

A Algorithm Details

A.1 Simulated Robot

Although our main experiments are conducted entirely in the real world, without any simulation, we also constructed a simulated environment to systematically evaluate and compare various algorithmic choices and ablations of our method. We construct a simulation of our training setup using the PyBullet physics simulator. The workspace is a 3-by-3 meter room with flat walls and wooden floors. The objects consist of 20 green spheres scattered randomly across the room without obstacles, or 30 spheres in the room with obstacles. We evaluate our method both with and without obstacles (shown in Figure 3 and 8 (right)). The simulated robot model is replicated to the real world and simulates how the real robot operates. During training, the environment is not reset and the same autonomous pseudo-resetting mechanism is used during training. During evaluation, the agent uses a greedy argmax policy to choose actions.

A.2 Model Architectures

For the navigation policy, the observations are 100×100 RGB images from the front facing camera, which goes into three convolutional layers of sizes 64, kernel sizes 3, and stride 1, with an average pooling layer of size 2 in between each layer. The result is passed through two dense layers of size 512. The grasping policy uses the same observations, but center cropped to 60×60 to remove unreachable regions from the image. Each of the $N = 6$ networks in the grasping ensemble is also structured as three convolutional layers of size 64, kernel size 3, and stride 2, then two dense layers of size 512. In all experiments, we discretize the grasp action space into a 15×15 grid and use $\alpha = 10$ and $\beta = 10$ for the grasping policy, $N_{\text{grasp}} = 2$, and a learning rate of 3^{-4} for training the networks.

A.3 Collision Avoidance

To ensure that we can continue training the robot for extended periods of time, the robot must be able to safely navigate in environments with obstacles without breaking. To support safe navigation, we use the depth channel of the robot’s camera to detect obstacles and walls in front of the robot. When an obstacle is detected as being inside the graspable area of the robot’s arm, the robot’s base rotates in a random direction until the obstacle is no longer obstructing the grasp space. We also simulate this setup for more exhaustive comparisons, with details of the simulation provided in Appendix A.1.

A.4 Real Evaluation Details

Here we include additional details on the evaluations, control policies and implementation for the comparison methods.

Env	no obst	obst	diverse	obst+rugs
<i>ReLMM-StatCurr</i>	25.1	36.2	35.7	50.9
<i>ReLMM-AutoCurr</i>	63.5	–	99.5	–

Table 2: Number of hours ReLMM was trained to achieve the real robot performance noted in Table 1.

ReLMM Due to the real world training constraints, we cannot run the evaluation protocol regularly during real-world training. We did run evaluation for several checkpoints for the ReLMM-StatCurr agent in the obst+rugs room, as shown in Figure 4. The total number of hours trained in each environment is given in Table 2. Frames from the evaluations are shown in Figure 6.

Rand all The navigation policy generates actions from $\pi_n \sim \mathcal{U}[-1, 1]^2$, and similarly the grasping policy chooses uniformly random from the space of discrete actions.

Rand nav [43] The navigation policy is $\pi_n \sim \mathcal{U}[-1, 1]^2$, but the grasping policy is loaded from a run of stationary curriculum right after the initial stationary phase (i.e. with stationary pretraining only).

Scripted The scripted controller takes the camera observation, convert it to grayscale, then detects contours using OpenCV and uses the contour centroids as object locations. Then it projects the pixel locations onto the flat ground plane. For grasping, it simply grasps at the closest position. For navigation, it outputs the forward and turn amount proportional to how much it needs to reach the



Figure 6: Frames from videos of the real robot evaluation in each room. The rooms starting from top down are *no obst*, *obst*, *diverse*.

closest object position, clipped by the action range, or random movement if there are no object in the observation.

B Grasping Curriculum

In Algorithm 3 and 4 we show the pseudocode of our Autonomous Curriculum algorithm. For our training, we also use hyperparameter values $N_{start} = 10$, $N_{stop} = 50$, and $N_{max} = 2000$. In addition to what was written in section 4.4, we include an additional hyperparameters for more stable training, $N_{bt} = 300$, which determines how many grasp there needs to be in the buffer \mathcal{D}_g before grasp training begins.

C Additional Tasks

Our main experiments use the room cleanup task, but the basic design of the ReLMM system can also be extended to other tasks, as we discuss in this section. Using the simulation environment, we evaluate ReLMM on a picking and placing task, which requires picking up objects and placing them on a red rectangle (see Figure 7). The policy is trained in a similar fashion as the grasping policy, with a sparse reward, and uses the same action space. It is given reward of 1 if the object is successfully placed on one of the designated areas marked red, shown in Figure 7, and 0 otherwise. During training, we strictly follow the Algorithm 2 and sample the actions according to the distribution defined in Equation 2. In step 9 of the algorithm the decision of whether to use the grasping or placing policy depends on whether or not an object is already held by the end effector. In contrast to the grasping task, the placing task does not use the ReLMM pseudo-reset mechanism, and instead simply learns to place objects down at random positions in the environment. Analogously to the pre-training procedure we use for the grasping task, we pretrain the grasping and placing policies with $N_{grasp, st}, N_{place, st} = 2000$ samples, which matches the number used for pretraining the original grasping task. N_{obj} is the number of objects available for grasping (20 and 40), and we use $N_{targets} = 3$ random targets for placing. Since the placing portion of the task is easier compared to the more demanding grasping task, we achieve 71% success rate during placing pretraining compared to 60% for the grasping. During the combined training process, the navigation policy is given a reward -1 when not holding an object, -0.5 if the robot is holding the successfully grasped object, and a reward

Algorithm 3 TrainGraspAutoCurr($G^1, \dots, G^N, \mathcal{D}_g, N$)

```
1: if  $|\mathcal{D}_g| \geq N_{max}$  then
2:   return TrainGrasp( $G^1, \dots, G^N, \mathcal{D}_g, N, 0$ )
3: end if
4:  $N_{since} = 0$ 
5:  $r_{max} = 0$ 
6: for  $n = 1, \dots$  do
7:   Get grasp observation  $\tilde{o}$ .
8:   Sample  $a_g \sim \pi_g(\cdot|\tilde{o})$ . // see Equation 2
9:   Perform grasp  $a_g$ , receiving  $r_g = 0$  or 1.
10:  Store  $(\tilde{o}, a_g, r_g)$  in  $\mathcal{D}_g$ .
11:  if  $|\mathcal{D}_g| \geq N_{bt}$  then
12:    Update  $G^1, \dots, G^N$  on  $\mathcal{D}_g$ 
13:  end if
14:  if  $r_g = 1$  then
15:     $N_{since} = 0$ 
16:     $r_{max} = 1$ 
17:  else
18:     $N_{since} = N_{since} + 1$ 
19:  end if
20:  if  $|\mathcal{D}_g| \geq N_{max}$  then
21:    break
22:  end if
23:  if  $r_{max} = 1$  then
24:    if  $N_{since} \geq N_{stop}$  then
25:      break
26:    end if
27:  else
28:    if  $N_{since} \geq N_{start}$  then:
29:      break
30:    end if
31:  end if
32:  if  $r_g = 1$  then:
33:    Drop object randomly in grasping area.
34:  end if
35: end for
36: return  $r_{max}$ 
```

of 0 when the object is successfully placed on a red target, which also terminates the episode. The positions of the target areas are randomized after each successful place. This experiment illustrates that the ReLMM system is general enough to be adapted to other tasks with more complex reward structures.

D Additional Results

Here we include additional results for the paper that include images from the real robot experiments and add to the ablation analysis.

D.1 Additional Simulation Results

In simulation we study three additional conditions for ReLMM. Starting with a comparison of using a *discrete* vs *continuous* action representation. In this experiment the action space uses the same underlying control structure of finding the best X - Y position to grasp on the ground, only the output policy distribution changes. The results in Figure 8 show that training with a discrete policy trains more than twice as fast. Next, we study the affect of using the learned grasping model to

Algorithm 4 ReLMM with AutoCurr

```
1: Init: function estimators  $\pi_n, G^1, \dots, G^M$ .
2: Replay buffers  $\mathcal{D}_n = \{\}, \mathcal{D}_g = \{\}$ 
3: Deleted TrainGrasp( $G^1, \dots, G^N, \mathcal{D}_g, N_{pt}, 1$ )
4: for  $t = 0, \dots, T$  steps do
5:   Get navigation observation  $o_t$ 
6:   Sample  $a_n \sim \pi_n(\cdot|o_t)$  and perform  $a_n$ 
7:   if  $\text{uniform}() \leq \mathbb{P}[\text{grasp}|o_t]$  then
8:      $r_g = \text{TrainGraspAutoCurr}(G^1, \dots, G^M, \mathcal{D}_g, N_{\text{grasp}})$ 
9:   else  $r_g = 0$ 
10:  Compute reward  $r_n = r_g - 1$ 
11:  Get next navigation observation  $o_{t+1}$ 
12:  Store  $(o_t, a_n, r_n, o_{t+1})$  in  $\mathcal{D}_n$ .
13:  Update  $\pi_n$  with  $\mathcal{D}_n$  using SAC.
14:  Pseudo-reset
15: end for
```

relabel rewards for the navigation policy, as described in subsection 4.2. We find that using this relabeling method can increase learning speed and final policy quality, as shown in Figure 9 (left). Last, we compare the different curriculum methods, stationary and autonomous, in a simulated room with obstacles. The stationary curriculum method appears to results in faster learning but requires human intervention to train. The autonomous curriculum may find this environment difficult do to the obstacles which make exploratory navigation less successful.

Ablation. We compare our pseudo-reset method to a prior algorithm for reset-free learning [2], which learns how to perturb the environment between episodes of running the actual task policy by maximizing a novelty based reward. In Figure 5 (right) we see our simpler pseudo-reset is equally or more effective than the learned perturbation method, without requiring any additional learning machinery.

D.2 Comparison to HRL4IN

For HRL4IN [21], we followed the method laid out in the HRL4IN paper. To make the setup match the one in our experiments, the observation space contains the RGB camera image (instead of the depth image HRL4IN uses), global XYZ position of the robot (which is not available on the real robot and not used by our method), and the local position of the gripper. The action spaces for the high-level and low-level are the velocity of the wheels and the change in gripper XYZ position. In the same manner as ReLMM, when the gripper’s height above ground goes below some threshold, the gripper closes and picks from the ground. We use the same high-level policy reward that we employ in ReLMM, where the low-level policy uses the same as the HRL4IN paper. After around 30 days of training in simulation, HRL4IN has collected a total of 30M environment steps, but the performance is still at around 1% objects collected on average during eval. This level of performance is expected, since the HRL4IN paper also took around 30M environment steps to achieve good performance on

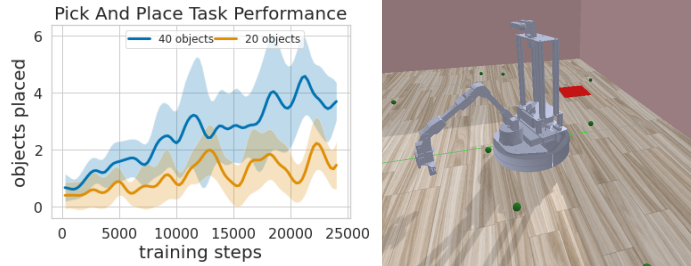


Figure 7: Left: Results for pick and place training, with either 40 or 20 objects available in the environment. We allow a maximum of 250 simulation steps for each evaluation rollout. Right: A snapshot from the proposed pick and place task with the red pace target behind the robot.

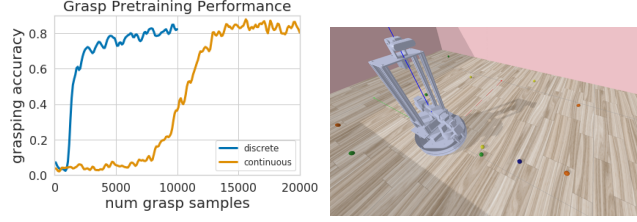


Figure 8: Left: Discrete grasping policies train significantly faster than continuous policies in our simulation ablation study. Right: Images from the simulated environment with diverse objects.

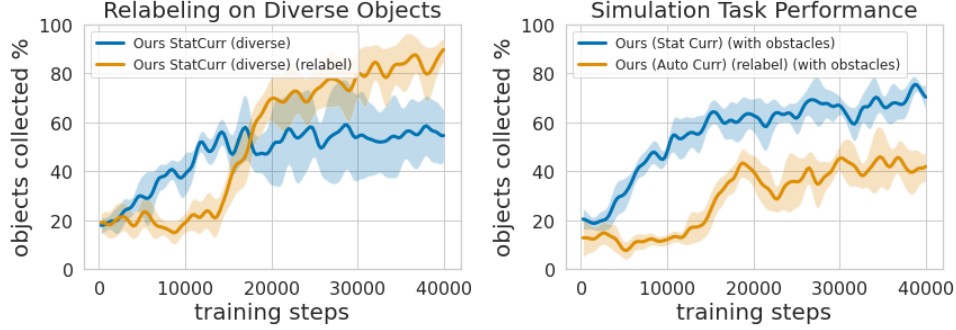


Figure 9: Further curriculum and relabeling comparison in simulation. We plot the performance for the simulated room with diverse objects (left) and with obstacles (right). We find that the relabeling reward helps significantly with the diverse objects because it encourages the navigation to go towards areas of high grasp uncertainty. However, the automatic curriculum in the obstacle room is still slower than the stationary curriculum, even with the best hyperparameters.

their task, but our environment is a sparse reward setting, whereas the HRL4IN paper uses a dense reward environment. On the other hand, our method only required around 30K environment steps to learn a strong policy with 90% objects collected on average during eval. Our method is focused on real-world training, and significantly more efficient.