be provided in the supplemental submission and the tasks are described in Table 1. The operator achieved good success on all tasks. Grasping plush objects proved easy as these grasps do not require much precision, but we observed that fine-grained grasps of smaller, more slippery objects like plastic cups occasionally proved difficult.

4.2. Human-Subject Study

To test usability and generality, we conduct a human-subject study in which 10 previously untrained operators

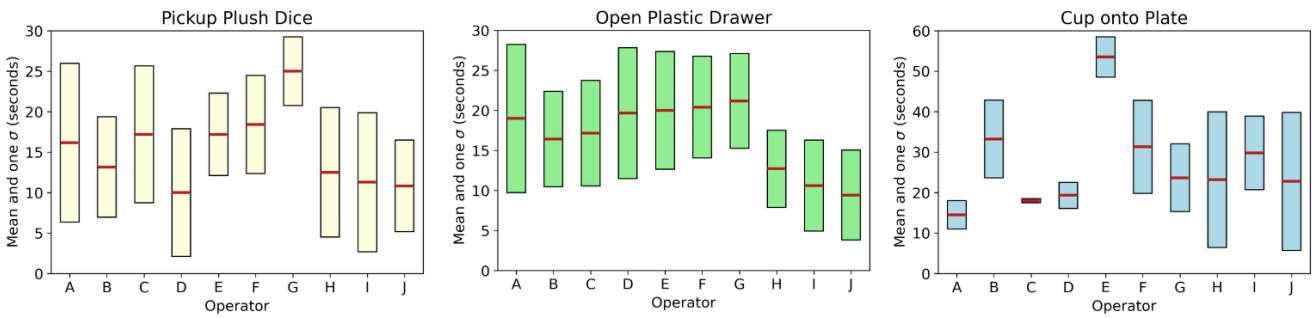

Figure 7. Ten rookie operators were asked to complete tasks: Picking up a plush dice, opening a plastic drawer, and placing a cup onto a plate. After seven tries at each task, the mean and standard deviation of their completion times were recorded per task.

each complete a set of 3 tasks 7 times. The first task is a plush dice pickup task (30 second timeout), the second is the drawer opening (30 second timeout), and last is the cup onto plate task (60 second timeout) with a small rest period in-between each set of seven. The total time for one human subject to learn about the system and complete tasks took approximately 15 minutes. Figure 7 reports the completion times of each operator on each of the three tasks.

The fluidity and quickness of the data collection process shows that the system can effectively collect demonstrations from humans. Although the underlying technology is complex, the user interface was easy to understand and use for all operators. Each operator differed in their style of motion, stances, and appearances, but there were no noticeable discrepancies in the behavior of the system or the distribution of results. This shows that the ability of system to work smoothly for a diverse group of operators.

Since the tasks were always completed in the same order, we found that subjects often struggled during the first few trials of the dice task. Some were initially confused by how the hand or arm moved, but all subjects were able to quickly adjust their motions. During the trials, many subjects expressed their compliments, but also their frustrations with the system. Our system was often complimented on its responsiveness and fluidity: subjects did not notice with any lag or jitter in the robot's imitation. Subjects enjoyed participating in the study, and some said felt that teleoperation the robot felt similar to a video-game. Additionally, subjects noted they felt safe and comfortable during teleoperation.

The largest frustrations with the system was in periodic errors in the retargeting of the human fingers to the Allegro robot hand. Many subjects noted instances when they were attempting complicated hand poses, but our system failed to accurately imitate them. In particular, we noticed systematic errors of our system in handling the flexion of the thumb. The shape and joint axes of the Allegro hand thumb is particularly different from that of the human thumb, and we suspect that our energy function does not place enough weight on accurate thumb retargeting. Some subjects observed that the system was worse at tracking their hand when it was all the way open with their palm parallel to the floor. We suspect this is due to the way the hand is oriented relative to where we placed the camera; when the palm is parallel to the floor and the camera is looking head-on, the entire hand projects to a single point, which can make hand pose estimation difficult.

4.3. Ablation: Online vs Offline Optimization

Here, we study the trade-offs between a hand retargeter that uses *online optimization* (via inference-time gradient descent) and a neural network retargeter that relies on *offline optimization* during training. We use the DexYCB dataset [8], which contains videos of 10 different human subjects grasping a variety of common objects. Each frame in each video is annotated with the ground-truth hand pose of the subject, and these poses are representative of the types of hand poses we expect to see during deployment of our system. For each video, we simulate the execution of both retargeters using the DexYCB subjects as our mock teleoperators. Each simulation lasts for as many timesteps as there are frames in a video (typically around 70), and at each timestep, the retargeter is given the ground-truth human hand pose, and a time budget of 40ms within which to output an Allegro joint angle vector. Notably, both retargeters are optimizing the same energy function: one does so online at deployment time while the other does so in an amortized fashion during training.

We consider a subset of 500 videos that depict right-handed grasps, and for each video, we store the outputs of both retargeters. It's difficult, however, to meaningfully assess the quality of the results, because we have no ground-truth Allegro joint angles against which to evaluate the outputs of the two retargeters. To circumvent this, we design a pseudo ground-truth oracle as follows. For each frame in the dataset, we pre-compute "ground-truth" Allegro joint angles by passing the known human hand pose to a gradient-based optimizer that is allowed an *infinite time budget* with which to minimize the energy function. We allow it to run to convergence, and our assumption is that the final output is as close to optimal as we can hope for. We then compute

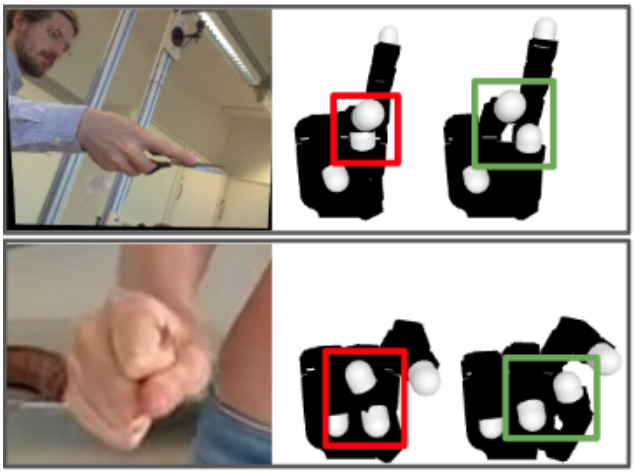

Figure 8. **The contribution of an adversarial self-collision loss**. We consider hand retargeting networks. The red boxes highlight instances where the vanilla network outputs Allegro hand poses that result in self-collision. The green boxes depict the predictions of the network trained with self-collision loss. These robot hand poses maintain functional similarity to the human's hand pose, but avoid self-collision.

the root mean squared error (RMSE) between the oracle's outputs and the outputs of each of our two methods.

The time-budgeted online optimizer achieves an RMSE of 0.25 radians per joint (about 14 degrees) over the entire dataset, and the neural network retargeter achieves an RMSE of 0.17 radians (about 10 degrees). Qualitatively, we observe that the online optimizer periodically gets stuck in local minima, and often remains stuck because each timestep's search is seeded from the final output of the previous timestep. The neural network makes errors too, but does not continue to suffer from errors in the past, because each output is independent of all previous outputs. Moreover, the neural network runs extremely fast (it takes around 3ms per timestep). Because each forward pass is so cheap, it's likely that ensembling several networks could further robustify the performance and eliminate outliers, while remaining comfortably within the time budget.

4.4. Ablation: Self-Collision Avoidance

Here, we study the effect of our adversarial self-collision loss. There are two questions we aim to answer. (1) As we increase the weight of the self-collision avoidance loss term, do we indeed produce fewer offending joint configurations? (2) As we increase this weighting term, how does our performance change with respect to minimizing the other terms in the energy function? We use a subset of 3000 randomly sampled frames from the FreiHand dataset [37], which contains a diverse set of human hand images, each of which is annotated with a ground truth hand pose. We consider 6 different hand retargeter networks, trained with collision-loss weights of 0, 0.2, 0.4, 0.6, 0.8 and 1. (A weight of 0.8, for example, means that the self-collision loss is weighed 0.8 as heavily as the sum of all the other key-vector matching loss terms in the energy function.) Each network makes predictions on each of the 3000 ground-truth human hand poses, and we compute (1) the fraction of resulting Allegro joint angle vectors that result in self-collision, and (2) the average value of the key-vector energy terms, over the dataset.

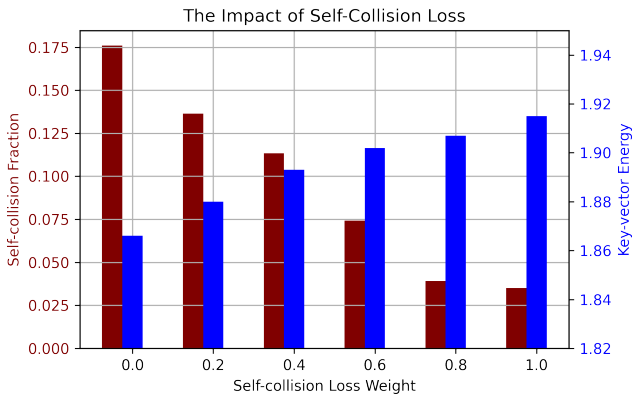

Figure 9. As we increase the weight of the adversarial self-collision loss, the hand retargeter network produces fewer self-colliding joint configurations (maroon), but incurs a higher energy with less similar poses (blue). A higher energy means that the predicted robot hand pose is dissimilar to the operator's hand pose.

We show example results in Figure 8 and plots in Figure 9. We observe a tradeoff between minimizing the prevalence of self-colliding joint configurations, but at the cost of increased key-vector dissimilarity. It is difficult to confidently assert that one is more valuable than the other, and in practice, we find that a middle ground works well.

5. Conclusion

We present a system for in-the-wild, real-time, remote visual teleoperation of a dexterous robotic hand and arm, which we call *Robotic Telekinesis*, where a human demonstrates tasks to the robot just by moving their own hands. We leverage the latest advancements in 3D human pose estimation and thousands of hours of raw day-to-day human footage on the internet to train a system that can understand human motion, and retarget it to corresponding robot actions. Our method requires only a single camera, and can be used out-of-the box for any operator on any task, without any actively collected robot training data. We show that our system enables experts and novices alike to successfully perform a number of different dexterous manipulation tasks. We hope that our system is used as a starting point for future research, rather than as an end product. We believe a powerful use of visual teleoperation is to bootstrap autonomous robot learning. By building an intuitive and low-cost platform for humans to provide task demonstrations, we hope to contribute to the democratization of robot learning.

References

[1] Cmu graphics lab motion capture database. http://mocap.cs.cmu.edu/. 2

[2] Haptx. https://haptx.com/. 1

[3] Dafni Antotsiou, Guillermo Garcia-Hernando, and Tae-Kyun Kim. Task-oriented hand motion retargeting for dexterous manipulation imitation. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0, 2018. 2

[4] Samarth Brahmbhatt, Cusuh Ham, Charles C. Kemp, and James Hays. ContactDB: Analyzing and predicting grasp contact via thermal imaging. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6 2019. 2

[5] Samuel R. Buss and Jin-Su Kim. Selectively damped least squares for inverse kinematics. *Journal of Graphics Tools*, 10(3):37–49, 2005. 5, 14

[6] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: realtime multi-person 2d pose estimation using part affinity fields. *IEEE transactions on pattern analysis and machine intelligence*, 43(1):172–186, 2019. 2, 11

[7] Zhe Cao, Ilija Radosavovic, Angjoo Kanazawa, and Jitendra Malik. Reconstructing hand-object interactions in the wild. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12417–12426, 2021. 2

[8] Yu-Wei Chao, Wei Yang, Yu Xiang, Pavlo Molchanov, Ankur Handa, Jonathan Tremblay, Yashraj S. Narang, Karl Van Wyk, Umar Iqbal, Stan Birchfield, Jan Kautz, and Dieter Fox. DexYCB: A benchmark for capturing hand grasping of objects. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 7

[9] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2020. 14

[10] Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Sanja Fidler, Antonino Furnari, Evangelos Kazakos, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, and Michael Wray. Scaling egocentric vision: The epic-kitchens dataset. In *European Conference on Computer Vision (ECCV)*, 2018. 2, 4

[11] Roger Davies. Technology versus terrorism. *Jane's International Defence Review*, pages 36–43, 2001. 1

[12] Yao Feng, Vasileios Choutas, Timo Bolkart, Dimitrios Tzionas, and Michael J Black. Collaborative regression of expressive bodies using moderation. *arXiv preprint arXiv:2105.05301*, 2021. 2

[13] Shreyas Hampali, Mahdi Rad, Markus Oberweger, and Vincent Lepetit. Honnotate: A method for 3d annotation of hand and object poses. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3196–3206, 2020. 2

[14] Ankur Handa, Karl Van Wyk, Wei Yang, Jacky Liang, Yu-Wei Chao, Qian Wan, Stan Birchfield, Nathan Ratliff, and Dieter Fox. Dexpilot: Vision-based teleoperation of dexterous robotic hand-arm system. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9164–9170, 2020. 2, 3, 4, 11

[15] Yana Hasson, Gül Varol, Dimitrios Tzionas, Igor Kalevatykh, Michael J. Black, Ivan Laptev, and Cordelia Schmid. Learning joint reconstruction of hands and manipulated objects. In *CVPR*, 2019. 2

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 11, 13

[17] Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3. 6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE transactions on pattern analysis and machine intelligence*, 36(7):1325–1339, 2013. 2

[18] Angjoo Kanazawa, Michael J. Black, David W. Jacobs, and Jitendra Malik. End-to-end recovery of human shape and pose. *CoRR*, abs/1712.06584, 2017. 2

[19] Shuang Li, Xiaojian Ma, Hongzhuo Liang, Michael Görner, Philipp Ruppel, Bin Fang, Fuchun Sun, and Jianwei Zhang. Vision-based teleoperation of shadow dexterous hand using end-to-end deep neural network. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 416–422. IEEE, 2019. 2

[20] Colin M Light, Paul H Chappell, and Peter J Kyberd. Establishing a standardized clinical assessment tool of pathologic and prosthetic hand function: normative data, reliability, and validity. *Archives of physical medicine and rehabilitation*, 83(6):776–783, 2002. 2

[21] Jia Liu, Fangxiaoyu Feng, Yuzuko C. Nakamura, and Nancy S. Pollard. A taxonomy of everyday grasps in action. *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 573–580, 2014. 2

[22] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J Black. Smpl: A skinned multi-person linear model. *ACM transactions on graphics (TOG)*, 34(6):1–16, 2015. 2

[23] Georgios Pavlakos, Vasileios Choutas, Nima Ghorbani, Timo Bolkart, Ahmed A. A. Osman, Dimitrios Tzionas, and Michael J. Black. Expressive body capture: 3D hands, face, and body from a single image. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 10975–10985, 2019. 2, 11

[24] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. *arXiv preprint arXiv:2004.00784*, 2020. 3

[25] Ilija Radosavovic, Xiaolong Wang, Lerrel Pinto, and Jitendra Malik. State-only imitation learning for dexterous manipulation. *arXiv preprint arXiv:2004.04650*, 2020. 1

[26] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017. 1

[27] Javier Romero, Dimitrios Tzionas, and Michael J. Black. Embodied hands: Modeling and capturing hands and bodies together. *ACM Transactions on Graphics, (Proc. SIGGRAPH Asia)*, 36(6), Nov. 2017. 2, 3

[28] Yu Rong, Takaaki Shiratori, and Hanbyul Joo. Frankmocap: A monocular 3d whole-body pose estimation system via regression and integration. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pages 1749–1759, October 2021. 2, 3, 4, 5, 11, 13, 14

[29] Dandan Shan, Jiaqi Geng, Michelle Shu, and David Fouhey. Understanding human hands in contact at internet scale. In *CVPR*, 2020. 2, 4, 13

[30] Omid Taheri, Nima Ghorbani, Michael J Black, and Dimitrios Tzionas. Grab: A dataset of whole-body human grasping of objects. In *European Conference on Computer Vision*, pages 581–600. Springer, 2020. 2

[31] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 13(04):376–380, 1991. 12

[32] Jean Vertut and Philippe Coiffet. Robot technology. vol. 3a. teleoperation and robotics: evolution and development. 1985. 1

[33] Ruben Villegas, Jimei Yang, Duygu Ceylan, and Honglak Lee. Neural kinematic networks for unsupervised motion retargetting. *CoRR*, abs/1804.05653, 2018. 2

[34] Jiayi Wang, Franziska Mueller, Florian Bernard, Suzanne Sorli, Oleksandr Sotnychenko, Neng Qian, Miguel A Otaduy, Dan Casas, and Christian Theobalt. Rgb2hands: real-time tracking of 3d hand interactions from monocular rgb video. *ACM Transactions on Graphics (TOG)*, 39(6):1–16, 2020. 2

[35] Kevin Zakka, Andy Zeng, Pete Florence, Jonathan Tompson, Jeannette Bohg, and Debidatta Dwibedi. Xirl: Cross-embodiment inverse reinforcement learning. *arXiv preprint arXiv:2106.03911*, 2021. 3

[36] Wenping Zhao, Jinxiang Chai, and Ying-Qing Xu. Combining marker-based mocap and rgb-d camera for acquiring high-fidelity hand motion data. In *Proceedings of the ACM SIGGRAPH/eurographics symposium on computer animation*, pages 33–42, 2012. 1

[37] Christian Zimmermann, Duygu Ceylan, Jimei Yang, Bryan Russell, Max Argus, and Thomas Brox. Freihand: A dataset for markerless capture of hand pose and shape from single rgb images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 813–822, 2019. 2, 4, 8, 13

A. Code and Reproducibility

Attached in the supplemental submission are a minimal set of files that contain the code for reproducing the human hand to robot hand retargeting network. We will release the remaining hardware code along with human-robot interface upon acceptance.

B. Hand/Body Bounding Box Detection

The first step in our retargeting pipeline is to detect two bounding boxes in an image of the human operator; one for the body and one for the right hand. These bounding boxes needn't be perfectly tight bounding boxes; it's more important that they contain the entire body/hand without truncations. We use an implementation of OpenPose [6] from the authors of FrankMocap [28]. First, a 2D body skeleton detector is run over the entire image, and outputs the predicted pixel locations for each of the 18 keypoints on the skeleton. The tight bounding rectangle around the points is then computed, and a fixed padding is applied on all sides to allow a margin of error.

The right hand bounding box is heuristically extracted based on the 2D body skeleton estimate. The bounding box is centered at the pixel corresponding to the right hand wrist, and the side length of the bounding box is conservatively chosen to ensure that the bounding box contains the entire hand. For an image of size 480x640, we use a side length of 150 pixels.

C. Hand Pose Estimation

The next step is to estimate the pose of the operator's right hand from a crop of the hand. The crop of the right hand is computed as described in the previous section, and is resized to a shape of 224x224. The crop is passed to a Convolutional Neural Network (CNN), which outputs a low-dimensional representation of the hand configuration. We use an implementation of the hand pose estimation network from [28]. This network uses a ResNet50 trunk [16], followed by a Multi-Layer Perceptron (MLP) regression head, which outputs three relevant parameters of the SMPL-X model [23]. (1) $\beta_h \in \mathbb{R}^{10}$ describes the *shape* of the hand (the dimensions of each finger and the palm), (2) $\theta_h \in \mathbb{R}^{45}$ describes the *pose* of the hand (how the fingers are arranged) and (3) $\phi_h \in \mathbb{R}^3$ describes the global orientation of the hand (how the hand's root coordinate frame is rotated in the image coordinate frame). The SMPL-X model maps the shape and pose parameters (β_h and θ_h) into a full 3D mesh of the hand, and the global orientation parameter ϕ_h transforms the coordinate frame of the mesh so the axes align with the axes of the image coordinate frame.

D. Human-to-Robot Hand Retargeting

Here, we describe two implementations of the human-to-robot hand retargeting module, one which uses online optimization (via inference-time gradient descent), and one which uses offline optimization (via a neural network). Both implementations take a human hand pose as input, and outputs joint angles for each of the 16 Allegro hand joints. The human hand pose is parameterized by the tuple (β_h, θ_h) as described in the previous section. The global orientation ϕ_h is not used in hand retargeting, because the Allegro hand has no wrist or palm joints, and therefore, matching the global orientation of the human hand is accomplished by the robot arm and not the robot hand. We use q_a to denote the vector of the 16 Allegro hand joint angles.

Human-to-Robot Hand Energy Function. Both implementations of the hand retargeting module minimize the same *energy function*, so we describe this first. Inspired by [14], the energy function aims to capture the functional similarity between a human hand pose and a robot hand pose. Five *keypoints* are defined on each hand: the index fingertip, the middle fingertip, the ring fingertip, the thumb fingertip, and the palm center. Each of these keypoints is associated with a coordinate frame, and the keypoint is the origin of the coordinate frame. Enumerating all pairs of keypoints yields ten *keyvectors*. Four of them are finger-to-palm keyvectors (index-to-palm, middle-to-palm, ring-to-palm and thumb-to-palm), three are inter-finger keyvectors (index-to-middle, index-to-ring and middle-to-ring), and three are finger-to-thumb keyvectors (index-to-thumb, middle-to-thumb and ring-to-thumb). Notably, each keyvector has one endpoint designated as the origin, and the other as the destination. The keyvectors are expressed in the coordinate basis of the origin keypoint's coordinate frame. We refer the reader to Figure 8 of [14], which elegantly depicts the keypoint coordinate frames and the keyvectors on both the human and Allegro hand.

The energy function between a human hand pose (parameterized by the tuple (β_h, θ_h)) and an Allegro hand pose (parameterized by the joint angles q_a) is computed as follows. First, for each $i \in \{1, \ldots, 10\}$, the i-th keyvector is computed on the human hand (call it $\mathbf{v_i^h}$) and the Allegro hand (call it $\mathbf{v_i^a}$). Then, each Allegro hand keyvector $\mathbf{v_i^a}$ is scaled by a constant c_i. The i-th term in the energy function is the Euclidean difference between $\mathbf{v_i^h}$ and $\mathbf{v_i^a}$:

$$E(\ (\beta_h, \theta_h),\ q_a\ ) = \sum_{i=1}^{10} ||\mathbf{v_i^h} - (c_i \cdot \mathbf{v_i^a})||_2^2 \qquad (3)$$

These scaling constants $\{c_i\}$ are hyperparameters that require some tuning. If the goal is to produce aesthetically appealing retargeted Allegro hand poses on generic hand gestures, one should set each c_i to around 0.625, in order

to account for the ratio in sizes between the average human hand and the Allegro hand. If the goal is to maximize functional similarity, in theory, one should set each c_i to 1, to encourage perfect matching of each keyvector. In practice, we find that setting the constants c_i to a value smaller than 1 is optimal for dexterous manipulation teleoperation. This is because in order to stably grasp an object, the fingers Allegro hand must exert forces pushing into the object. This is achieved by commanding the Allegro finger joints to positions that penetrate the object. These joint angles are never actually reached because the fingers end up colliding with the object, which is precisely the goal. For our experiments, we use a scaling constant of 0.8 for each of the finger-to-thumb and finger-to-finger keyvectors, and a scaling constant of 0.5 for each of the finger-to-palm keyvectors. This means that in order to ensure a stable grasp, operators must squeeze their fingers closer together than they normally would when grasping, but through our human subject study, we find that novice operators quickly realize this and naturally adjust.

Computing the keyvectors on the human hand. Having described what the keyvectors are and how they are used to define the energy function, we now describe how to compute the keyvectors on the human hand, given the SMPL-X model parameters (β_h, θ_h). The first step is to use the SMPL-X model to generate a full posed 3D mesh of the human hand. Given β_h and θ_h (and a template hand mesh), the SMPL-X model generates a 3D mesh that correctly captures the shape and pose of the human hand. The next step is to transform these vertices into a canonical coordinate frame, centered at the palm center, with the positive x axis pointing out of the hand, the positive y axis pointing toward the thumb, and the positive z axis pointing toward the middle fingertip. This is done by applying a hand-coded transformation between the SMPL-X coordinate frame and the canonical coordinate frame. The next step is to compute the transformation between each of the keypoint coordinate frames and the canonical coordinate frame. This is done using the Kabsch-Umeyama Algorithm [31] for estimating the transformation that best aligns corresponding pairs of points. Concretely, for each keypoint, we manually determine four vertices on the template hand mesh: (1) the keypoint itself, (2) a vertex located along 0.05m along the positive x axis from the keypoint, (3) a vertex located 0.05m along the positive y axis from the keypoint, and (4) a vertex located 0.05m along the positive z axis from the keypoint. This is pre-computed once, up front. At runtime, for a given posed human hand mesh, we gather the 3D coordinates for each of these three points in the canonical coordinate frame. We define a corresponding set of four points: $\{[0,0,0],[0.05,0,0],[0,0.5,0],[0,0,0.5]\}$, which denote the coordinates of these points in the coordinate frame of the keypoint. Given these four correspondences, the Kabsch-Umeyama computes the transformation between the keypoint coordinate frame and the canonical coordinate frame that best aligns these corresponding point pairs.

Computing the keyvectors on the Allegro hand. Here, we describe how to compute the keyvectors on the Allegro hand, given a vector of Allegro hand joint angles. The key idea here is to exploit forward kinematics. The URDF of the Allegro hand defines the kinematic skeleton of the Allegro hand. The forward kinematics map takes as input a joint angle vector and outputs the transformation between each link's coordinate frame and the root coordinate frame. Each of our keypoints conveniently corresponds to a particular link on the Allegro hand, so the keypoint coordinate frames can simply be read off from the forward kinematics result.

The energy function is differentiable. One critical point to note is that the forward kinematics map is a fully differentiable function of the Allegro hand joint angles. This is because forward kinematics is essentially a chain of sines, cosines and matrix multiplications. This is important because it means that the energy function is a fully differentiable function of the Allegro joint angles (by the Chain Rule). This is important because it allows us to compute the gradient of the energy function with respect to the Allegro joint angles, and use gradient descent to find the Allegro joint angles that minimize the energy, with respect to a given human hand pose. We now describe two ways to exploit this differentiability: via online gradient descent and via offline gradient descent.

Retargeting via Online Gradient Descent. We implement the online optimization retargeter by using Stochastic Gradient Descent (SGD) to minimize the energy function. At each iteration, we seed the Allegro joint angles to an initial value. At the first iteration, that initial value is the all-zero vector (which corresponds to an open palm with outstretched fingers). At all subsequent iterations, the seed vector is the result from the previous iteration. We then run a fixed number of SGD steps with a learning rate of 0.05. The number of gradient steps is a hyperparameter, and presents a tradeoff between accuracy and speed. Running more steps results in better convergence, but takes more time. We find 100 steps to be a good point on this spectrum.

Retargeting via Neural Networks. We implement the offline optimization retargeter with a neural network that takes as input a human hand pose parameterization $\text{concat}(\beta_h, \theta_h) \in \mathbb{R}^{55}$ and outputs a vector of Allegro joint angles $q_a \in \mathbb{R}^{16}$. We use a Multi Layer Perceptron (MLP) with three hidden layers of sizes 256, 256, 128, and intermediate tanh activations. We apply a tanh to the final output to

squeeze the values from $[-\infty, \infty]^{16}$ to $[-1, 1]^{16}$, and then scale the squeezed values to the appropriate values within each of the joint's ranges. (For example, joint 1 on the Allegro hand has a range of $[-0.196, 1.61]$ (in radians). If the raw output of the network for this joint were 1.23, the tanh operation would squeeze it to 0.84 and the rescale operation would rescale it to 1.47 radians, which is 0.84 of the way between -0.196 and 1.61.)

We train this network using a mixture of human hand poses from the Freihand Dataset [37] and "in-the-wild" human hand poses from the 100 Days of Hands dataset [29]. The Freihand dataset contains ground-truth SMPL-X shape and pose parameters for over 30,000 hand configurations, which we use as inputs to the network. The 100 Days of Hands dataset is simply a list of links to YouTube videos that depict humans using their hands for everyday tasks. In total, these span hundreds of millions of frames. We generate human hand poses by running our Hand Pose Estimation module (built on the CNN from [28]). Our final dataset consists of 30,000 samples from the FreiHand dataset and 30,000 randomly chosen samples from the 100 Days of Hands dataset.

E. Body Pose Estimation

The body pose estimation pipeline consists of two steps. The first is to estimate a rough body pose from a crop of the operator's body. The second is to refine the right-hand portion of the rough pose estimate by fusing in the more accurate hand pose estimate from the Hand Pose Estimation module.

Rough Body Pose Estimation via a CNN This step takes as input a crop of the operator's body, resized to a shape of 224 x 224. The crop is passed to a CNN, which outputs a low-dimensional representation of the body configuration. We use an implementation of the body pose estimation network from [28]. This network uses a ResNet50 trunk [16], followed by a Multi-Layer Perceptron (MLP) regression head, which outputs three relevant parameters of the SMPL-X model. (1) $\beta_b \in \mathbb{R}^{10}$ describes the body *shape* (the dimensions of the various body parts), (2) $\theta_b \in \mathbb{R}^{45}$ describes the *pose* of the body (how the limbs are arranged) and (3) $\phi_b \in \mathbb{R}^3$ describes the global orientation of the body (how the body's root coordinate frame is rotated in the image coordinate frame). The pose parameter θ_b is of shape $(24, 3, 3)$, and each of these 24 matrices denotes the 3 x 3 rotation matrix of a particular joint in the human body skeleton. The SMPL-X model maps the shape and pose parameters (β_b and θ_b) into a full 3D mesh of the body, and the global orientation parameter ϕ_b transforms the coordinate frame of the mesh so the axes align with the axes of the image coordinate frame.

Body-Pose Refinement via Hand Pose Integration. The body pose estimate obtained via the CNN can fail to capture the finer details of the hand pose, and crucially, can produce incorrect estimates for the rotation of the right-hand wrist relative to its parent in the human body kinematic chain. The Hand Pose Estimation module, however, operates on a zoomed-in crop of the operator's hand, and often produces much better estimates of the hand's global orientation. To exploit this fact, we use the "Copy-and-Paste" Body-Hand Integration module from [28], which refines the local orientation of the wrist based on ϕ_h, the global orientation of the hand estimated by the Hand Pose Estimation module.

F. Human-to-Robot Body Retargeting

We now describe how to map from a body pose estimate (β_b, θ_b) to a target pose for the xArm6's end-effector link, relative to its base link. The first step is to define two pairs of corresponding coordinate frames between the human and robot "bodies". The first pair is between the human torso and the robot "torso". We define the robot torso to be 25cm above the robot base frame. Both torso frames are oriented such that the positive x axis points out of the front of the torso, the positive y axis points towards the left side of the body, and the positive z axis points upwards toward the head. The second pair of coordinate frames is between the human's right hand wrist and the robot's wrist (i.e. end-effector). The wrist coordinate frames are centered at the wrist center, with the positive x axis oriented parallel to the vector originating at the palm center and pointing out of the front of the palm, the positive y axis pointing toward the thumb, and the positive z axis pointing toward the middle fingertip.

The problem of determining the relative transformation between the robot's end-effector and its base coordinate frame reduces to the problem of determining the relative transformation between the end-effector and the torso (because the torso coordinate frame is fixed relative to the robot's base frame). And this problem reduces to determining the relative transformation between the human's right hand wrist coordinate frame and the human's torso coordinate frame. In order to do this, we start at the torso joint, and traverse the human body kinematic chain (defined by the SMPL-X model) from the torso to the wrist, chaining rotations along the path.

G. xArm6 Inverse Kinematics Controller

The human-to-robot body retargeter module outputs target poses for the xArm6's end-effector, relative to it's base coordinate frame. The final step is to build a model that uses this target pose to send a steady and smooth stream of joint angle commands to the xArm6's default controller. In practice, we found that this module is crucial for perfor-

mance and must be carefully implemented with attention to details; if the robot arm does not move smoothly, dexterous manipulation tasks become impossible.

The first step is to handle outliers caused by erroneous body pose estimates. This is done by computing the difference between the arm's current end-effector pose and the end-effector pose output by the retargeting module. If the difference is greater than a threshold, it is clipped. The next step is to combine the (possibly clipped) end-effector pose target with a running Exponentially Moving Average (EMA) of end-effector poses. This helps ensure smooth motion in the presence of noise in the pose estimation and retargeting modules. The following update rule is used to update running average P_{EMA} to incorporate the new target pose P_{new}:

$$P_{\text{EMA}} = \alpha \cdot P_{\text{new}} + (1 - \alpha) \cdot P_{\text{EMA}} \tag{4}$$

We find $\alpha = 0.25$ to work well. We note that a lower value of α can introduce lag, but we find that because our system runs at such a high frequency, this is not an issue in practice.

The next step is to compute the difference between the robot's current end-effector pose, and the (newly updated) pose target, and to apply linear interpolation to divide that difference into equally spaced waypoints. Each waypoint end-effector pose is then passed to a Selectively Damped Least Squares (SDLS) Inverse Kinematics (IK) solver [5], implemented in PyBullet [9], which returns a vector of joint angles for the six joints in the xArm6. These joint angle commands are sent to the xArm6 servo controller.

H. Software Architecture

We now describe how we put together all of the aforementioned modules into a single system that efficiently retargets human motion to robot trajectories. We found it natural to design our system as a dataflow graph, with computation being done at the nodes, and inputs/outputs travelling along the edges. We first summarize the computation nodes we use, and then discuss how we optimized runtime performance by using parallel computation within a publisher-subscriber architecture.

The nodes in the dataflow graph. Each node corresponds roughly to one of the modules described in previous sections:

- **CameraNode**: captures RGB images of the operator at 30Hz.
- **HandBoundingBoxDetectorNode**: receives an operator image, and computes a bounding box of the right hand.
- **BodyBoundingBoxDetectorNode**: receives an operator image, and computes a bounding box of the body.
- **HandPoseEstimationNode**: receives a crop of the operator's right hand, and estimates the SMPL-X model parameters $(\beta_h, \theta_h, \phi_h)$ that parameterize the hand's shape, pose and global orientation.
- **BodyPoseEstimationNode**: receives a crop of the operator's body, and estimates the SMPL-X model parameters $(\beta_b, \theta_b, \phi_b)$ that parameterize the body's shape, pose and global orientation.
- **BodyHandIntegrationNode**: receives a hand pose estimate $(\beta_h, \theta_h, \phi_h)$ and a body pose estimate $(\beta_b, \theta_b, \phi_b)$, and computes a refined body pose estimate by using the Copy-and-Paste integration method from [28].
- **HandRetargetNode**: receives a hand pose estimate $(\beta_h, \theta_h, \phi_h)$, and computes the Allegro joint angles q_a that maximizes similarity with the operator's hand.
- **BodyRetargetNode**: receives a (refined) body pose estimate $(\beta_b, \theta_b, \phi_b)$, computes the relative transformation between the right hand wrist and the torso, and converts this to a target pose of the xArm6's end-effector link, relative to its base link.
- **AllegroHandControllerNode**: receives a target Allegro hand joint angle vector from the **HandRetargetNode**, interpolates the difference between robot's current joint angle values and the target into small fixed-size intervals, and sends a stream of interpolated joint angle commands to the robot's controller at a fixed frequency.
- **xArm6ControllerNode**: receives a target end-effector pose from the **BodyRetargetNode**, and commands a smoothly interpolated stream of xArm6 joint angle configurations to the robot's controller at a fixed frequency.

Optimizing performance via parallel computation. It is crucial that our system run as fast as possible, in order to ensure smooth robot motion and to avoid lagging behind the operator. Therefore, a key design decision was to opt for a parallel computation paradigm. The first implementation of our system sequentially chained together the various modules, and achieved a runtime of approximately 3Hz. In this squential implementation, the retargeting time was the sum of the time taken by each module. Our optimized implementation instead used the ROS publisher-subscriber architecture, with each node running on a separate process. Nodes pass inputs and outputs to each other via inter-process messages. With this approach, the retargeting time was determined only by the slowest node, and this achieved a runtime of approximately 25Hz, which greatly improves usability.

I. Hardware Architecture

Our setup consists of a Ufactory xArm6 robot arm mounted to a Vention table, with a Wonik Robotics Allegro Hand mounted as the end-effector. The Allegro Hand was upgraded with four 3D printed fingertips that are skinnier than the default tips. 3M TB641 grip tape is applied to the inner parts of the hand and around the fingertip which allows the Allegro Hand to better grip objects, as 3D printed components and the built in metal/plastic parts are slippery. One Intel Realsense D415 camera tracks the operator; we use only the RGB stream. In our experiments, the operator is standing near the robot, but this is not a requirement. The operator only needs to be able to see the robot, in order for them to adjust their movements to effectively complete tasks. In the future, we hope to enable this via internet webcams which would allow the operator to be located anywhere in the world. Running the system is a desktop system with an AMD Ryzen 3960x CPU, 128GB of RAM and two NVIDIA GeForce RTX 3080TI GPU's.

J. Human Subject Study Details

The 10 subjects that participated in the study were volunteer colleagues from the author's lab. A few were familiar with this project, but they were not intimately familiar with the details. Critically, they had never used the system before. The human subjects were assured that the data collected was anonymous, the robot never interacted with them in any way, and if they ever felt uncomfortable with the task for any reason they could terminate the experiment early. The act of collecting the data would fall under a Benign Behavioral Intervention: verbal, written responses, (including data entry or audiovisual recording) from adult subjects who prospectively agrees and the following is met: Recorded information cannot readily identify the subject (directly or indirectly/linked). This therefore gives an exemption for IRB approval. Example of this category are solving puzzles under various noise conditions, playing an economic game, being exposed to stimuli such as color, light or sound (at safe levels), performing cognitive tasks.

One author was the conductor of the study. The conductor briefed each subject on how to operate the system. The subjects were asked to stand and stay in frame of the camera during the duration of the experiments. They were asked to not move around too quickly as that would trigger safety limits of the control system, but this was never an issue. No other significant instructions were given. The conductor of the study also kept an emergency stop switch next to them for the safety of the robot system, but it was never used.

For the first few trials, many subjects were confused by the system but quickly adapted to it. The conductor instructed them to continually adapt to the system and try to complete the tasks without giving them additional information. The conductor took down notes on the compliments and complaints of the system while the system was being used.

The conductor only verbally told each subject the goal of the task but did not explain the best way to complete them. The tasks were very simple and intuitive so the subjects were not confused by them. For each task, a failure was recorded when either the time expired, the task became impossible to complete from the object state on the table, or the subject asked for a reset. Between each trial within each task, the subjects were asked to move the robot arm up away from the table to allow the conductor to reset the object. Between each task, the Telekinesis system was paused and subjects were allowed to rest their arm for a few minutes. The dice pickup task and drawer task was 30 seconds each for 7 trials. The last cup in plate task was 60 seconds long for each of the 7 trials. The total time to complete all three tasks was about 15 minutes.