The following Appendix sections are meant to add helpful context to our submission. In App. A, we provide a more formal definition of pre-grasps (see App. A.1), contrast them against grasps (see App. A.2), and show how prior dexterous, learning algorithms relied on pre-grasps for stable performance (see App. A.3). App. B presents all the hyper-parameters used in our task formulation (see App. B.1), TCDM benchmark (see App. B.2), and PGDM learning framework (see App. B.4). App. C provides a more thorough breakdown of our experimental validation results (see Sec. 5.1). Finally, App. D gives more information about the baseline implementations, and App. E discusses our hardware setup, built using the Franka Panda and D'Manus [2] robots.

A Pre-Grasps in Depth

In this appendix we give a more formal introduction to pre-grasps (see App. A.1), and compare them to grasps (see App. A.2), which the reader may be more familiar with. Finally, we demonstrate that pre-grasps are often leveraged as an unstated assumption in prior work (see App. A.3).

A.1 What are Pre-Grasps?

The pose of the hand (position, orientation, and joint articulation) just before the initiation of a hand-object interaction is called a "pre-grasp." It places the hand in a favourable region of the state space relative to the object, so that the intermittent contact behaviors of dexterous manipulation can evolve stably. Note that we did not develop this concept: it is a classical robotics construct [18, 19, 20].

We now outline a few key properties of pre-grasp states (shown in Fig. 7) – (1) First, a pre-grasp positions the robot close to the target object, and orients the robot's palm and wrist joints towards the object. This proximity ensures that pre-grasps can easily evolve into a stable grasp, without requiring the robot to explore the whole state space. (2) In addition, pre-grasp finger poses encode valuable information about functional parts of an object, without requiring the robot to reason about it explicitly. For example, a pre-grasp that curls a robot's fingers around a mug handle, offers a crucial signal for the robot to interact with the mug by grasping the handle. This property also implies that there might be multiple pre-grasps possible for every object (corresponding to different functions). (3) Finally, pre-grasp states incentivize favorable contacts (e.g. interaction with tool handles) and avoid dangerous contacts with the object (e.g. knife edge) and/or any other parts of the scene (e.g. pressing into the table). This is crucial because dexterous manipulation is full of contacts that are difficult to effectively model, predict, and reason about. A good pre-grasp provides a favourable start and strong momentum for learning the downstream manipulation behavior.

A.2 Grasp vs Pre-Grasp

Traditionally, successfully grasping an object is considered to be the most important event for dexterous manipulation. Indeed, dexterous manipulation has often been defined as a series of grasps in sequential order [36, 15, 6]. The first grasp in the sequence has been extensively analyzed both for grasp synthesis [36] and for object stability [15, 6]. Therefore, it begs an important question: are pre-grasps or grasps the key states for dexterous manipulation? We believe that pre-grasps are

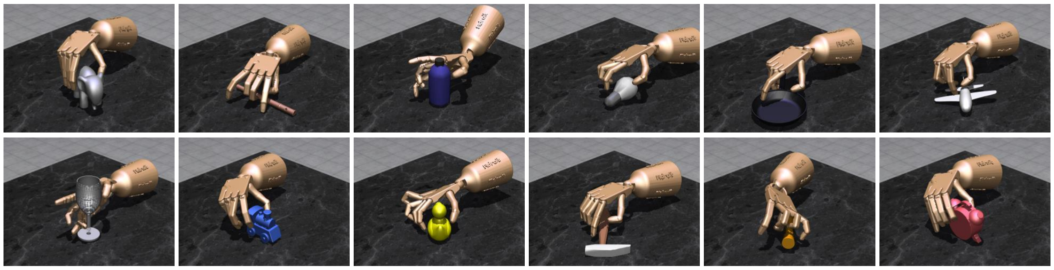


Figure 7: Examples of pre-grasps used in our experiments. Note how they (1) place the hand in proximity to the target object and (2) and pose the fingers around functional parts of the object (e.g. hammer's handle). However, pre-grasps are *not* stable grasps. In fact, the hand does not make contact with the objects at all!

Task Formulation Parameters		PGDM Framework Parameters	
Parameter	Value	Parameter	Value
λ_1	10	γ (discount)	0.95
λ_2	2.5	λ (GAE)	0.95
α	50	Learning Rate	1e-5
β	5	Value Fn. Coef.	0.5
ζ	0.02	Entropy Coef.	0.001
ω	0.25	PPO Clip	0.2
–	–	Steps per Iter	4096
–	–	Mini-Batch Size	256
–	–	Epochs	5
–	–	Policy Net ($Mean$)	MLP([256, 128]); TanH + Ortho. Init.
–	–	Policy Net (σ)	Param(size=*adim*, value=exp(-1.6))
–	–	Value Fn. Net	MLP([256, 128]); TanH + Ortho. Init.

Table 3: Experimental hyper-parameters for our task creation and policy learning stacks. Note that no per-task tuning was allowed at all for any of the listed parameters!

a stronger and more practical primitive to leverage when compared to grasps. They are stronger because if pre-grasps can simplify policy learning, then grasps should as well. However, the reverse is not true, since grasping comes after pre-grasps. Furthermore, they are more practical, since reaching a stable grasp is significantly harder than reaching a pre-grasp. After all, grasps are based on precise geometric and surface properties (friction, softness, etc) that cannot be accurately detected with modern position and force sensing technologies. Furthermore, grasps have extremely narrow stability margins that can be violated by small deviations.

Our key insight is that a grasp is a "post-condition" (effect) of gaining control of the object. However, to make dexterous manipulation policy learning feasible, we need a "pre-condition" (which is easier to reach than "post-condition") that can initialize the manipulation behavior in a healthy manner. We hypothesize that pre-grasps (i.e. hand pose before the onset of object interaction) are the key pre-condition, and arriving/initializing at a pre-grasp can significantly lower the complexity in synthesizing dexterous manipulation behaviors. Unlike grasps, pre-grasps– (1) are devoid of precise hand-object contacts and therefore doesn't have extreme sensing requirements; (2) are easy to achieve and satisfy as most robots have good joint position control; (3) have a wide stability basin such that various contact transitions needed for dexterous manipulation can evolve within its boundaries.

A.3 Pre-Grasps in Past Learning Work

We note that many dexterous manipulation papers in the last decade used pre-grasps to some degree without explicit mention. For example, the Pen-Task in DAPG's ADROIT suite [45] required pre-grasp initialization to work. Similarly, the OpenAI Rubik's cube experiment [3] assumed a stable in-hand (i.e. pre-grasp) initialization.

Two recent dexterous manipulation papers [11, 16] considered the in-hand object rotation task. For most of their experiments they start at a pre-grasp pose (object laid flat on the palm). As a test, we experimentally demonstrate for Huang et al. [16] this choice is crucial – removing the pre-grasp initialization causes learning to fail entirely. For Chen et al. [11] this choice is explicitly addressed in the paper – the only way they were able to avoid starting at pre-grasp was by using a *gravity curriculum*. Such a procedure would be impossible in the real world, where gravity is a fixed constant.

B Hyper-Parameters

In this appendix we list the hyper-parameters for our task formulation (see App. B.1), TCDM benchmark (see App. B.2), simulated environments (see App. B.3), and PGDM learning framework (see App. B.1).

B.1 Task Formulation Hyper-Parameters

As discussed in Sec. 3.1, our task formulation automatically parameterizes task MDPs using exemplar object trajectories. This is accomplished (in part) by using a simple reward function and termination function. The hyper-parameters for these functions are listed in Table 3 (left).

B.2 TCDM Tasks

As discussed in Sec. 4.1, we leverage our task formulation to automatically generate a benchmark – TCDM. Recall that each task in TCDM is parameterized using an exemplar object trajectory (more info in Sec. 3.1 and App. B.1). The trajectories for these tasks were mined from three sources listed below:

- **Human Motion Capture (GRAB [52]):** The GRAB data-set contains motion capture sequences of human-object interactions. Specifically, each recording in GRAB consists of a human performing various skills with a collection of common objects (e.g. use hammer, pass flute, lift duck, etc.) from the ContactDB dataset [7]. The recordings contain the target object's pose (i.e. position and orientation), and a mesh reconstruction of the articulated hand at each time-step. In other words, GRAB contains paired exemplar object trajectories (X from Sec. 3.1) and hand pose trajectories $H = [h_0, \ldots, h_n]$. Note that these hand trajectories are not used in our method, but are required for the baselines. We created 40 tasks using trajectories from GRAB.
- **Expert Policies (ADROIT [45]):** Another source of object trajectories are expert policies, acquired either by learning/writing a controller or through human expert tele-operation. Since expert policies produce object behaviors worth imitating, one can simply extract the object trajectory from successful policy roll-outs. Specifically, we take expert trajectories from the ADROIT benchmark suite, which were collected by tele-operating a simulated ShadowHand. ADROIT object trajectories (no actions!) were taken to parameterize 3 tasks.
- **Human Animators:** Finally, we showcase that object trajectories for our method need not come from data. Instead, suitable (i.e. smooth) object trajectories can be scripted by human animators. We manually animated 7 object trajectories and used them to create the final tasks.

In addition pre-grasps for each task are mined from the following sources:

- **MoCap:** We extract human hand poses from the GRAB MoCap dataset (discussed above) and transfer them to the robot using an inverse kinematics procedure (solve for joints that achieve human finger-tip positions). This allows us to easily create pre-grasps for trajectories sourced from MoCap datasets.
- **Tele-Op:** Expert Tele-Op trajectories provide a natural source for pre-grasps. We extract pre-grasp states from the dataset provided by the DAPG paper [45].
- **Human Labels:** These pre-grasps are manually labeled by a human annotator. This is accomplished using the MuJoCo visualizer UI.
- **Learned Model:** We feed the object mesh and pose into the GrabNet model [52], which in turn predicts a human grasp pose (w/ MANO parameters). This pose is transferred to the robot using inverse kinematics in the same fashion as the MoCap pre-grasps.

Altogether, we created 50 unique tasks using these trajectory sources, and bundled them together into the TCDM benchmark. However, the the baselines required extra supervision (i.e. full hand reconstruction) that was only present in the GRAB data-set. In addition, 10 of the tasks constructed using GRAB data focused on "simple" lifting behaviors that (while good for debugging) were not representative of free-form object motion. Thus, we created the TCDM-30 benchmark for our baseline study that (as the name suggests) consisted of the remaining 30 TCDM tasks parameterized w/ GRAB data. Table 4 lists all of the tasks, alongside the object trajectory source, pre-grasp sources, and PGDM performance. Please refer to our code release for benchmark membership (i.e. in TCDM-30 or not) for each task.

B.3 Implementing TCDM w/ a Physics Simulator

Prior sections (Sec. 3.1, 4.1 and App. B.1, B.4) have discussed how TCDM tasks are formulated, the hyper-parameters used, the trajectories that parameterize them, and the design decisions made while building them. In contrast, this section will describe how TCDM tasks are actually implemented.

Our investigation leveraged the MuJoCo [53] physics simulator in place of a real robotics setup. Simulation was used since dexterous hardware is hard to acquire, and because real setups are less reproducible. For each task, we built a simulated scene that contains a table, a single robot, and a single target object. Object matching was done to explicitly pair exemplar trajectories with the intended object (e.g. drinking trajectory is matched w/ mug), while the robot matching was done arbitrarily. We consider 3 robot morphologies – ShadowHand [23], D'Hand [13], and D'Manus [2] – alongside 34 objects from the ContactDB [7] and YCB [9] object sets. Note that the simulated object properties are carefully defined to avoid inconsistencies. For the ContactDB objects, we infer object properties (e.g. mass and moments of inertia) by adopting common properties from 3D printed objects (ContactDB objects are printed). This was achieved by setting the object density to 1.25 g/cm^3 (density of PLA) and then using MuJoCo to solve necessary object properties from the convex decomposition of the objects. The convex decomposition was solved using VHACD [1]. The YCB object properties are defined by measuring real world versions of the objects.

We've now described the simulated setup and MDP formulation for every TCDM task. These are implemented together into a single Gym environment [8]. As a result, our tasks can be easily "plugged" into other RL and control code-bases. We now describe the observation space and action space used in our environment:

- **Observation Space:** We use a simple state space consisting of the robot joints, robot fingertip locations, and the object pose. In addition, we use positional encoding [55] to mark the current simulation time-step and add that to the end of the state vector. This is needed to handle time varying goal behaviors (see Sec. 3.1).
- **Action Space:** Our action space is joint position control, achieved using a simple low level PD torque controller. The first six joints handle wrist position/orientation, and the remaining joints control the robot fingers themselves. The simulated gains are tuned to be realistic, and gravity compensation is applied (by modifying applied torques). These assumptions are all consistent with real dexterous hand hardware.

Please check our website for the code and further documentation of TCDM tasks and environments: https://pregrasps.github.io/.

B.4 Learning Behaviors w/ PGDM

As discussed in Sec. 4.2, our PGDM framework uses pre-grasp states as exploration primitives to speed up dexterous behavior learning. It operates in two phases: (1) the robot hand is brought to a pre-grasp state using a heuristic trajectory optimization planner; and (2) the behavior policy is learned by optimizing task reward (e.g. from TCDM) using a RL algorithm. Both ingredients are described in detail below. Furthermore, PGDM hyper-parameters are listed in Table. 3 (right), and psuedo-code is shown in Alg. 1.

Reaching Pre-Grasp States: After the task scene is reset (e.g. object brought to reset position), PGDM's first stage begins. Specifically, the robot hand is brought to an appropriate pre-grasp position for the task and target object. Note that the pre-grasp state is not predicted, instead it is manually annotated (per-task) in an offline dataset. A CEM trajectory planner [31] solves for actions that bring the robot to the pre-grasp, without disturbing the object. This is done in a geometry-free way (i.e. w/out any object mesh knowledge). First, the hand is brought far above (e.g. 30cm) the target object. Second, the thumb joint begins moving towards its pre-grasp configuration. And finally, the rest of the hand is brought to the pre-grasp state. While a simple heuristic, this motion plan successfully brings the robot to pre-grasps for *all* of the 50 diverse tasks (and 34 objects) considered in TCDM, without *any* scene/geometry information. This will not scale to more complicated settings (e.g. reaching pre-grasp states in clutter), but those situations are beyond the scope of this paper.

Behavior Learning w/ Pre-Grasps: Once at a pre-grasp state, PGDM swaps to a policy learning algorithm, to optimize the original task reward. Specifically, we utilize the PPO [49] policy-gradient

Algorithm 1 Learning TCDM tasks w/ PGDM

```
####################### INPUTS ###########################
# policy: Randomly initialized policy network ($\pi$)
# CEM: Trajectory Planner for reset policy
# PPO: Implementation of PPO algorithm (e.g. SB3)
# goals, pre_grasp: exemplar trajectory and pregrasp state
###################### CONSTANTS #########################
# N_ITERS, N_STEPS: PPO iterations/rollout buf size
# L_1, L_2, A, B, Z: constants for Eqn. (1)
# gamma: termination bound for Eqn. (2)
##########################################################

class TCDMEnv(gym.Env)
   .....

   # reward helper function
   def reward_fn(self)
      state = self.get_state()
      t = state.time
      g = self.goal[t]

      # calculate error terms
      delta_xyz = l2error(state.obj_xyz - g.obj_xyz)
      delta_ori = absang(state.obj_ori, g.obj_ori)

      # calculate reward with Eqn. (1)
      R = L_1 * exp(-A * delta_xyz - B * delta_ori)
      if next_s.obj_z > Z and g.obj_z > Z:
         R += L_2
      return R

   # termination helper function
   def termination_fn(self):
      state = self.get_state()
      t = state.time

      # terminate if error exceeds threshold
      delta_xyz = l2error(state.obj_xyz - g.obj_xyz)
      return delta_xyz > gamma


# helper function for rolling out policy
def get_rollouts(policy, n_steps):
   # aggregates data from the rollouts
   buffer = RolloutBuffer()

   while len(buffer) < n_steps:
      # reset environment and move robot hand to pre_grasp
      s = env.reset()
      reset_policy.moveto(env, pre_grasp)

      while True:
         # get action from policy and step environment
         a = policy.sample(s, g)
         next_s, r, done, info = env.step(a)

         # append to buffer and advance state
         buffer.append(s, a, next_s, R)
         s = next_s

         # reset episode if done
         if done:
            break
   return buffer


# load task and reset planner
env = TCDMEnv(*)
reset_policy = CEM()

# train policy with simple PPO update loop
env = TCDMEnv(*)
for i in range(N):
   eval_and_log(policy)
   buffer = get_rollouts(policy, N_STEPS)
   PPO.update(policy, buffer)
```

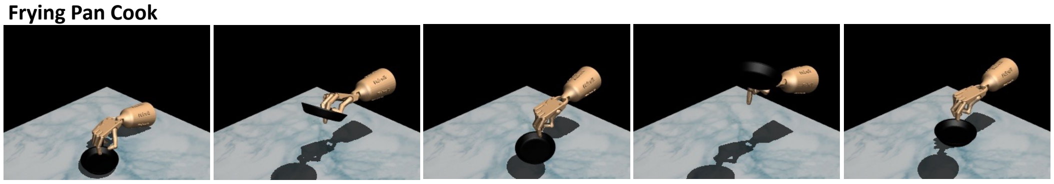
`l2error`: euclidean distance; `absang`: mag. of quat. angle; `exp`: natural exponent.

RL algorithm for policy learning. Our code uses the Stable-Baselines3 [44] implementation that is built upon the Pytorch [40] deep learning library. Please check Table. 3 for the exact RL hyper-parameters.

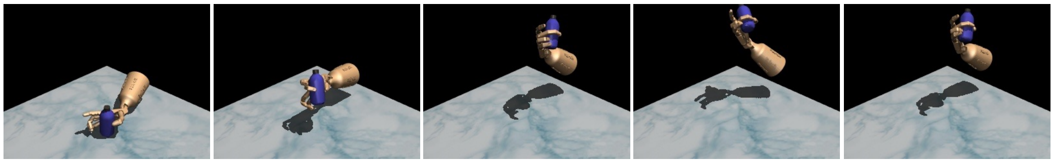
Task	Pre-Grasp Source	Trajectory Source	Success	Error (m)
airplane-fly1	MoCap	MoCap [52]	71.7%	6.66e-03
airplane-pass1	MoCap	MoCap [52]	60.7%	2.69e-03
alarmclock-lift	MoCap	MoCap [52]	41.8%	7.39e-03
alarmclock-see1	Learned	MoCap [52]	85.7%	1.39e-03
banana-pass1	MoCap	MoCap [52]	99.7%	4.11e-04
binoculars-pass1	MoCap	MoCap [52]	74.3%	2.35e-03
cup-drink1	MoCap	MoCap [52]	97.6%	3.66e-04
cup-pour1	MoCap	MoCap [52]	83.1%	8.80e-04
dhand-alarmclock	Labeled	Animator	84.8%	8.85e-04
dhand-binoculars	Labeled	Animator	13.7%	3.80e-02
dhand-cup	Labeled	Animator	85.5%	3.13e-03
dhand-elephant	Labeled	Animator	10.0%	4.19e-02
dhand-waterbottle	Labeled	Animator	70.1%	2.48e-03
dmanus-coffeecan	Labeled	Animator	62.9%	1.46e-03
dmanus-crackerbox	Labeled	Animator	98.0%	6.28e-04
door-open	Tele-Op	Expert [45]	92.2%	5.02e-04
duck-inspect1	Learned	MoCap [52]	99.3%	3.94e-04
duck-lift	MoCap	MoCap [52]	97.2%	3.00e-04
elephant-pass1	MoCap	MoCap [52]	50.5%	1.58e-02
eyeglasses-pass1	MoCap	MoCap [52]	39.4%	1.73e-02
flashlight-lift	MoCap	MoCap [52]	97.1%	6.53e-04
flashlight-on2	MoCap	MoCap [52]	94.9%	5.13e-04
flute-pass1	MoCap	MoCap [52]	65.3%	7.71e-03
fryingpan-cook2	MoCap	MoCap [52]	98.7%	3.75e-04
hammer-strike	Tele-Op	Expert [45]	64.0%	2.50e-03
hammer-use1	MoCap	MoCap [52]	99.3%	4.11e-04
hand-inspect1	MoCap	MoCap [52]	97.1%	1.02e-03
headphones-pass1	MoCap	MoCap [52]	55.3%	1.14e-02
knife-chop1	MoCap	MoCap [52]	93.4%	7.30e-04
lightbulb-pass1	MoCap	MoCap [52]	43.7%	3.13e-02
mouse-lift	MoCap	MoCap [52]	57.2%	6.39e-03
mouse-use1	MoCap	MoCap [52]	21.7%	2.82e-03
mug-drink3	MoCap	MoCap [52]	35.4%	1.39e-02
piggybank-use1	MoCap	MoCap [52]	1.6%	6.06e-03
scissors-use1	MoCap	MoCap [52]	87.4%	8.12e-04
spheremedium-lift	MoCap	MoCap [52]	100.0%	1.61e-04
spheremedium-relocate	Tele-Op	Expert [45]	97.3%	3.68e-04
stamp-stamp1	MoCap	MoCap [52]	44.9%	1.18e-02
stanfordbunny-inspect1	Learned	MoCap [52]	86.2%	2.88e-03
stapler-lift	MoCap	MoCap [52]	60.8%	5.16e-03
toothbrush-lift	MoCap	MoCap [52]	100.0%	3.03e-04
toothpaste-lift	MoCap	MoCap [52]	100.0%	2.80e-04
toruslarge-inspect1	MoCap	MoCap [52]	70.7%	5.15e-03
train-play1	MoCap	MoCap [52]	99.5%	2.78e-04
watch-lift	MoCap	MoCap [52]	97.8%	5.99e-04
waterbottle-lift	MoCap	MoCap [52]	51.2%	1.74e-03
waterbottle-shake1	MoCap	MoCap [52]	92.4%	1.03e-03
wineglass-drink1	MoCap	MoCap [52]	99.8%	1.82e-04
wineglass-drink2	MoCap	MoCap [52]	100.0%	1.94e-04
wineglass-toast1	MoCap	MoCap [52]	92.1%	1.04e-03

Table 4: This table lists the 50 tasks in TCDM, along with the pre-grasp trajectory sources used to parameterize them. In addition, we list the success and error metrics at the end of training (achieved by our PGDM method) for each task, averaged across 3 random seeds.

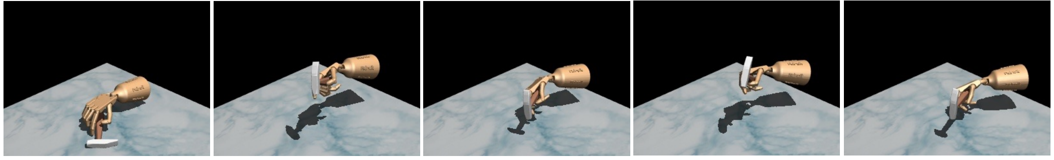
Frying Pan Cook



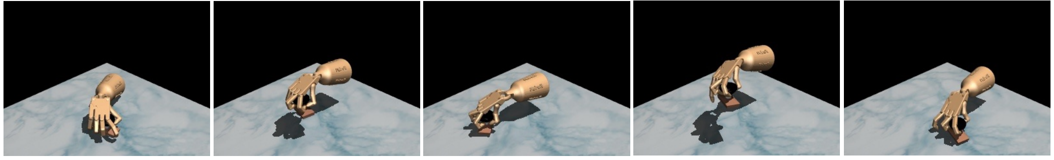
Water Bottle Shake



Hammer Striking



Stamping



Lifting Duck

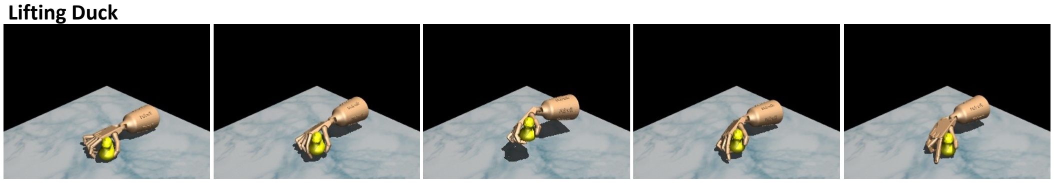


Figure 8: Qualitative results of our learned policies on select tasks from TCDM. Note how TCDM contains diverse objects with behaviors ranging from shaking, to hammering, to lifting. Furthermore, our policies learn natural "human like" actions, despite never optimizing for them. For animated visualizations please refer to our website: https://pregrasps.github.io/.

C PGDM Validation Experiment Breakdown

This appendix presents additional context and results from the full validation experiment that could not fit in the main paper. Recall (from Sec. 5.1) that we deployed our PGDM framework on all 50 tasks in TCDM using 3 random seeds per task. Performance by task is broken down in Table 4, and learning curves are presented in Fig. 9. Additionally, qualitative examples of the behaviors are shown in Fig. 8. Note the diversity of our task suite, and how PGDM works across a wide variety of scenarios, with no fine-tuning and minimal variance between seeds. For animated visualizations, please check our website: https://pregrasps.github.io/.

D Baseline Implementations

This appendix provides added context for the DeepMimic (see App. D.1), GRAFF* (see App. D.2), and Task Curriculum (see App. D.3) methods used as baselines in our experiments (see Sec. 5.2). Note that these baselines require added hand supervision that is only found in the GRAB data-set (see App. B.2). As a result, our baselines are run on only a subset of tasks (i.e. TCDM-30).

D.1 DeepMimic

DeepMimic [41] uses additional rewards to supervise the hand motion in addition to the object motion. Specifically, given an expert hand trajectory – $H = [h_0, \ldots, h_T]$ where h_i is a hand-pose (i.e. joint

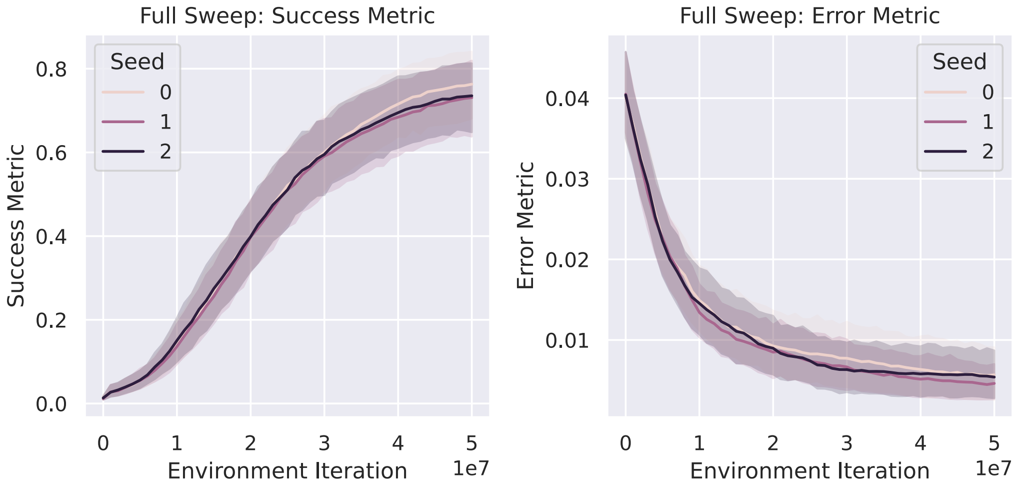


Figure 9: These learning curves show how PGDM's error and success metrics (averaged across all TCDM tasks) evolve over RL training. Our experiments were run with three seeds, each plotted separately. Note how PGDM effectively learns the tasks with low run to run variance.

angles) at step t – and a matching exemplar object trajectory X (see Sec. 3.1), the robot is trained to match the expert trajectory and exemplar object trajectory at the same time. This is done with added (in addition to TCDM-30) reward and termination condition, $\lambda_3 exp(-\eta||h_t - \hat{h}_t||_2)$ and $||h_t - \hat{h}_t||_2 > \gamma_h$ respectively, where $\hat{h}_t$ is the achieved robot pose at step t. We set $\lambda_3 = 1, \eta = 15, \gamma_h = 0.1$, since that gave best performance. Note that the original DeepMimic implementation used additional reward terms, which we dropped since they didn't change performance in our setting. Additionally, H is calculated by applying inverse kinematics to the original hand pose trajectories from GRAB.

D.2 GRAFF*

The original GRAFF [32] baselines predicted contact affordance regions on objects and encouraged the robot to interact with them. However, this original implementation used visual observations, whereas our setup uses robot state information. To make comparison fair, we extracted ground truth contact locations on the object from the GRAB dataset. In other words, we found where the human expert made contact with the object during MoCap recording, and thus created an optimal contact pattern for the robot to match. We then added a reward that incentivizes the robot to move its fingertips to those contact locations – $\lambda exp(-\eta||c_t - \hat{f}_t||_2)$ where c_t is desired contact at step t and $\hat{f}_t$ is robot fingertips. We found $\lambda = 1, \eta = 15$ gave best performance. This reward was added on top of the standard (e.g. TCDM-30) reward. To summarize, we re-implemented GRAFF using state-only observations, and made the comparison fair by adding expert (e.g. optimal fingertip location) data.

D.3 Task Curriculum

The task curriculum is our simplest baseline. We start by training the robot to only lift the object (i.e. satisfy the lifting bonus from Sec. 3.1). The hope is that the robot will be able to easily learn the desired behavior (e.g. toast wineglass) once it knows how to lift the target object. Thus, the object matching losses are activated over the course of training. This can be done by linearly blending λ_1 (see Sec. 3.1) from 0 to 1 over 4 million environment steps. This induces a curriculum into the TCDM tasks.

E Real World

This appendix describes our real world verification experiment (see Sec. 5.1) in further depth. The goal here is to demonstrate that actions predicted by PGDM policies can be deployed on real hardware.

We do **not** consider training PGDM policies in the real world, nor do we fully transfer closed-loop control policies from sim to the real world.

Task Setup Our task is to use a D'Manus robot, which is a low-cost hand from RoBEL suite [2], to lift the Cheez-Itz cracker-box object from the YCB object suite [9]. A suitable task automatically defined using our MDP formulation (see Sec. 3.1). Specifically, we use a human animated "object lifting" exemplar trajectory, which lifts and holds the box 0.2m off the table, to define the task. In addition, we annotate an appropriate pre-grasp state, with the object positioned between the D'Manus outstretched fingers and thumb (see Fig. 6), for the task.

Simulated Policy Learning After defining the task, we created a simulated environment for it like normal (see App. B.3), and learn a behavior policy to solve it using PGDM. The learning procedure is almost exactly the same as our other simulated experiments, except for the addition of *domain randomization*. Specifically, at every environment reset the cracker-box's width, mass, and friction are randomized within a pre-specified range. This is done to make the policy robust across a range of realistic possibilities, since the physical cracker-box will not perfectly match its simulated counterpart.

Real World Playback After training the policy, we extract roll-outs from it in simulation and execute the actions taken in sim on a matching real world setup. Specifically, our real world setup uses a tabletop scene, with a YCB cracker-box target object, and a D'Manus robot mounted on a Franka Panda arm (see Fig. 6). For the D'Manus hand action playback happens in a straightforward manner: the learned actions are are simply replayed on the real world hardware using a PD controller. However, the base joints are modelled in sim using a free-joint with attached $x, y, z, \theta_x, \theta_y, \theta_z$ degrees of freedom. While this is suitable for sim, it is not achievable in the real world. To overcome this issue, we extract a base joint trajectory (e.g. the commanded base positions over time), convert it into Franka joint space using inverse kinematics [54], and play those joints back on the real Franka robot using a PD controller from Polymetis [28]. The Franka actions are played in real-time alongside the D'Manus actions, resulting in smooth action playback for both the base and fingers.

Results As discussed in Sec. 5.1, we are able to successfully lift the cracker-box object using the actions optimized in simulation. No real world adaptation or fine-tuning was required to make this possible (videos on our website[5]). This suggests that our simulated results are plausible in the real world. In addition, we demonstrate that the learned actions are not too aggressive for robot hardware, despite the lack of any additional human supervision. However, this result does not prove that our actual policies can be transferred onto hardware. Achieving real world transfer will require significant engineering outside the scope of this paper (see Sec. 7).

---

[5] https://pregrasps.github.io/