A. Method Details

In this section, we present more details of our method.

A.1. Algorithm

Algorithm 1 provides the psuedo-code of training our method. Algorithm 2 provides the psuedo-code of deploying our method on real robot during test time.

A.2. Data Collection Details

Our Robot-Free Data Collection is built around the 19-inch RMS Handi Grip Reacher and Intel RealSense D435 camera to collect visual data. We attach a 3D printed mount above the stick to hold the camera in place. At the base of the reacher-grabber, there is a lever to control the opening and closing of the gripper fingers. To collect demonstrations, a human user uses the setup shown in Fig 4 (a) which allows the user to easily push, grab and interact with everyday objects in an intuitive manner. We also use an Intel RealSense T265 camera to track the end-effector position via visual inertial odometry. The demonstrations are represented as a sequence of images I_t with corresponding end-effector positions P_t.

Once we have the end-effector pose P_t for every image I_t,we extract the relative transformation $T_{t,t+1} = P_t^{-1} \times P_{t+1}$ between consecutive frames and use them as the action for training.

A.3. Training Details

Our method is composed of two modules: 1) a pretrained representation network, R, to encode observations, $i_t = R(I_t)$, and enables control via distance learning. 2) a dynamics function F, to predict the future state for a possible action a_t.

Here we encode the image I_t via a ResNet18 [17] and use a 1-layer projection head to get the visual embedding $i_t \in R^{128}$. For the dynamics function $F(i_t, a_t)$, we use a 3-layer MLP (128 + action dimension to 128 to 128) with ReLU activation. Both modules are trained jointly with $\mathcal{L} = \lambda_d \mathcal{L}_d + \lambda_F \mathcal{L}_F$ where $\lambda_d = \lambda_F = 1$. We use the Adam optimizer [19] for training the network with a batch size of 64 and a learning rate of 10^{-3}. We train the network for 500 epochs and report the performance.

For each current observation, we randomly sample 4096 actions from training set as negative logits and use the ground truth action as possitive logit. For rotation-free tasks, like pushing and stacking, we use only the translation of $T_{t,t+1}$ as the action, such that $a_t \in R^3$. For rotation-heavy tasks, like knob turning, we use both translation and rotation of $T_{t,t+1}$ as the action, such that $a_t \in R^{12}$ (first three rows in the $SE(3)$ homogeneous transformation).

Algorithm 1 LMLS (Train of Passive Videos)

```
##################Initialize##################
R: observation encoder; F: dynamics function
G: gripper action classifier
###################Input#####################
I_t: current images; I_t+1: current images;
I_g: goal images; a_t: current actions;
a_r: sampled random actions; g_t: gripper action
#############################################
# Learning Task-Centric Distances
for x in loader: # load a minibatch x with N samples
    i_t = R(x.I_t) # encode current images
    i_t+1 = R(x.I_t+1) # encode next images
    i_g = R(x.I_g) # encode goal images
    i_p = F(i_t, x.a_t) # predict next state
    i_h = F(i_t, x.a_r) # hallucinated next state
    l_pos = cosine_similarity(i_p, i_g)#positive: N*1
    l_neg = cosine_similarity(i_h, i_g)#negative: N*M
    logits = cat([l_pos, l_neg], dim=1)#logits: Nx(1+K)
    labels = zeros(N) # contrastive loss
    loss_dis = CrossEntropyLoss(logits, labels)
    loss_dyn = MSELoss(i_p, i_t+1) # dynamics loss
    loss = loss_dis + loss_dyn
    loss.backward()
    update(R.params, F.params) # Adam update
# Learning Binary Gripper Classifier
for x in loader: # load a minibatch x with N samples
    i_t = R(x.I_t) # encode current images
    g = G(i_t) # predict gripper action
    loss = BCELoss(g, x.g_t)
    loss.backward() # Adam update
    update(R.params, F.params)
```

Algorithm 2 LMLS (Test on Robots)

```
##################Initialize##################
T_0: robot home position; I_0: initial observation
###################Input#####################
R: observation encoder; F: dynamics function
G: gripper action classifier; I_g: goal;
a_r: sampled random actions
#############################################
i_g = R(I_g)
While not reach_goal or t < max_step:
    i_h = F(R(I_t), a_r) # hallucinated next state
    distance = - cosine_similarity(i_h, i_g)
    # choose action leads to smallest distance-to-goal
    best_action_index = argmin(distance)
    a_t = a_r[best_action_index]
    g = G(I_t) # predict gripper action
    # Send command to robot and get new observation
    T_t+1, I_t+1 = Robot(T_t, a_t, g)
```

To improve the performance of our networks with limited data, we use the following data augmentations in training: (a) color jittering: randomly adds up to $\pm 20\%$ random noise to the brightness, contrast and saturation of each observation. (b) gray scale: we randomly convert image to grayscale with a probability of 0.05. (c) crop: images are randomly cropped to 224×224 from an original image of size 240×240.

B. Experiment Details

B.1. Hardware Setup and Control Stack

Our real-world experiments make use of a Franka Panda robot arm with all state logged at 50 Hz. Ob-

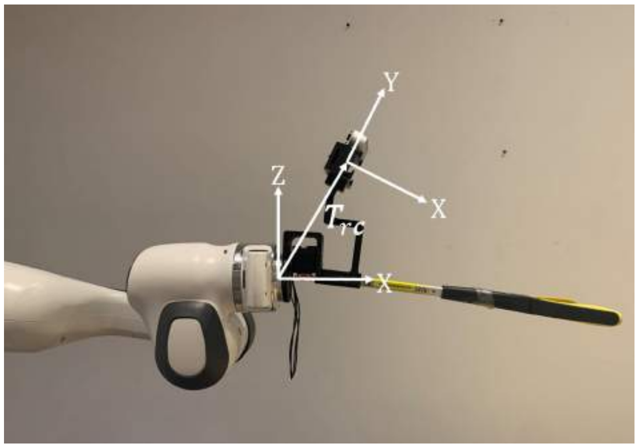

Figure 7: Transform actions in camera frame to robot frame.

servations are recorded from an Intel RealSense D435 camera, using RGB-only images at 1280×720 resolution, logged at 30 Hz. On the robot's end, we use the same 19-inch RMS Handi Grip Reacher and attach it using metal studs to the robot end effector through a 3D-printed mount. To control the fingers of the tool, we remove the lever at the base of the grip reacher and replace it with a dynamixel XM430-W350-R servo motor.

The learned visual-feedback policy operates at 5 Hz. On a GTX 1080 Ti GPU. The learned action space is a 6 Dof homogeneous transformation, from the previous end-effector pose to the new one. We then calculate the new joint position using inverse kinematics through Mujoco [45]. The joint positions are linearly interpolated from their 5 Hz rate to be 100 Hz setpoints to our joint level controller. The joint positions are then sent to Facebook Polymetis [26] to control the Franka robot.

It worth noticing the learned action space is in camera frame instead of robot frame. Thus, we need to transform the predicted actions $T_{c^0c^1}$ to robot frame through a fixed homogeneous transformation T_{cr} (Fig 7).

Using chain rule, we can easily calculate the motion in robot frame ($T_{r^0r^1}$) as:

$$T_{r^0r^1} = T_{rc} \times T_{c^0c^1} \times T_{cr} \tag{1}$$

$$= T_{cr}^{-1} \times T_{c^0c^1} \times T_{cr} \tag{2}$$

B.2. Baselines

We compare our method against three SOTA baselines: behavior cloning, implicit behavior cloning, implicit Q-learning. To make the comparisons fair, we parameterize all neural networks with the same R3M representation backbone used by our method, and tune hyper-parameters for best possible performance.

- **Behavior Cloning [35, 48] (BC):** BC learns a policy (via regression) that directly predicts actions from image observations: $\min_\pi ||\pi(I_t, I_g) - a_t||_2$. This provides a strong comparison point for a whole class of LfD methods that focus on learning motor policies directly (i.e. learn policies that predict actions). Here we encode the image I_t via a ResNet18 [17] and use a 4-layer multi-layer perceptron [34] to regress the actions (512-256-128-action dimension). The predicted actions are supervised with ground-truth actions via MSELoss. We use the Adam optimizer [19] for training the network with a batch size of 64 and a learning rate of 10^{-3}. We train the network for 200 epochs and report the performance.

- **Implicit Behavior Cloning [13] (IBC):** IBC learns an energy based model that can predict actions during test time via optimization: $a_t = argmin_a E(a, I_t)$. This method is conceptually very similar to behavior cloning, but has the potential to better handle multi-modal action distributions and discontinuous actions. Similarly, we encode the image I_t via a ResNet18 [17] and use a 1-layer projection head to get the visual embedding $i_t \in R^{128}$. We also encode the actions with a 3-layer multi-layer perceptron (action dimension to 32 to 64 to 128). For each current observation, we randomly sample 4096 actions $\hat{a}_j$ from training set as negative logits and use the ground truth action a_t as possitive logit. Both visual encoder and action encoder are trained with NCE loss:

$$\mathcal{L} = \frac{exp(cos(i_t, a_t))}{exp(cos(i_t, a_t)) + \Sigma_j exp(cos(i_t, \hat{a}_j))}$$

  We use the Adam optimizer [19] for training the network with a batch size of 64 and a learning rate of 10^{-3}. We train the network for 500 epochs and report the performance.

- **Implicit Q-Learning [20] (IQL):** IQL is an offline-RL baseline that learns a Q function $Q(s, a) = Q((I_t, I_g), a_t)$, alongside a policy that maximizes it $\pi(I_t, I_g) = argmax_a Q(s, a)$. Note that IQL's training process require us to annotate our offline trajectories $\mathcal{D}$ with a reward signal r_t for each time-step. Here we label the trajectories with sparse reward: +1 for end-effector reaching the target object, +2 for reaching the goal state, and +0 for all other states. We use d3rlpy [44] and trained the model for 500k steps.