A Videos

Task videos, performance videos, data collection and example of internet videos can be found at: https://videodex.github.io

B Additional Ablations

Comparing Effects of Actions, Visual and Physical Priors: Firstly, we ran an ablation where we pertained a policy on human videos performing the place task and finetune it on the uncover task (using robot data). Similarly, we pretrained a policy on Uncover and finetuned on place. The results are in the below table under `VideoDex-Transfer`. We see that for both tasks the performance degrades slightly, especially in the place task. We also train by adding noise to the demonstration trajectories, by adding two different levels of Gaussian noise with standard deviation being 0.01 and 0.05, shown as `VideoDex-Noise-0.01` and `VideoDex-Noise-0.05`. We find that adding more noise definitely hurts the performance of the method. We also train ResNet18 [58] features initialized from ImageNet [71] training instead of the R3M [6] features, and the results in `VideoDex-ImageNet`. We can see that performance drops off, which indicates that the visual priors are important. Note that all of the reported numbers are on test objects. We present the results in Table 6.

C Retargeting Details

We first retarget human videos from Epic-Kitchens [2]. Specifically, we use the new data (refresher) from their GoPro Hero 7 Black. We retarget video clips of humans completing tasks that are similar to the robot task. These clips are on average 5-10 seconds each, depending on the task.

Wrist in Camera frame The goal of Perspective-n-Point is to estimate the pose of the calibrated camera given a set of N 3D points in the world and their corresponding 2D point projections in the image. First the camera must be calibrated. To do this, we use COLMAP [63] on a set of videos. It tracks keypoints through frames and estimates the calibration from the internet videos. We find these camera intrinsics for the GoPro:

$$\begin{bmatrix} 2304.002572862 & 0 & 960 \\ 0 & 2304.002572862 & 540 \\ 0 & 0 & 1 \end{bmatrix}$$

Using this calibration, we can now complete the Perspective-n-Point process. We are given two sets of points, 16 points in 3D on the hand model in the model's frame $\begin{bmatrix} X_w, Y_w, Z_w, 1 \end{bmatrix}^t$, and another

Transforms	Description	Method
$M_{C_t}^{Wrist}$	Wrist in each Camera	FrankMocap + PnP
$M_{C_1}^{C_t}$	Track Moving Camera	IMU/ORBSLAM
$M_{World}^{C_1}$	Make Camera parallel to Ground	IMU/Stabilization Sensor
T_{Robot}^{World}	Rescale and Reorient for Robot	Heuristic
M_{Robot}^{Wrist}	$T_{Robot}^{World} \cdot M_{World}^{C_1} \cdot M_{C_1}^{C_t} \cdot M_{C_t}^{wrist}$	

Table 5: Transformations required to calculate wrist in robot frame from passive videos to use in learning. M denotes a transformation matrix, where T is a general transformation

.

Method/Task	Place	Uncover
`VideoDex-Noise-0.01`	0.55	0.87
`VideoDex-Noise-0.05`	0.50	0.60
`VideoDex-ImageNet`	0.40	0.62
`VideoDex-Transfer`	0.60	0.87
`VideoDex-Original`	0.70	0.90

Table 6: Ablations that compare effects of different action, visual and physical priors, as well as seeing how pretraining on different data transfers to other tasks.

set of 16 2D points in image frame $[\ u, v, 1,\ ]^t$:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

We use the OpenCV3 solvePnPRANSAC to complete this calculation. This implementation ensures that the process is resilient to erroneous detections.

Camera in First Camera frame In the SLAM section, the goal is to track the camera through the video. This is required to compensate for the movement of the camera. We use this on a selection of videos where we found the camera to move significantly.

We start the SLAM process two seconds before the action clip begins. We mark the start of the action's frame as the first frame, run it through SLAM and then recover the trajectory of the camera through the entire clip. Specifically we recover the transformation: $M_{C_1}^{C_t}$

The process of monocular SLAM is only valid up to a scale factor. Although Epic Kitchens has noisy accelerometer and gyro information from the camera's sensors, we do not use this data to disambiguate this scale factor throughout the duration of the video clip.

ORBSLAM3 [64] only evaluates real-time video going forward through time and does not recalculate prior poses from future information. While this seems imperfect for this purpose, we find that the results are satisfactory for our purpose. In our setup, we are using these retargeted videos as a prior for learning. These retargeted trajectories are not used directly on the robot so they do not need complete accuracy. The more important characteristic is speed. COLMAP [63] can take hours to process larger video clips, but ORBSLAM3 [64] can complete this process faster than real time. This enables us to use many videos as an action prior for the robot behavior.

Camera Parallel to Ground Now that we have the trajectory in the C_1 frame after SLAM and PnP, we still are missing some key transformations to get into the robot frame. First, the C_1 is not always upright compared to gravity, but the robot always is. If we have a vector normal to the ground either from the synthesized pseudo-depth map from the original Videodex method or privledged information from an accelerometer (accelerometer is not used in the original Videodex method) we can use:

$$\text{pitch} = \tan^{-1}(x_{Acc}/\sqrt{y_{Acc}^2 + z_{Acc}^2}) \tag{6}$$

$$\text{roll} = \tan^{-1}(y_{Acc}/\sqrt{x_{Acc}^2 + z_{Acc}^2}) \tag{7}$$

$$\text{theta} = \tan^{-1}(\sqrt{x_{Acc}^2 + y_{Acc}^2}/z_{Acc}) \tag{8}$$

The pitch and roll would be used to make the trajectory upright. The yaw is not something that is calculable this way because this rotation is around the z axis, or the direction of gravity so it isn't detected by an accelerometer. The theta represents how far the accelerometer z axis is off from upright but is not useful to reorient the frame.

Task	Robot Demos	Objects
Pick	125	8
Rotate	140	8
Open	120	4
Cover	124	12
Uncover	145	12
Place	175	10
Push	136	14

Table 7: Left: Number of trajectories we used for each task. Robot data is collected locally using teleportation. Most of these trajectories are 5-15 seconds in length and capture the motion trajectory of the task and visual data. Right: The number of different objects we used for each task's data collection. In our testing, we show generalization outside of this set of objects.

Accelerometer Robot Reorientation There's book-keeping transformations that must be included to rotate everything into the same frame conventions. Accelerometers, like the one in the GoPro have their frame where Z is up, y is into the screen from the lens, and x is to the left if you're looking at the screen. The camera frame has the x-axis pointing to the right from the screen point of view, the y-axis facing down, and the z-axis facing out of the lens. The robot frame has its x-axis facing out towards the table, the y-axis faces to the left from the robot point of view, and the z-axis points up. This then leads to the following results. The camera in world frame in roll, pitch, yaw using fixed axis is: $[pitch, 0, -roll]$. The world frame to robot frame rotation in roll, pitch, yaw using fixed axis is $[-3.14/2, 0, -3.14/2]$ This is used to rotate the trajectories to the robot frame and is the rotation component of T_{Robot}^{World}.

Rescaling for Robot We must fit the trajectories from the human videos into the robot frame. The robot frame has significant workspace limits that the human does not have. Even if the human arm is smaller than the robot's, the human can walk around whereas the robot arm cannot move from the middle of the table. We therefore center the trajectory and ensure it fits in the robot frame. This is the scaling portion of T_{Robot}^{World}.

We rescale each dimension of the arm trajectory as:

$$M_{World}^{Wrist_N} = M_{World}^{Wrist_N} - (\max(M_{World}^{Wrist_1..N}) + \min(M_{World}^{Wrist_1..N}))/2 + \text{robotWorkspaceCenter}$$

We would like to generate more similar trajectories to use in possible data augmentation. The naive method is to add gaussian noise to the trajectory. While this can be valid, it adds noise to an already noisy system. Instead we leverage the coordinate frames to create more accurate trajectories. We randomize the workspace scaling that is used by 10 percent. Additionally, we create a rotation M_{World}^{World} that rotates the initial world frame by up to 10 degrees in each fixed axis in roll pitch yaw convention. This perturbs the direction that the robot moves in its frame.

While this augmentation can be helpful with lower amounts of internet data, in our results it was not used as it led to similar results to not using data augmentation.

We interpolate the length of the trajectories using RBF basis functions. All trajectories from the internet data are rescaled to 200 datapoints. This uniformity enables efficient batch training and was used for all of the results.

Hand Re-targeting We use a similar approach to retargerting as Sivakumar et al. [56] and Handa et al. [72]. Specifically, we use the the detected human hand poses using MANO Romero et al. [31] (and FrankMocap Rong et al. [36]) to match 3D keypoints between human hands and the allegro hand. Given human hand parameters (β, θ), the goal is to minimize the difference between human

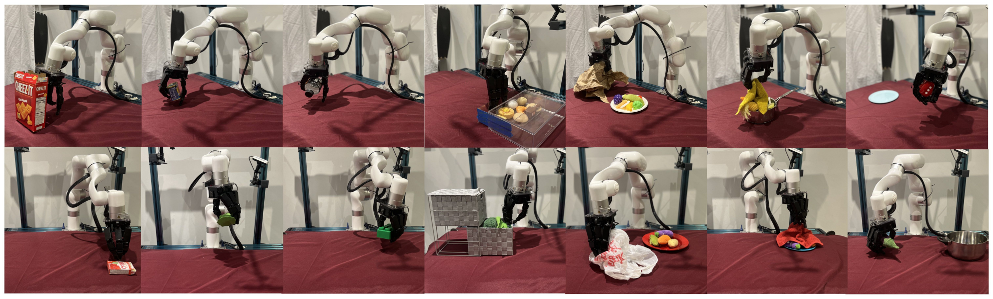

Figure 7: **Task Images**. A more detailed look at the tasks completed by VideoDex: push, pick, rotate, open, cover, uncover and place. and our website at `https://videodex.github.io` for further details.

	Pick		Rotate		Open		Cover		Uncover		Place		Push	
	train	test	train	test	train	test	train	test	train	test	train	test	train	test
`BC-NDP` [14]	0.64 ± 0.11	0.38 ± 0.13	0.94 ± 0.06	0.56 ± 0.13	0.90 ± 0.10	0.60 ± 0.16	0.78 ± 0.15	0.58 ± 0.15	0.88 ± 0.13	0.82 ± 0.12	0.70 ± 0.15	0.35 ± 0.11	1.00 ± 0.00	0.71 ± 0.13
`BC-Open`[51]	0.50 ± 0.12	0.44 ± 0.13	0.72 ± 0.11	0.38 ± 0.13	0.80 ± 0.13	0.40 ± 0.16	0.44 ± 0.18	0.58 ± 0.15	1.00 ± 0.00	0.91 ± 0.09	0.40 ± 0.16	0.25 ± 0.10	1.00 ± 0.00	0.93 ± 0.07
`BC-RNN` [51]	0.56 ± 0.12	0.31 ± 0.12	0.78 ± 0.10	0.50 ± 0.13	0.90 ± 0.10	0.50 ± 0.17	0.56 ± 0.18	0.42 ± 0.15	0.88 ± 0.13	0.75 ± 0.13	0.70 ± 0.15	0.50 ± 0.11	1.00 ± 0.00	1.00 ± 0.00
`VideoDex`	0.81 ± 0.09	0.75 ± 0.11	0.89 ± 0.08	0.69 ± 0.12	0.90 ± 0.10	0.80 ± 0.13	0.78 ± 0.15	0.67 ± 0.14	1.00 ± 0.00	0.90 ± 0.10	0.90 ± 0.10	0.70 ± 0.11	1.00 ± 0.00	1.00 ± 0.00

Table 8: We present the variance of train objects and test objects for Videodex and baselines described above. See the main paper for the mean results.

and robot keypoints: Human v_i^h and robot v_i^r. The robot keypoints are a function of robot joint pose: q. This is done by the implicit energy function (c_i are scale hyperparameters):

$$E_\pi(\ (\beta_h, \theta_h),\ q\ ) = \sum_i ||v_i^h - (c_i \cdot v_i^r)||_2^2 \tag{9}$$

This is inefficient to compute in real time, thus similarly to Sivakumar et al. [56], we distill this into a single neural network

$$f_{\text{hand}}((\beta_h, \theta_h)) = \hat{q}$$

This network learns to minimize the energy function E_π described above and is trained by observing internet videos. The hand retargeting setup can be seen in Figures 3 and 4 of the main paper.

Task

The tasks that were completed are Pick, Rotate, Open, Cover, Uncover, Place, Push. In pick, the task is to pickup objects off the table, or a plate/pan. Rotate involves turning an object in place. Open involves opening a drawer. Cover and uncover involve putting on or removing some form of cloth (dish, rubber, paper or plastic) on or from a plate. For place, the robot has to pickup an object and drop it in a plate or pot/pan. For push, the robot has to poke the object with its fingers. These tasks can be seen in Figure 7. We used videos from Epic Kitchens [2] that were as close as possible to these tasks, and doing similar types of actions. More details can be found in Table 9.

D Learning Pipeline Details

Learning Setup For our approach, we use the ResNet18 from R3M [6] weights as the visual backbone. This produces an intermediate feature vector of size 512. This is processed with a 2 layer MLP with a hidden dimension of 512. The visual features are concatenated with the starting hand and wrist pose. We employ two such MLPs, one for the hand and wrist trajectories. These are then processed with an NDP [13]. The NDP processes the input with a single hidden layer to project it into the desired size (parameters W and g). For more information we point the readers to Bahl et al. [13]. We use the implementation from Dasari et al. [51]. We use standard data augmentations from Pytorch. Specifically, we use RandomResizeCrop from a scale of 0.8 to 1.0. We use RandomGrayscale with a probability of 0.05 We use ColorJitter with a brightness of 0.4, contrast of 0.3, saturation of 0.3 and hue of 0.3. Finally, we normalize the RGB values around the typical mean and standard deviation for color images: $\mu = (0.485, 0.456, 0.406)$ $\sigma = (0.229, 0.224, 0.225)$. For different baselines, we used the same backbone (R3M [6]) as our method. We use the same architecture style as well, with the visual features being processed by both a wrist and hand stream. Finally, all network sizes are the same or very similar. We describe our hyperparameters in Table 9.

Parameter	Value
Learning Rate	1×10^{-3}
Batch Size	32
Training Demonstrations Per Task	120-175
Human Videos Per Task	350 (Cover/Unicver, Rotate, Push) - 2500 (Open, Pick, Place)
Trajectory Length Human Videos	200 (rescaled)
NDP [13] Basis Functions N	300
NDP [13] Global Parameter α	15

Table 9: Parameter List

E Experimental Setup

We collect data using a dexterous hand robotic teloperation setup [56, 72]. A trained operator stands in front of the camera within view of the robot and operates the system in real-time to collect demonstrations with a trained, uniform style. Another manager stands by. The goal of this manager is to place items on the table for manipulation, randomize locations and types of objects, to start and stop demonstrations for the operator and manage the robot system. We collect about 120-175 demonstrations per task. See table 5 for details.

F Hardware Details

Our hardware setup consists of an LEAP 16 DOF Hand and an XArm 6 manipulator (from Ufactory). The hand is mounted on the wrist of the XArm. To collect data we use a similar teleoperation system as provided by Sivakumar et al. [56] and Handa et al. [72]. We use Intel Realsense D415 cameras to collect human teleoperation and robot videos. We use four NVIDIA RTX 2080TI's for training the policy and running the teleoperated system. We experiment with the Allegro Hand and use it to collect some teleoperated demonstration data, but we find it to be very unreliable and break many times. The motors also quickly overheat and are weak in practice. Therefore, to alleviate these issues we use the LEAP Hand for collecting the final results.

External Codebases

We use the following different external codebases for our pipeline:

- Human body and hand detection: FrankMocap [36], `https://github.com/facebookresearch/frankmocap`
- NDP and Behavior Cloning [51] code from `https://github.com/AGI-Labs/robot_baselines`
- Code for R3M [6] `https://github.com/facebookresearch/r3m`
- CQL baseline from Takuma Seno [73] (`https://github.com/takuseno/d3rlpy`)
- COLMAP from Schönberger et al. [63] (`https://github.com/colmap/colmap`)
- ORBSLAM3 from Campos et al. [64] (`https://github.com/UZ-SLAMLab/ORB_SLAM3`)
- GoPro Metadata Extractor from (`https://github.com/JuanIrache/gpmf-extract`)
- Rigid Transform class from (`https://github.com/BerkeleyAutomation/autolab_core/blob/master/autolab_core/rigid_transformations.py`)
- PnP from (`https://opencv.org/`)