A ALGORITHM DETAILS

Algorithm 1 MERL: Multimodal end-to-end deep RL

Parametric networks: Image encoder f_ξ, proprioception encoder g_ζ, multimodal fusion for critic $h_{\psi_{\text{critic}}}$, multimodal fusion for actor $h_{\psi_{\text{actor}}}$, actor π_θ, critic Q_ϕ

Hyper-parameters: Training steps T, mini-batch size B, learning rate α, target update rate τ, standard deviation σ, clip value c

Image augmentations: Random shift $\texttt{aug}$

for $t \leftarrow 1 \cdots T$ **do**

$\boldsymbol{a}_t \leftarrow \pi_\theta(h_{\psi_{\text{actor}}}(f_\xi(\boldsymbol{o}_t^{\text{image}}), g_\zeta(\boldsymbol{o}_t^{\text{prop}}))) + \epsilon$ and $\epsilon \sim \mathcal{N}(0, \sigma^2)$

$\boldsymbol{o}_{t+1} \sim P(\cdot \mid \boldsymbol{o}_t, \boldsymbol{a}_t)$

$\mathcal{D} \leftarrow (\boldsymbol{o}_t, \boldsymbol{a}_t, R(\boldsymbol{o}_t, \boldsymbol{a}_t), \boldsymbol{o}_{t+1}) \cup \mathcal{D}$

UPDATECRITIC($\mathcal{D}$)

UPDATEACTOR($\mathcal{D}$)

end for

procedure UPDATECRITIC($\mathcal{D}$)

$\{(\boldsymbol{o}_t, \boldsymbol{a}_t, r_{t:t+n-1}, \boldsymbol{o}_{t+n})\} \sim \mathcal{D}$

$\boldsymbol{r}_t^{\text{mm\_c}}, \boldsymbol{r}_{t+n}^{\text{mm\_c}} \leftarrow h_{\psi_{\text{critic}}}(f_\xi(\texttt{aug}(\boldsymbol{o}_t^{\text{image}})), g_\zeta(\boldsymbol{o}_t^{\text{prop}})), h_{\psi_{\text{critic}}}(f_\xi(\texttt{aug}(\boldsymbol{o}_{t+n}^{\text{image}})), g_\zeta(\boldsymbol{o}_{t+n}^{\text{prop}}))$

$\boldsymbol{r}_t^{\text{mm\_a}}, \boldsymbol{r}_{t+n}^{\text{mm\_a}} \leftarrow h_{\psi_{\text{actor}}}(f_\xi(\texttt{aug}(\boldsymbol{o}_t^{\text{image}})), g_\zeta(\boldsymbol{o}_t^{\text{prop}})), h_{\psi_{\text{actor}}}(f_\xi(\texttt{aug}(\boldsymbol{o}_{t+n}^{\text{image}})), g_\zeta(\boldsymbol{o}_{t+n}^{\text{prop}}))$

$\boldsymbol{a}_{t+n} \leftarrow \pi_\theta(\boldsymbol{r}_{t+n}^{\text{mm\_a}}) + \epsilon$ and $\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma^2), -c, c)$

Compute $\mathcal{L}_{\phi_1, \psi_{\text{critic}}, \xi, \zeta}$ and $\mathcal{L}_{\phi_2, \psi_{\text{critic}}, \xi, \zeta}$

$\xi \leftarrow \xi - \alpha \nabla_\xi (\mathcal{L}_{\phi_1, \psi_{\text{critic}}, \xi, \zeta} + \mathcal{L}_{\phi_2, \psi_{\text{critic}}, \xi, \zeta})$

$\zeta \leftarrow \zeta - \alpha \nabla_\zeta (\mathcal{L}_{\phi_1, \psi_{\text{critic}}, \xi, \zeta} + \mathcal{L}_{\phi_2, \psi_{\text{critic}}, \xi, \zeta})$

$\psi_{\text{critic}} \leftarrow \psi_{\text{critic}} - \alpha \nabla_{\psi_{\text{critic}}} (\mathcal{L}_{\phi_1, \psi_{\text{critic}}, \xi, \zeta} + \mathcal{L}_{\phi_2, \psi_{\text{critic}}, \xi, \zeta})$

$\phi_k \leftarrow \phi_k - \alpha \nabla_{\phi_k} \mathcal{L}_{\phi_k, \psi_{\text{critic}}, \xi, \zeta} \quad \forall k \in \{1, 2\}$

$\bar{\phi}_k \leftarrow (1 - \tau)\bar{\phi}_k + \tau \phi_k \quad \forall k \in \{1, 2\}$

end procedure

procedure UPDATEACTOR($\mathcal{D}$)

$\{(\boldsymbol{o}_t)\} \sim \mathcal{D}$

$\boldsymbol{r}_t^{\text{mm\_c}} \leftarrow h_{\psi_{\text{critic}}}(f_\xi(\texttt{aug}(\boldsymbol{o}_t^{\text{image}})), g_\zeta(\text{o}_t^{\text{prop}}))$

$\boldsymbol{r}_t^{\text{mm\_a}} \leftarrow h_{\psi_{\text{actor}}}(f_\xi(\texttt{aug}(\boldsymbol{o}_t^{\text{image}})), g_\zeta(\text{o}_t^{\text{prop}}))$

$\boldsymbol{a}_t \leftarrow \pi_\theta(\boldsymbol{r}_t^{\text{mm\_a}}) + \epsilon$ and $\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma^2), -c, c)$

Compute $\mathcal{L}_{\theta, \psi_{\text{actor}}}$

$\psi_{\text{actor}} \leftarrow \psi_{\text{actor}} - \alpha \nabla_{\psi_{\text{actor}}} \mathcal{L}_{\theta, \psi_{\text{actor}}}$

$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_{\theta, \psi_{\text{actor}}}$

end procedure

B TASK DETAILS

In this section, we describe details of the three 3D robotic manipulation tasks from DMC (*jaco reach duplo, jaco move box*, and *jaco lift box*) with respect to task descriptions, reward design, and task visualizations.

B.1 TASK DESCRIPTIONS

We design a set of three 3D robotic manipulation tasks from DMC (*jaco reach duplo, jaco move box*, and *jaco lift box*) (Tunyasuvunakool et al., 2020) to include two different types of raw sensory data (that is, RGB images obtained from a fixed vision camera and proprioception obtained from a robot's joint encoders) as observation, where task success requires joint reasoning over visual and proprioceptive feedback. For each task, we randomize the configuration of the initial positions of the robot and the box at the beginning of each episode, during both training and testing, to enhance the robustness and generalization of the model.

In the case of the task *jaco reach duplo*, the robot (more precisely, the center point of the robot's end-effector) is required to reach a duplo randomly placed on a workspace. The task is considered successful when the robot reaches within 5 cm of the duplo.

In the case of the task *jaco move box*, the robot is required to move a box randomly placed on a workspace to a specific target position. The task is considered successful when the box reaches within 1 cm of the target position.

In the case of the task *jaco lift box*, the robot is required to lift a box randomly placed on a workspace to a specific target position. The task is considered successful when the box reaches within 1 cm of the target position.

B.2 REWARD DESIGN

We adopt a staged, structured, and multi-component reward function to guide the RL algorithm, which simplifies the challenge of exploration and leads to effective policy learning (Lee et al., 2020b; Yu et al., 2020). The reward function, R, is a combination of a reaching reward, pushing reward, vertical reward, floating reward, and lifting reward, or subsets thereof for simpler tasks that only include reaching or pushing. With this design, the reward is bounded by $[0, 1]$ per timestep.

Jaco Reach Duplo The reward function for task *jaco reach duplo* is defined as follows:

$$r_t = r_{\text{reaching}} \quad \text{(reaching)} \tag{3}$$

Jaco Move Box The reward function for task *jaco move box* is defined as follows:

$$r_t = \begin{cases} r_{\text{reaching}} & \text{(reaching)} \\ r_{\text{reaching}} + r_{\text{pushing}} & \text{if } \text{dist}(\boldsymbol{p}_{\text{tcp}}, \boldsymbol{p}_{\text{obj}}) < \varepsilon_{\text{reaching}} \quad \text{(pushing)} \end{cases} \tag{4}$$

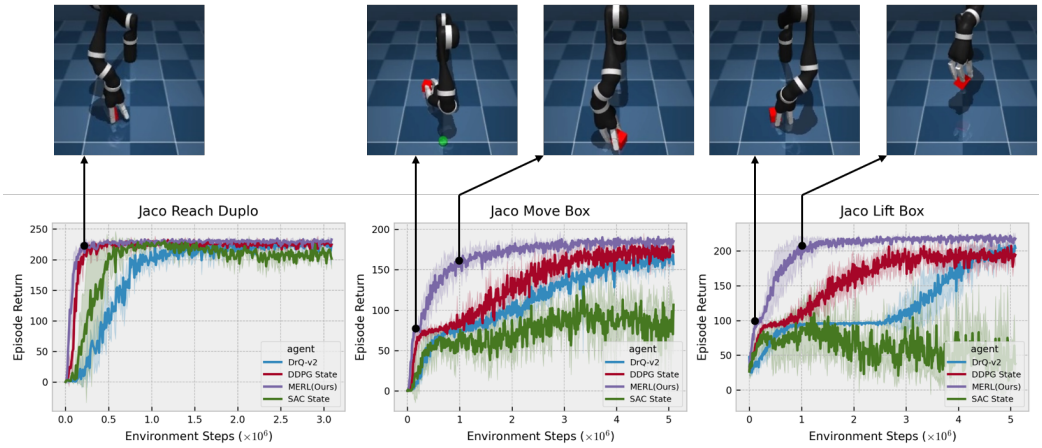
Jaco Lift Box The reward function for task *jaco lift box* is defined as follows:

$$r_t = \begin{cases} r_{\text{reaching}} + r_{\text{vertical}} & & \text{(reaching)} \\ r_{\text{reaching}} + r_{\text{vertical}} + r_{\text{floating}} & \text{if } \texttt{dist}(\boldsymbol{p}_{\text{tcp}}, \boldsymbol{p}_{\text{obj}}) < \varepsilon_{\text{reaching}} & \text{(floating)} \\ r_{\text{reaching}} + r_{\text{vertical}} + r_{\text{floating}} + r_{\text{lifting}} & \text{if } z_{\text{obj}} > \varepsilon_{\text{floating}} & \text{(lifting)} \end{cases} \tag{5}$$

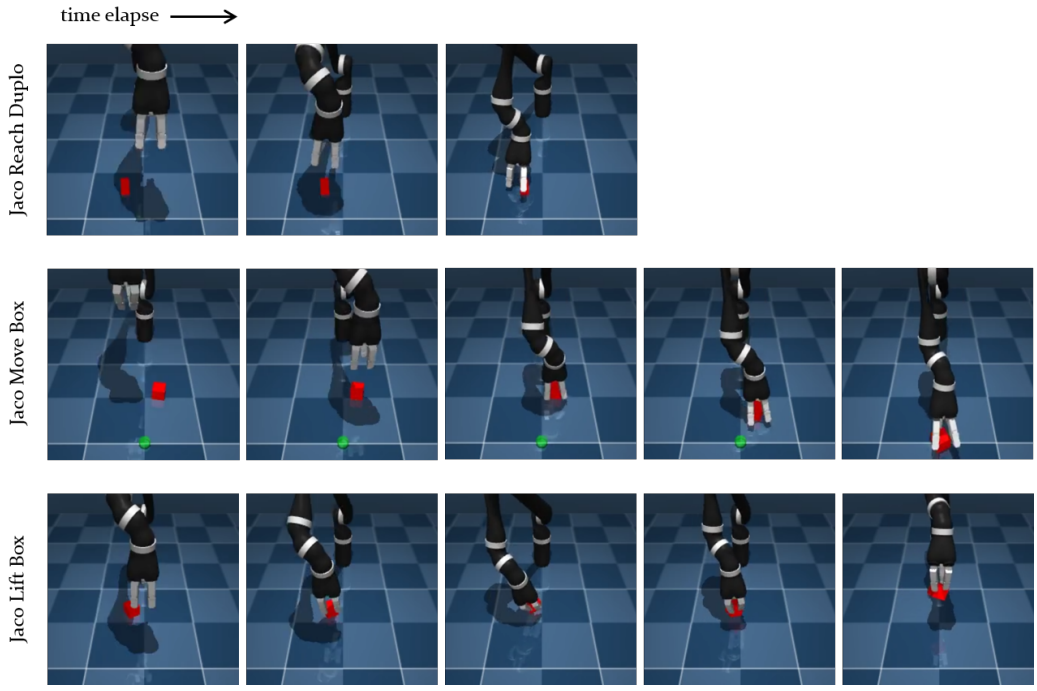
where $r_{\text{reaching}} = \frac{1}{N}\texttt{tolerance}\big(\texttt{dist}(\boldsymbol{p}_{\text{tcp}}, \boldsymbol{p}_{\text{obj}}), \varepsilon_{\text{reaching}}\big)$, $r_{\text{pushing}} = \frac{1}{N}\texttt{tolerance}\big(\texttt{dist}(\boldsymbol{p}_{\text{target}}, \boldsymbol{p}_{\text{obj}}), \varepsilon_{\text{pushing}}\big)$, $r_{\text{vertical}} = \frac{1}{N}\texttt{tolerance}\big(\texttt{cosdist}(\boldsymbol{v}_{\text{hand}}, \boldsymbol{u}_{\text{-z}}), 1 - \varepsilon_{\text{vertical}}\big)$, $r_{\text{floating}} = \frac{1}{N}\frac{z_{\text{obj}}}{z_{\text{target}}}$, and $r_{\text{lifting}} = \frac{1}{N}\texttt{tolerance}\big(\texttt{dist}(\boldsymbol{p}_{\text{obj}}, \boldsymbol{p}_{\text{target}}), \varepsilon_{\text{lifting}}\big)$. $\boldsymbol{p}_{\text{tcp}}$ represents the position of the end-effector's center point, $\boldsymbol{p}_{\text{obj}}$ represents the position of the object (here, the box's center point), $\boldsymbol{p}_{\text{target}}$ represents the target position, z_{obj} represents the z-axis value of the object's center point, z_{target} represents the z-axis value of the target position, $\boldsymbol{v}_{\text{hand}}$ represents a unit vector vertical to the robot hand, and $\boldsymbol{u}_{\text{-z}}$ represents a unit vector $[0, 0, -1]$. The number of stages is represented by N. Here, we adopted the same function `tolerance` as in DMC (Tunyasuvunakool et al., 2020).

B.3 TASK VISUALIZATIONS

Figure 4 provides visualizations for behaviors generated by MERL in relation to the three 3D robotic manipulation tasks from DMC (*jaco reach duplo, jaco move box*, and *jaco lift box*). The visualization of behaviors learned by MERL during the training procedure is shown in Figure 4a. We note here that MERL solves all three of the 3D robotic manipulation tasks in less than 1M environment steps. Figure 4b illustrates the visualization of successful trajectories generated by MERL.



(a) Visualization of final behaviors generated by MERL during the training procedure.



(b) Visualization of successful trajectories generated by MERL.

Figure 4: Visualizations for behaviors generated by our method (MERL) in relation to the 3D robotic manipulation tasks from DMC (*jaco reach duplo, jaco move box*, and *jaco lift box*): (a) we visualize the behaviors learned by MERL during the training procedure and (b) we visualize the successful trajectories learned by MERL.

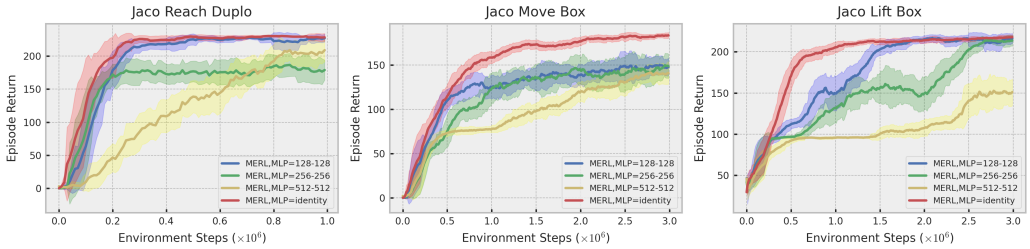


Figure 5: An additional ablation study that led us to the final version of MERL in relation to the proprioception encoder. We observe that the proprioception encoder set as the `identity` encoder (*red*) provides the best performance for the given 3D robotic manipulation tasks compared to MLP encoders (*blue, green, and yellow*).

C Implementation Details

In this section, we describe our implementation details for MERL. We use PyTorch as a deep learning tool and DMC and MuJoCo for our simulation. We conduct our experiments on a workstation with an Intel i9-9980XE CPU and Nvidia Quadro RTX 8000 GPU.

C.1 Network Architectures

For all networks, we initialize the weight matrix of the convolutional and fully-connected layers with an orthogonal initialization (Saxe et al., 2013) and set the bias to be zero.

Image Encoder Network The image encoder network is modeled as four convolutional layers with 3×3 kernels and 32 channels, as in SAC+AE (Yarats et al., 2021c). An `ReLU` activation is applied after each convolutional layer. We use stride 1 everywhere, except for the first convolutional layer, which has stride 2. Here, we note that only the critic optimizer is allowed to update the image encoder network weights (that is, we prevent the actor's gradients from updating the image encoder network weights).

Proprioception Encoder Network The proprioception encoder network is modeled as a 2-layer MLP with `ReLU` activations after each layer. Here, the final version of our method uses the proprioception encoder as the `identity` encoder. This is because as shown in Figure 5, contrary to the results in (Lee et al., 2019a; 2020b), the case where the proprioception encoder is set as the `identity` encoder provides better performance in relation to the three 3D robotic manipulation tasks from DMC (*jaco reach duplo, jaco move box*, and *jaco lift box*), compared to the case where the proprioception encoder is set as an MLP encoder. This suggests that the proprioception itself is enough to learn a latent multimodal representation and a policy in an efficient, joint, and end-to-end manner, without the need for additional encodings; that is, it is already well-encoded. Note that only the critic optimizer is allowed to update the proprioception encoder network weights.

Multimodal Fusion Network The encoded visual and proprioceptive representations are fed into multimodal fusion to learn a latent multimodal representation. The multimodal fusion network is modeled as a single fully-connected layer normalized by `LayerNorm` (Ba et al., 2016). After `LayerNorm`, the `tanh` nonlinearity is applied to the output of the fully-connected layer, which serves to match the latent multimodal representation and action to the same scale. Finally, each output is concatenated to produce a single latent multimodal representation vector of dimension `d`. The weights of each multimodal fusion network for actor and critic are updated by the gradients of actor and critic, respectively.

Actor and Critic Networks As in TD3 (Fujimoto et al., 2018), we use clipped double Q-learning for the critic network, where each Q-function is parametrized as a 3-layer (in the case of the tasks *jaco reach box* and *jaco lift box*) or 4-layer (in the case of the task *jaco move box*) MLP with `LeakyReLU` activations with α of 0.1 after each layer except the last. The actor network is also modeled as a 3-layer (in the case of the tasks *jaco reach box* and *jaco lift box*) or 4-layer (in the case of the task *jaco*

move box) MLP with `LeakyReLU` activations with α of 0.1, which outputs the mean and covariance for the diagonal Gaussian that represents the policy. The hidden dimension is set to 1024 for both the critic and the actor.

C.2 HYPER-PARAMETERS

We provide a comprehensive overview of the hyper-parameters used in the case of the three 3D robotic manipulation tasks from DMC (*jaco reach duplo, jaco move box*, and *jaco lift box*) in Table 1.

Table 1: An overview of hyper-parameters.

Parameter	Setting
Replay buffer capacity	10^6
Mini-batch size	256
Frame stack	3
Seed frames	4000
Exploration steps	2000
Action repeat	1
Discount factor	0.99
Optimizer	Adam
Learning rate	10^{-4}
Soft-update rate	0.01
n-step returns	3
Exploration std. dev.	0.2
Exploration std. dev. clip	0.3
Hidden dim. for actor–critic	1024
Latent dim. for multimodal representation	128