A. Experiment Details

In this section, we describe implementation details and specify the hyperparameters used for our algorithm. In all our experiments, the policy and critic(s) are implemented with feed-forward neural networks. We use Adam (Kingma & Ba, 2015) for optimization.

In our continuous control tasks, the policy returns a Gaussian distribution with a diagonal covariance matrix, i.e, $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s}) = \mathcal{N}(\mu, \boldsymbol{\Sigma})$. The policy is parametrized by a neural network, which outputs the mean $\mu = \mu(\boldsymbol{s})$ and diagonal Cholesky factors $A = A(\boldsymbol{s})$, such that $\Sigma = AA^T$. The diagonal factor A has positive diagonal elements enforced by the softplus transform $A_{ii} \leftarrow \log(1 + \exp(A_{ii}))$ to ensure positive definiteness of the diagonal covariance matrix.

Table 2 shows the default hyperparameters that we use for MO-MPO and scalarized MPO, and Table 3 shows the hyperparameters that differ from these defaults for the humanoid, Shadow Hand, and Sawyer experiments. For all tasks that we train policies with MO-MPO for, a separate critic is trained for each objective, with a shared first layer. The only exception is the action norm penalty in humanoid ***run***, which is computed exactly from the action. We found that for both policy and critic networks, layer normalization of the first layer followed by a hyperbolic tangent ($\tanh$) is important for the stability of the algorithm.

In our experiments comparing the Pareto front found by MO-MPO versus scalarized MPO, we ran MO-MPO with a range of ϵ_k settings and ran scalarized MPO with a range of weight settings, to obtain a set of policies for each algorithm. The settings for humanoid ***run*** and the Shadow Hand tasks are listed in Table 4. The settings for humanoid mocap and Sawyer are specified in the main paper, in Sec. 6.3 and 6.4, respectively.

In the following subsections, we first give suggestions for choosing appropriate ϵ_k to encode a particular preference (Sec. A.1), then we describe the discrete MO-MPO used for Deep Sea Treasure (Sec. A.2), and finally we describe implementation details for humanoid mocap (Sec. A.3).

A.1. Suggestions for Setting ϵ

Our proposed algorithm, MO-(V-)MPO, requires practitioners to translate a desired preference across objectives to numerical choices for ϵ_k for each objective k. At first glance, this may seem daunting. However, in practice, we have found that encoding preferences via ϵ_k is often more intuitive than doing so via scalarization. In this subsection, we seek to give an intuition on how to set ϵ_k for different desired preferences. Recall that each ϵ_k controls the influence of objective k on the policy update, by constraining the KL-divergence between each objective-specific distribution and the current policy (Sec. 4.1). We generally choose ϵ_k in the range of 0.001 to 0.1.

Hyperparameter	**Default**
policy network	
layer sizes	$(300, 200)$
take $\tanh$ of action mean?	yes
minimum variance	10^{-12}
maximum variance	unbounded
critic network(s)	
layer sizes	$(400, 400, 300)$
take $\tanh$ of action?	yes
Retrace sequence size	8
discount factor γ	0.99
both policy and critic networks	
layer norm on first layer?	yes
$\tanh$ on output of layer norm?	yes
activation (after each hidden layer)	ELU
MPO	
actions sampled per state	20
default ϵ	0.1
KL-constraint on policy mean, β_μ	10^{-3}
KL-constraint on policy covariance, β_Σ	10^{-5}
initial temperature η	1
training	
batch size	512
replay buffer size	10^6
Adam learning rate	3×10^{-4}
Adam ϵ	10^{-3}
target network update period	200

Table 2. Default hyperparameters for MO-MPO and scalarized MPO, with decoupled update on mean and covariance (Sec. C.3).

Equal Preference. When all objectives are equally important, the general rule is to set all ϵ_k to the same value. We did this in our ***orient*** and humanoid mocap tasks, where objectives are aligned or mostly aligned. In contrast, it can be tricky to choose appropriate weights in linear scalarization to encode equal preferences—setting all weights equal to $1/K$ (where K is the number of objectives) is only appropriate if the objectives' rewards are of similar scales. We explored this in Sec. 5.1 and Fig. 3.

When setting all ϵ_k to the same value, what should this value be? The larger ϵ_k is, the more influence the objectives will have on the policy update step. Since the per-objective critics are learned in parallel with the policy, setting ϵ_k too high tends to destabilize learning, because early on in training, when the critics produce unreliable Q-values, their influence on the policy will lead it in the wrong direction. On the other hand, if ϵ_k is set too low, then it slows down learning, because the per-objective action distribution is only allowed to deviate by a tiny amount from the current policy, and the updated policy is obtained via supervised learning on the combination of these action distributions.

Humanoid *run*	
policy network	
layer sizes	$(400, 400, 300)$
take $\tanh$ of action mean?	no
minimum variance	10^{-8}
critic network(s)	
layer sizes	$(500, 500, 400)$
take $\tanh$ of action?	no
MPO	
KL-constraint on policy mean, β_μ	5×10^{-3}
KL-constraint on policy covariance, β_Σ	10^{-6}
training	
Adam learning rate	2×10^{-4}
Adam ϵ	10^{-8}

Shadow Hand	
MPO	
actions sampled per state	30
default ϵ	0.01

Sawyer	
MPO	
actions sampled per state	15
training	
target network update period	100

Table 3. Hyperparameters for humanoid, Shadow Hand, and Sawyer experiments that differ from the defaults in Table 2.

Condition	Settings
Humanoid ***run*** (one seed per setting)	
scalarized MPO	$w_{\text{task}} = 1 - w_{\text{penalty}}$
	$w_{\text{penalty}} \in \text{linspace}(0, 0.15, 100)$
MO-MPO	$\epsilon_{\text{task}} = 0.1$
	$\epsilon_{\text{penalty}} \in \text{linspace}(10^{-6}, 0.15, 100)$
Humanoid ***run***, normal vs. scaled (three seeds per setting)	
scalarized MPO	$w_{\text{task}} = 1 - w_{\text{penalty}}$
	$w_{\text{penalty}} \in \{0.01, 0.05, 0.1\}$
MO-MPO	$\epsilon_{\text{task}} = 0.1$
	$\epsilon_{\text{penalty}} \in \{0.01, 0.05, 0.1\}$
Shadow Hand ***touch*** and ***turn*** (three seeds per setting)	
scalarized MPO	$w_{\text{task}} = 1 - w_{\text{penalty}}$
	$w_{\text{penalty}} \in \text{linspace}(0, 0.9, 10)$
MO-MPO	$\epsilon_{\text{task}} = 0.01$
	$\epsilon_{\text{penalty}} \in \text{linspace}(0.001, 0.015, 15)$
Shadow Hand ***orient*** (ten seeds per setting)	
scalarized MPO	$w_{\text{touch}} = w_{\text{height}} = w_{\text{orientation}} = 1/3$
MO-MPO	$\epsilon_{\text{touch}} = \epsilon_{\text{height}} = \epsilon_{\text{orientation}} = 0.01$

Table 4. Settings for ϵ_k and weights. (linspace(x, y, z) denotes a set of z evenly-spaced values between x and y.)

Eventually the learning will converge to more or less the same policy though, as long as ϵ_k is not set too high.

Unequal Preference. When there is a difference in preferences across objectives, the *relative* scale of ϵ_k is what matters. The more the relative scale of ϵ_k is compared to ϵ_l, the more influence objective k has over the policy update, compared to objective l. And in the extreme case, when ϵ_l is near-zero for objective l, then objective l will have no influence on the policy update and will effectively be ignored. We explored this briefly in Sec. 5.1 and Fig. 4.

One common example of unequal preferences is when we would like an agent to complete a task, while minimizing other objectives—e.g., an action norm penalty, "pain" penalty, or wrist force-torque penalty, in our experiments. In this case, the ϵ for the task objective should be higher than that for the other objectives, to incentivize the agent to prioritize actually doing the task. If the ϵ for the penalties is too high, then the agent will care more about minimizing the penalty (which can typically be achieved by simply taking no actions) rather than doing the task, which is not particularly useful.

The scale of ϵ_k has a similar effect as in the equal preference case. If the scale of ϵ_k is too high or too low, then the same issues arise as discussed for equal preferences. If all ϵ_k increase or decrease in scale by the same (moderate) factor, and thus their relative scales remain the same, then typically they will converge to more or less the same policy. This can be seen in Fig. 5 (right), in the Deep Sea Treasure domain: regardless of whether ϵ_{time} is 0.01, 0.02, or 0.05, the relationship between the relative scale of $\epsilon_{\text{treasure}}$ and ϵ_{time} to the treasure that the policy converges to selecting is essentially the same.

A.2. Deep Sea Treasure

In order to handle discrete actions, we make several minor adjustments to scalarized MPO and MO-MPO. The policy returns a categorical distribution, rather than a Gaussian. The policy is parametrized by a neural network, which outputs a vector of logits (i.e., unnormalized log probabilities) of size $|\mathcal{A}|$. The KL constraint on the change of this policy, β, is 0.001. The input to the critic network is the state concatenated with a four-dimensional one-hot vector denoting which action is chosen (e.g., the up action corresponds to $[1, 0, 0, 0]^\top$). Critics are trained with one-step temporal-difference error, with a discount of 0.999. Other than these changes, the network architectures and the MPO and training hyperparameters are the same as in Table 2.

Humanoid Mocap	
Scalarized VMPO	
KL-constraint on policy mean, β_μ	0.1
KL-constraint on policy covariance, β_Σ	10^{-5}
default ϵ	0.1
initial temperature η	1
Training	
Adam learning rate	10^{-4}
batch size	128
unroll length (for n-step return, Sec. D.1)	32

Table 5. Hyperparameters for the humanoid mocap experiments.

A.3. Humanoid Mocap

For the humanoid mocap experiments, we used the following architecture for both MO-VMPO and VMPO: for the policy, we process a concatentation of the mocap reference observations and the proprioceptive observations by a two layer MLP with 1024 hidden units per layer. This reference encoder is followed by linear layers to produce the mean and log standard deviation of a stochastic latent variable. These latent variables are then concatenated with the proprioceptive observations and processed by another two layer MLP with 1024 hidden units per layer, to produce the action distribution. For VMPO, we use a three layer MLP with 1024 hidden units per layer as the critic. For MO-VMPO we use a shared two layer MLP with 1024 hidden units per layer followed by a one layer MLP with 1024 hidden units per layer per objective. In both cases we use k-step returns to train the critic with a discount factor of 0.95. Table 5 shows additional hyperparameters used in our experiments.

B. Experimental Domains

We evaluated our approach on one discrete domain (Deep Sea Treasure), three simulated continuous control domains (humanoid, Shadow Hand, and humanoid mocap), and one real-world continuous control domain (Sawyer robot). Here we provide more detail about these domains and the objectives used in each task.

B.1. Deep Sea Treasure

Deep Sea Treasure (DST) is a 11×10 grid-world domain, the state space $\mathcal{S}$ consists of the x and y position of the agent and the action space $\mathcal{A}$ is $\{\texttt{up}, \texttt{right}, \texttt{down}, \texttt{left}\}$. The layout of the environment and values of the treasures are shown in Fig. 8. If the action would cause the agent to collide with the sea floor or go out of bounds, it has no effect. Farther-away treasures have higher values. The episode terminates when the agent collects a treasure, or after 200 timesteps.

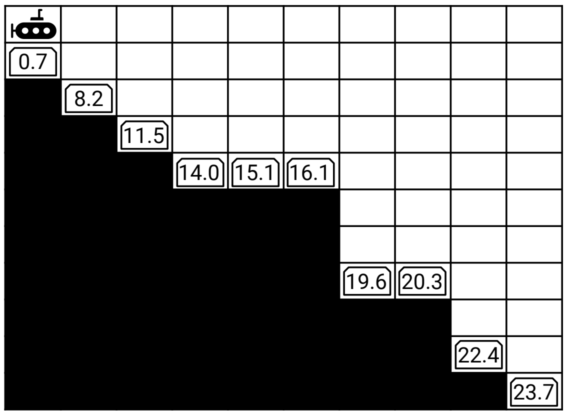

Figure 8. Deep Sea Treasure environment from Vamplew et al. (2011), with weights from Yang et al. (2019). Treasures are labeled with their respective values. The agent can move around freely in the white squares, but cannot enter the black squares (i.e., the ocean floor).

There are two objectives, time penalty and treasure value. A time penalty of -1 is given at each time step. The agent receives the value of the treasure it picks up as the reward for the treasure objective. In other words, when the agent picks up a treasure of value v, the reward vector is $[-1, v]$; otherwise it is $[-1, 0]$.

B.2. Shadow Hand

Our robot platform is a simulated Shadow Dexterous Hand (Shadow Robot Company, 2020) in the MuJoCo physics engine (Todorov et al., 2012). The Shadow Hand has five fingers and 24 degrees of freedom, actuated by 20 motors. The observation consists of joint angles, joint velocities, and touch sensors, for a total of 63 dimensions. Each fingertip of our Shadow Hand has a 4×4 spatial touch sensor. This sensor has three channels, measuring the normal force and the x and y-axis tangential forces, for a sensor dimension of $4\times4\times3$ per fingertip. We simplify this sensor by summing across spatial dimensions, resulting in a $1\times1\times3$ observation per fingertip.

In the ***touch*** task, there is a block in the environment that is always fixed in the same pose. In the ***turn*** task, there is a dial in the environment that can be rotated, and it is initialized to a random position between $-30°$ and $30°$. The target angle of the dial is $0°$. The angle of the dial is included in the agent's observation. In the ***orient*** task, the robot interacts with a rectangular peg in the environment; the initial and target pose of the peg remains the same across episodes. The pose of the block is included in the agent's observation, encoded as the xyz positions of four corners of the block (based on how Levine et al. (2016) encodes end-effector pose).

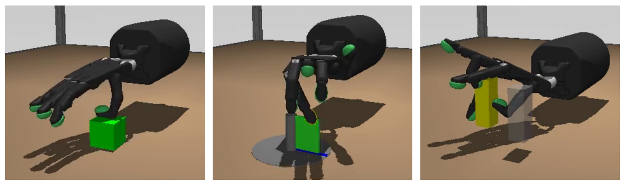

Figure 9. These images show what task completion looks like for the ***touch***, ***turn***, and ***orient*** Shadow Hand tasks (from left to right).

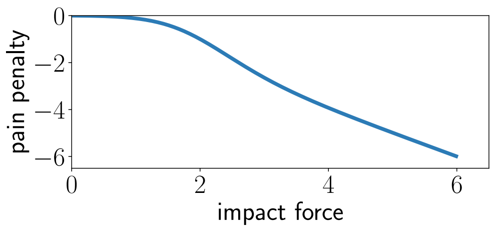

Figure 10. In the ***touch*** and ***turn*** Shadow Hand tasks, the pain penalty is computed based on the impact force, as plotted here. For low impact forces, the pain penalty is near-zero. For high impact forces, the pain penalty is equal to the negative of the impact force.

B.2.1. Balancing Task Completion and Pain

In the ***touch*** and ***turn*** tasks, there are two objectives, "pain" penalty and task completion. A sparse task completion reward of 1 is given for pressing a block with greater than 5N of force or turning a dial to a fixed target angle, respectively. In both tasks, the episode terminates when the agent completes the task; i.e., the agent gets a total reward of either 0 or 1 for the task completion objective per episode. The Pareto plots for these two tasks in the main paper (Fig. 6) show the *discounted* task reward (with a discount factor of $\gamma = 0.99$), to capture how long it takes agents to complete the task.

The "pain" penalty is computed as in (Huang et al., 2019). It is based on the impact force $m(s, s')$, which is the increase in force from state s to the next state s'. In our tasks, this is measured by a touch sensor on the block or dial. The pain penalty is equal to the negative of the impact force, scaled by how unacceptable it is:

$$r_{\text{pain}}(s, a, s') = -\big[1 - a_\lambda(m(s, s'))\big] m(s, s'), \quad (6)$$

where $a_\lambda(\cdot) \in [0, 1]$ computes the acceptability of an impact force. $a_\lambda(\cdot)$ should be a monotonically decreasing function, that captures how resilient the robot and the environment are to impacts. As in (Huang et al., 2019), we use

$$a_\lambda(m) = \text{sigmoid}(\lambda_1(-m + \lambda_2)), \quad (7)$$

with $\lambda = [2, 2]^\top$. The relationship between pain penalty and impact force is plotted in Fig. 10.

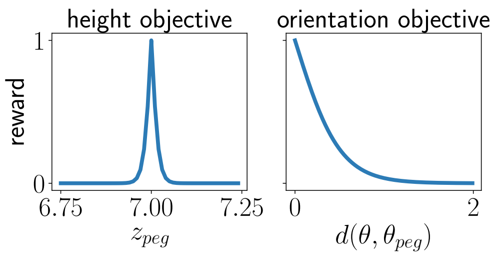

Figure 11. In the ***orient*** Shadow Hand task, the height and orientation objectives have shaped rewards, as plotted here.

B.2.2. Aligned Objectives

In the ***orient*** task, there are three non-competing objectives: touching the peg, lifting the peg to a target height, and orienting the peg to be perpendicular to the ground. The target z position and orientation is shown by the gray transparent peg (Fig. 9, right-most); note that the x and y position is not specified, so the robot in the figure gets the maximum reward with respect to all three objectives.

The touch objective has a sparse reward of 1 for activating the peg's touch sensor, and zero otherwise. For the height objective, the target z position is 7cm above the ground; the peg's z position is computed with respect to its center of mass. The shaped reward for this objective is $1 - \tanh(50|z_{\text{target}} - z_{\text{peg}}|)$. For the orientation objective, since the peg is symmetrical, there are eight possible orientations that are equally valid. The acceptable target orientations Q_{target} and the peg's orientation q_{peg} are denoted as quaternions. The shaped reward is computed with respect to the closest target quaternion, as

$$\min_{q \in Q_{\text{target}}} 1 - \tanh(2^* d(q, q_{\text{peg}})),$$

where $d(\cdot)$ denotes the $\ell 2$-norm of the axis-angle equivalent (in radians) for the distance between the two quaternions. Fig. 11 shows the shaping of the height and orientation rewards.

B.3. Humanoid

We make use of the humanoid ***run*** task from Tassa et al. (2018).[8] The observation dimension is 67 and the action dimension is 21. Actions are joint accelerations with minimum and maximum limits of -1 and 1, respectively. For this task there are two objectives:

- The original task reward given by the environment. The goal is is to achieve a horizontal speed of 10 meters per second, in any direction. This reward is shaped: it is equal to $\min(h/10, 1)$ where h is in meters per second. For this objective we learn a Q-function.

[8] This task is available at github.com/deepmind/dm_control.

- Limiting energy usage, by penalizing taking high-magnitude actions. The penalty is the $\ell 2$-norm of the action vector, i.e, $r_{\text{penalty}}(\boldsymbol{a}) = -\|\boldsymbol{a}\|_2$. For this objective, we do not learn a Q-function. Instead, we compute this penalty directly to evaluate a given action in a *state-independent way* during policy optimization.

B.4. Humanoid Mocap

The motion capture tracking task used in this paper was developed by Hasenclever et al. (2020). We use a simulated humanoid adapted from the "CMU humanoid" available at dm_control/locomotion (Merel et al., 2019). We adjusted various body and actuator parameters to be more comparable to that of an average human. This humanoid has 56 degrees of freedom and the observation is 1021-dimensional.

This motion capture tracking task is broadly similar to previous work on motion capture tracking (Chentanez et al., 2018; Peng et al., 2018; Merel et al., 2019). The task is associated with an underlying set of motion capture clips. For this paper we used roughly 40 minutes of locomotion motion capture clips from the CMU motion capture database.[9]

At the beginning of a new episode, we select a frame uniformly at random from all frames in the underlying mocap data (excluding the last 10 frames of each clip). The simulated humanoid is then initialized to the pose of the selected frame.

The observations in this environment are various proprioceptive observations about the current state of the body as well as the relative target positions and orientations of 13 different body parts in the humanoid's local frame. We provide the agent with the target poses for the next 5 timesteps.

The choice of reward function is crucial for motion capture tracking tasks. Here we consider five different reward components, each capturing a different aspect of the similarity of the pose between the simulated humanoid and the mocap target. Four of our reward components were initially proposed by Peng et al. (2018). The first reward component is based on the difference in the center of mass:

$$r_{\text{com}} = \exp\left(-10\|p_{\text{com}} - p_{\text{com}}^{\text{ref}}\|^2\right) ,$$

where p_{com} and $p_{\text{com}}^{\text{ref}}$ are the positions of the center of mass of the simulated humanoid and the mocap reference, respectively. The second reward component is based on the difference in joint angle velocities:

$$r_{\text{vel}} = \exp\left(-0.1\|q_{\text{vel}} - q_{\text{vel}}^{\text{ref}}\|^2\right) ,$$

where q_{vel} and $q_{\text{vel}}^{\text{ref}}$ are the joint angle velocities of the simulated humanoid and the mocap reference, respectively. The third reward component is based on the difference in end-effector positions:

$$r_{\text{app}} = \exp\left(-40\|p_{\text{app}} - p_{\text{app}}^{\text{ref}}\|^2\right) ,$$

where p_{app} and $p_{\text{app}}^{\text{ref}}$ are the end effector positions of the simulated humanoid and the mocap reference, respectively. The fourth reward component is based on the difference in joint orientations:

$$r_{\text{quat}} = \exp\left(-2\|q_{\text{quat}} \ominus q_{\text{quat}}^{\text{ref}}\|^2\right) ,$$

where $\ominus$ denotes quaternion differences and q_{quat} and $q_{\text{quat}}^{\text{ref}}$ are the joint quaternions of the simulated humanoid and the mocap reference, respectively.

Finally, the last reward component is based on difference in the joint angles and the Euclidean positions of a set of 13 body parts:

$$r_{\text{trunc}} = 1 - \frac{1}{0.3}(\overbrace{\|b_{\text{pos}} - b_{\text{pos}}^{\text{ref}}\|_1 + \|q_{\text{pos}} - q_{\text{pos}}^{\text{ref}}\|_1}^{:=\varepsilon}),$$

where b_{pos} and $b_{\text{pos}}^{\text{ref}}$ correspond to the body positions of the simulated character and the mocap reference and q_{pos}, and $q_{\text{pos}}^{\text{ref}}$ correspond to the joint angles.

We include an early termination condition in our task that is linked to this last reward term. We terminate the episode if $\varepsilon > 0.3$, which ensures that $r_{\text{trunc}} \in [0, 1]$.

In our MO-VMPO experiments, we treat each reward component as a separate objective. In our VMPO experiments, we use a reward of the form:

$$r = \frac{1}{2} r_{\text{trunc}} + \frac{1}{2}\left(0.1 r_{\text{com}} + \lambda r_{vel} + 0.15 r_{\text{app}} + 0.65 r_{\text{quat}}\right) , \tag{8}$$

varying λ as described in the main paper (Sec. 6.3).

B.5. Sawyer

The Sawyer peg-in-hole setup consists of a Rethink Robotics Sawyer robot arm. The robot has 7 joints driven by series elastic actuators. On the wrist, a Robotiq FT 300 force-torque sensor is mounted, followed by a Robotiq 2F85 parallel gripper. We implement a 3D end-effector Cartesian velocity controller that maintains a fixed orientation of the gripper. The agent controls the Cartesian velocity setpoints. The gripper is not actuated in this setup. The observations provided are the joint positions, velocities, and torques, the end-effector pose, the wrist force-torque measurements, and the joint velocity command output of the Cartesian controller. We augment each observation with the two previous observations as a history.

In each episode, the peg is initalized randomly within an 8×8×8 cm workspace (with the peg outside of the hole).

[9]This database is available at mocap.cs.cmu.edu.

The environment is then run at 20 Hz for 600 steps. The action limits are ± 0.05 m/s on each axis. If the magnitude of the wrist force-torque measurements exceeds 15 N on any horizontal axis (x and y axis) or 5 N on the vertical axis (z axis), the episode is terminated early.

There are two objectives: the task itself (inserting the peg), and minimizing wrist force measurements.

The reward for the task objective is defined as

$$r_{\text{insertion}} = \max\Big(0.2 r_{\text{approach}}, r_{\text{inserted}}\Big) \tag{9}$$

$$r_{\text{approach}} = s(p^{\text{peg}}, p^{\text{opening}}) \tag{10}$$

$$r_{\text{inserted}} = r_{\text{aligned}}\ s(p_z^{\text{peg}}, p_z^{\text{bottom}}) \tag{11}$$

$$r_{\text{aligned}} = p_{xy}^{\text{peg}} - p_{xy}^{\text{opening}} < d/2 \tag{12}$$

where p^{peg} is the peg position, p^{opening} the position of the hole opening, p^{bottom} the bottom position of the hole and d the diameter of the hole.

$s(p_1, p_2)$ is a shaping operator

$$s(p_1, p_2) = 1 - \tanh^2\Big(\frac{\operatorname{atanh}(\sqrt{l})}{\epsilon}\|p_1 - p_2\|_2\Big), \tag{13}$$

which gives a reward of $1 - l$ if p_1 and p_2 are at a Euclidean distance of ϵ. Here we chose $l_{\text{approach}} = 0.95$, $\epsilon_{\text{approach}} = 0.01$, $l_{\text{inserted}} = 0.5$, and $\epsilon_{\text{inserted}} = 0.04$. Intentionally, 0.04 corresponds to the length of the peg such that $r_{\text{inserted}} = 0.5$ if $p^{\text{peg}} = p^{\text{opening}}$. As a result, if the peg is in the approach position, r_{inserted} is dominant over r_{approach} in $r_{\text{insertion}}$.

Intuitively, there is a shaped reward for aligning the peg with a magnitude of 0.2. After accurate alignment within a horizontal tolerance, the overall reward is dominated by a vertical shaped reward component for insertion. The horizontal tolerance threshold ensures that the agent is not encouraged to try to insert the peg from the side.

The reward for the secondary objective, minimizing wrist force measurements, is defined as

$$r_{\text{force}} = -\|F\|_1, \tag{14}$$

where $F = (F_x, F_y, F_z)$ are the forces measured by the wrist sensor.

C. Algorithmic Details

C.1. General Algorithm Description

We maintain one online network and one target network for each Q-function associated to objective k, with parameters denoted by ϕ_k and ϕ'_k, respectively. We also maintain one online network and one target network for the policy, with parameters denoted by θ and θ', respectively. Target networks are updated every fixed number of steps by copying parameters from the online network. Online networks are updated using gradient descent in each learning iteration. Note that in the main paper, we refer to the target policy network as the old policy, π_{old}.

We use an asynchronous actor-learner setup, in which actors regularly fetch policy parameters from the learner and act in the environment, writing these transitions to the replay buffer. We refer to this policy as the behavior policy throughout the paper. The learner uses the transitions in the replay buffer to update the (online) Q-functions and the policy. Please see Algorithms 2 and 3 for more details on the actor and learner processes, respectively.

Algorithm 2 MO-MPO: Asynchronous Actor

```
1: given (N) reward functions $\{r_k(s,a)\}_{k=1}^N$, $T$ steps per episode
2: repeat
3:   Fetch current policy parameters $\boldsymbol{\theta}$ from learner
4:   // Collect trajectory from environment
5:   $\tau = \{\}$
6:   for $t = 0, \dots, T$ do
7:     $a_t \sim \pi_{\boldsymbol{\theta}}(\cdot|s_t)$
8:     // Execute action and determine rewards
9:     $\boldsymbol{r} = [r_1(s_t,a_t), \dots, r_N(s_t,a_t)]$
10:    $\tau \leftarrow \tau \cup \{(s_t, a_t, \boldsymbol{r}, \pi_{\boldsymbol{\theta}}(a_t|s_t))\}$
11:  end for
12:  Send trajectory $\tau$ to replay buffer
13: until end of training
```

C.2. Retrace for Multi-Objective Policy Evaluation

Recall that the goal of the policy evaluation stage (Sec. 4.1) is to learn the Q-function $Q_{\phi_k}(s,a)$ for each objective k, parametrized by ϕ_k. These Q-functions are with respect to the policy π_{old}. We emphasize that it is valid to use any off-the-shelf Q-learning algorithm, such as TD(0) (Sutton, 1988), to learn these Q-functions. We choose to use Retrace (Munos et al., 2016), described here.

Given a replay buffer $\mathcal{D}$ containing trajectory snippets $\tau = \{(s_0, a_0, \boldsymbol{r}_0, s_1), \dots, (s_T, a_T, \boldsymbol{r}_T, s_{T+1})\}$, where $\boldsymbol{r}_t$ denotes a reward vector $\{r_k(s_t,a_t)\}_{k=1}^N$ that consists of a scalar reward for each of N objectives, the Retrace objective is as follows:

$$\min_{\phi_k} \mathbb{E}_{\tau\sim\mathcal{D}}\Big[\big(Q_k^{ret}(s_t,a_t) - \hat{Q}_{\phi_k}(s_t,a_t)\big)^2\Big], \tag{15}$$

with

$$Q_k^{\text{ret}}(s_t,a_t) = \hat{Q}_{\phi'_k}(s_t,a_t) + \sum_{j=t}^{T} \gamma^{j-t}\Big(\prod_{z=t+1}^{j} c_z\Big)\delta^j,$$

$$\delta^j = r_k(s_j,a_j) + \gamma V(s_{j+1}) - \hat{Q}_{\phi'_k}(s_j,a_j),$$

$$V(s_{j+1}) = \mathbb{E}_{\pi_{\text{old}}(a|s_{j+1})}[\hat{Q}_{\phi'_k}(s_{j+1},a)].$$

Algorithm 3 MO-MPO: Asynchronous Learner

```
1: given batch size (L), number of actions (M), (N) Q-functions, (N) preferences $\{\epsilon_k\}_{k=1}^N$, policy networks, and replay buffer $\mathcal{D}$
2: initialize Lagrangians $\{\eta_k\}_{k=1}^N$ and $\nu$, target networks, and online networks such that $\pi_{\boldsymbol\theta'} = \pi_{\boldsymbol\theta}$ and $Q_{\phi'_k} = Q_{\phi_k}$
3: repeat
4:   repeat
5:     // Collect dataset $\{s^i, a^{ij}, Q_k^{ij}\}_{i,j,k}^{L,M,N}$, where
6:     // $s^i \sim \mathcal{D}, a^{ij} \sim \pi_{\boldsymbol\theta'}(a|s^i)$, and $Q_k^{ij} = Q_{\phi_k'}(s^i, a^{ij})$
7:
8:     // Compute action distribution for each objective
9:     for k = 1, ..., $N$ do
10:      $\delta_{\eta_k} \leftarrow \nabla_{\eta_k} \eta_k \epsilon_k + \eta_k \sum_i^L \frac{1}{L} \log\left(\sum_j^M \frac{1}{M} \exp\left(\frac{Q_k^{ij}}{\eta_k}\right)\right)$
11:      Update $\eta_k$ based on $\delta_{\eta_k}$
12:      $q_k^{ij} \propto \exp(\frac{Q_k^{ij}}{\eta_k})$
13:    end for
14:
15:    // Update parametric policy with trust region
16:    // sg denotes a stop-gradient
17:    $\delta_\pi \leftarrow -\nabla_{\boldsymbol\theta} \sum_i^L \sum_j^M \sum_k^N q_k^{ij} \log \pi_{\boldsymbol\theta}(a^{ij}|s^i)$
18:        $+\mathrm{sg}(\nu)\Big(\beta - \sum_i^L \mathrm{KL}(\pi_{\boldsymbol\theta'}(a|s^i) \,\|\, \pi_{\boldsymbol\theta}(a|s^i))\,\mathrm{d}s\Big)$
19:    $\delta_\nu \leftarrow \nabla_\nu \nu\Big(\beta - \sum_i^L \mathrm{KL}(\pi_{\boldsymbol\theta'}(a|s^i) \,\|\, \pi_{\mathrm{sg}(\boldsymbol\theta)}(a|s^i))\,\mathrm{d}s\Big)$
20:    Update $\pi_{\boldsymbol\theta}$ based on $\delta_\pi$
21:    Update $\nu$ based on $\delta_\nu$
22:
23:    // Update Q-functions
24:    for k = 1, ..., $N$ do
25:      $\delta_{\phi_k} \leftarrow \nabla_{\phi_k} \sum_{(s_t,a_t)\in\tau\sim\mathcal{D}} \left(\hat{Q}_{\phi_k}(s_t,a_t) - Q_k^{\mathrm{ret}}\right)^2$
26:                                    with $Q_k^{\mathrm{ret}}$ as in Eq. 15
27:      Update $\phi_k$ based on $\delta_{\phi_k}$
28:    end for
29:
30:  until fixed number of steps
31:  // Update target networks
32:  $\pi_{\boldsymbol\theta'} = \pi_{\boldsymbol\theta}, Q_{\phi_k'} = Q_{\phi_k}$
33: until convergence
```

The importance weights c_k are defined as

$$c_z = \min\left(1, \frac{\pi_{\text{old}}(a_z|s_z)}{b(a_z|s_z)}\right),$$

where $b(a_z|s_z)$ denotes the behavior policy used to collect trajectories in the environment. When $j = t$, we set $\left(\prod_{z=t+1}^{j} c_z\right) = 1$.

We also make use of a target network for each Q-function (Mnih et al., 2015), parametrized by ϕ'_k, which we copy from the online network ϕ_k after a fixed number of gradient steps on the Retrace objective (15).

C.3. Policy Fitting

Recall that fitting the policy in the policy improvement stage (Sec. 4.2.2) requires solving the following constrained optimization:

$$\pi_{\text{new}} = \operatorname*{argmax}_{\boldsymbol\theta} \sum_{k=1}^{N} \int_s \mu(s) \int_a q_k(a|s) \log \pi_{\boldsymbol\theta}(a|s)\,\mathrm{d}a\,\mathrm{d}s$$
$$\text{s.t.} \int_s \mu(s)\ \mathrm{KL}(\pi_{\text{old}}(a|s) \,\|\, \pi_{\boldsymbol\theta}(a|s))\,\mathrm{d}s < \beta\,. \quad (16)$$

We first write the generalized Lagrangian equation, i.e.

$$L(\boldsymbol\theta, \nu) = \sum_{k=1}^{N} \int_s \mu(s) \int_a q_k(a|s) \log \pi_{\boldsymbol\theta}(a|s)\,\mathrm{d}a\,\mathrm{d}s \quad (17)$$
$$+ \nu\Big(\beta - \int_s \mu(s)\ \mathrm{KL}(\pi_{\text{old}}(a|s) \,\|\, \pi_{\boldsymbol\theta}(a|s))\,\mathrm{d}s\Big),$$

where ν is the Lagrangian multiplier. Now we solve the following primal problem,

$$\max_{\boldsymbol\theta} \min_{\nu>0} L(\boldsymbol\theta, \nu)\,.$$

To obtain the parameters $\boldsymbol\theta$ for the updated policy, we solve for ν and $\boldsymbol\theta$ by alternating between 1) fixing ν to its current value and optimizing for $\boldsymbol\theta$, and 2) fixing $\boldsymbol\theta$ to its current value and optimizing for ν. This can be applied for any policy output distribution.

For Gaussian policies, we decouple the update rules to optimize for mean and covariance independently, as in (Abdolmaleki et al., 2018b). This allows for setting different KL bounds for the mean (β_μ) and covariance (β_Σ), which results in more stable learning. To do this, we first separate out the following two policies for mean and covariance,

$$\pi_{\boldsymbol\theta}^{\mu}(a|s) = \mathcal{N}\Big(a; \mu_{\boldsymbol\theta}(s), \Sigma_{\boldsymbol\theta_{old}}(s)\Big)\,, \quad (18)$$
$$\pi_{\boldsymbol\theta}^{\Sigma}(a|s) = \mathcal{N}\Big(a; \mu_{\boldsymbol\theta_{old}}(s), \Sigma_{\boldsymbol\theta}(s)\Big)\,. \quad (19)$$

Policy $\pi_{\boldsymbol\theta}^{\mu}(a|s)$ takes the mean from the online policy network and covariance from the target policy network, and policy $\pi_{\boldsymbol\theta}^{\Sigma}(a|s)$ takes the mean from the target policy network and covariance from online policy network. Now our optimization problem has the following form:

$$\max_{\boldsymbol\theta} \sum_{k=1}^{N} \int_s \mu(s) \int_a q_k(a|s)\Big(\log \pi_{\boldsymbol\theta}^{\mu}(a|s)\pi_{\boldsymbol\theta}^{\Sigma}(a|s)\Big)\,\mathrm{d}a\,\mathrm{d}s$$
$$\text{s.t.} \int_s \mu(s)\ \mathrm{KL}(\pi_{\text{old}}(a|s) \,\|\, \pi_{\boldsymbol\theta}^{\mu}(a|s))\,\mathrm{d}s < \beta_\mu$$
$$\int_s \mu(s)\ \mathrm{KL}(\pi_{\text{old}}(a|s) \,\|\, \pi_{\boldsymbol\theta}^{\Sigma}(a|s))\,\mathrm{d}s < \beta_\Sigma\,. \quad (20)$$

As in (Abdolmaleki et al., 2018b), we set a much smaller bound for covariance than for mean, to keep the exploration and avoid premature convergence. We can solve this optimization problem using the same general procedure as described above.

C.4. Derivation of Dual Function for η

Recall that obtaining the per-objective improved action distributions $q_k(a|s)$ in the policy improvement stage (Sec. 4.2.1) requires solving a convex dual function for the temperature η_k for each objective. For the derivation of this dual function, please refer to Appendix D.2 of the original MPO paper (Abdolmaleki et al., 2018b).

D. MO-V-MPO: Multi-Objective On-Policy MPO

In this section we describe how MO-MPO can be adapted to the on-policy case, in which a state-value function $V(s)$ is learned and used to estimate the advantages for each objective. We call this approach MO-V-MPO.

D.1. Multi-objective policy evaluation in the on-policy setting

In the on-policy setting, to evaluate the previous policy π_{old}, we use advantages $A(s,a)$ estimated from a learned state-value function $V(s)$, instead of a state-action value function $Q(s,a)$ as in the main text. We train a separate V-function for each objective by regressing to the standard n-step return (Sutton, 1988) associated with each objective. More concretely, given trajectory snippets $\tau = \{(s_0, a_0, \boldsymbol{r}_0), \ldots, (s_T, a_T, \boldsymbol{r}_T)\}$ where $\boldsymbol{r}_t$ denotes a reward vector $\{r_k(s_t, a_t)\}_{k=1}^N$ that consists of rewards for all N objectives, we find value function parameters ϕ_k by optimizing the following objective:

$$\min_{\phi_{1:k}} \sum_{k=1}^{N} \mathbb{E}_\tau \Big[\big(G_k^{(T)}(s_t, a_t) - V_{\phi_k}^{\pi_{\text{old}}}(s_t) \big)^2 \Big]. \tag{21}$$

Here $G_k^{(T)}(s_t, a_t)$ is the T-step target for value function k, which uses the actual rewards in the trajectory and bootstraps from the current value function for the rest: $G_k^{(T)}(s_t, a_t) = \sum_{\ell=t}^{T-1} \gamma^{\ell-t} r_k(s_\ell, a_\ell) + \gamma^{T-t} V_{\phi_k}^{\pi_{\text{old}}}(s_{\ell+T})$. The advantages are then estimated as $A_k^{\pi_{\text{old}}}(s_t, a_t) = G_k^{(T)}(s_t, a_t) - V_{\phi_k}^{\pi_{\text{old}}}(s_t)$.

D.2. Multi-objective policy improvement in the on-policy setting

Given the previous policy $\pi_{\text{old}}(a|s)$ and estimated advantages $\{A_k^{\pi_{\text{old}}}(s,a)\}_{k=1,\ldots,N}$ associated with this policy for each objective, our goal is to improve the previous policy. To this end, we first learn an improved variational distribution $q_k(s,a)$ for each objective, and then combine and distill the variational distributions into a new parametric policy $\pi_{\text{new}}(a|s)$. Unlike in MO-MPO, for MO-V-MPO we use the joint distribution $q_k(s,a)$ rather than local policies $q_k(a|s)$ because, without a learned Q-function, only one action per state is available for learning. This is a multi-objective variant of the two-step policy improvement procedure employed by V-MPO (Song et al., 2020).

D.2.1. Obtaining improved variational distributions per objective (Step 1)

In order to obtain the improved variational distributions $q_k(s,a)$, we optimize the RL objective for each objective A_k:

$$\max_{q_k} \int_{s,a} q_k(s,a)\, A_k(s,a)\, \mathrm{d}a\, \mathrm{d}s \tag{22}$$
$$\text{s.t. } \mathrm{KL}(q_k(s,a) \| p_{\text{old}}(s,a)) < \epsilon_k \,,$$

where the KL-divergence is computed over all (s,a), ϵ_k denotes the allowed expected KL divergence, and $p_{\text{old}}(s,a) = \mu(s)\pi_{\text{old}}(a|s)$ is the state-action distribution associated with π_{old}.

As in MO-MPO, we use these ϵ_k to define the preferences over objectives. More concretely, ϵ_k defines the allowed contribution of objective k to the change of the policy. Therefore, the larger a particular ϵ_k is with respect to others, the more that objective k is preferred. On the other hand, if $\epsilon_k = 0$ is zero, then objective k will have no contribution to the change of the policy and will effectively be ignored.

Equation (22) can be solved in closed form:

$$q_k(s,a) \propto p_{\text{old}}(s,a) \exp\left(\frac{A_k(s,a)}{\eta_k} \right), \tag{23}$$

where the temperature η_k is computed based on the constraint ϵ_k by solving the following convex dual problem

$$\eta_k = \operatorname*{argmin}_{\eta_k} \Bigg[\eta_k\, \epsilon_k + \tag{24}$$
$$\eta_k \log \int_{s,a} p_{\text{old}}(s,a) \exp\left(\frac{A_k(s,a)}{\eta} \right) \mathrm{d}a\, \mathrm{d}s \Bigg].$$

We can perform the optimization along with the loss by taking a gradient descent step on η_k, and we initialize with the solution found in the previous policy iteration step. Since η_k must be positive, we use a projection operator after each gradient step to maintain $\eta_k > 0$.

Top-k advantages. As in Song et al. (2020), in practice we used the samples corresponding to the top 50% of advantages in each batch of data.

D.2.2. Fitting a new parametric policy (Step 2)

We next want to combine and distill the state-action distributions obtained in the previous step into a single parametric policy $\pi_{\text{new}}(a|s)$ that favors all of the objectives according to the preferences specified by ϵ_k. For this we solve a supervised learning problem that fits a parametric policy as follows:

$$\pi_{\text{new}} = \arg\max_{\theta} \sum_{k=1}^{N} \int_{s,a} q_k(s,a) \log \pi_\theta(a|s) \,\mathrm{d}a\,\mathrm{d}s$$
$$\text{s.t.} \int_s \mathrm{KL}(\pi_{\text{old}}(a|s) \,\|\, \pi_\theta(a|s)) \,\mathrm{d}s < \beta\,, \tag{25}$$

where θ are the parameters of our function approximator (a neural network), which we initialize from the weights of the previous policy π_{old}, and the KL constraint enforces a trust region of size β that limits the overall change in the parametric policy, to improve stability of learning. As in MPO, the KL constraint in this step has a regularization effect that prevents the policy from overfitting to the local policies and therefore avoids premature convergence.

In order to optimize Equation (25), we employ Lagrangian relaxation similar to the one employed for ordinary V-MPO (Song et al., 2020).

E. Multi-Objective Policy Improvement as Inference

In the main paper, we motivated the multi-objective policy update rules from an intuitive perspective. In this section, we show that our multi-objective policy improvement algorithm can also be derived from the RL-as-inference perspective. In the policy improvement stage, we assume that a Q-function for each objective is given, and we would like to improve our policy with respect to these Q-functions. The derivation here extends the derivation for the policy improvement algorithm in (single-objective) MPO in Abdolmaleki et al. (2018a) (in appendix) to the multi-objective case.

We assume there are observable binary improvement events, $\{R_k\}_{k=1}^N$, for each objective. $R_k = 1$ indicates that our policy has improved for objective k, while $R_k = 0$ indicates that it has not. Now we ask, if the policy has improved with respect to *all* objectives, i.e., $\{R_k = 1\}_{k=1}^N$, what would the parameters $\boldsymbol{\theta}$ of that improved policy be? More concretely, this is equivalent to the maximum a posteriori estimate for $\{R_k = 1\}_{k=1}^N$:

$$\max_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(R_1 = 1, R_2 = 1, \dots, R_N = 1)\, p(\boldsymbol{\theta})\,, \tag{26}$$

where the probability of improvement events depends on $\boldsymbol{\theta}$. Assuming independence between improvement events R_k and using log probabilities leads to the following objective:

$$\max_{\boldsymbol{\theta}} \sum_{k=1}^{N} \log p_{\boldsymbol{\theta}}(R_k = 1) + \log p(\boldsymbol{\theta})\,. \tag{27}$$

The prior distribution over parameters, $p(\boldsymbol{\theta})$, is fixed during the policy improvement step. We set the prior such that $\pi_{\boldsymbol{\theta}}$ stays close to the target policy during each policy improvement step, to improve stability of learning (Sec. E.2).

We use the standard expectation-maximization (EM) algorithm to efficiently maximize $\sum_{k=1}^N \log p_{\boldsymbol{\theta}}(R_k = 1)$. The EM algorithm repeatedly constructs a tight lower bound in the E-step, given the previous estimate of $\boldsymbol{\theta}$ (which corresponds to the target policy in our case), and then optimizes that lower bound in the M-step. We introduce a variational distribution $q_k(s,a)$ per objective, and use this to decompose the log-evidence $\sum_{k=1}^N \log p_{\boldsymbol{\theta}}(R_k = 1)$ as follows:

$$\sum_{k=1}^{N} \log p_{\boldsymbol{\theta}}(R_k = 1) \tag{28}$$
$$= \sum_{k=1}^{N} \mathrm{KL}(q_k(s,a) \,\|\, p_{\boldsymbol{\theta}}(s,a|R_k = 1)) -$$
$$\sum_{k=1}^{N} \mathrm{KL}(q_k(s,a) \,\|\, p_{\boldsymbol{\theta}}(R_k = 1, s, a))\,.$$

The second term of this decomposition is expanded as:

$$\sum_{k=1}^{N} \mathrm{KL}(q_k(s,a) \,\|\, p_{\boldsymbol{\theta}}(R_k = 1, s, a)) \tag{29}$$
$$= \sum_{k=1}^{N} \mathbb{E}_{q_k(s,a)}\Big[\log \frac{p_{\boldsymbol{\theta}}(R_k = 1, s, a)}{q_k(s,a)}\Big]$$
$$= \sum_{k=1}^{N} \mathbb{E}_{q_k(s,a)}\Big[\log \frac{p(R_k = 1|s,a)\pi_{\boldsymbol{\theta}}(a|s)\mu(s)}{q_k(s,a)}\Big]\,,$$

where $\mu(s)$ is the stationary state distribution, which is assumed to be given in each policy improvement step. In practice, $\mu(s)$ is approximated by the distribution of the states in the replay buffer. $p(R_k = 1|s,a)$ is the likelihood of the improvement event for objective k, if our policy chose action a in state s.

The first term of the decomposition in Equation (28) is always positive, so the second term is a lower bound on the log-evidence $\sum_{k=1}^N \log p_{\boldsymbol{\theta}}(R_k = 1)$. $\pi_{\boldsymbol{\theta}}(a|s)$ and $q_k(a|s) = \frac{q_k(s,a)}{\mu(s)}$ are unknown, even though $\mu(s)$ is given. In the E-step, we estimate $q_k(a|s)$ for each objective k by minimizing the first term, given $\boldsymbol{\theta} = \boldsymbol{\theta}'$ (the parameters of the target policy). Then in the M-step, we find a new $\boldsymbol{\theta}$ by fitting the parametric policy to the distributions from first step.

E.1. E-Step

The E-step corresponds to the first step of policy improvement in the main paper (Sec. 4.2.1). In the E-step, we choose the variational distributions $\{q_k(a|s)\}_{k=1}^N$ such that the lower bound on $\sum_{k=1}^N \log p_{\boldsymbol{\theta}}(R_k = 1)$ is as tight as possible when $\boldsymbol{\theta} = \boldsymbol{\theta}'$, the parameters of the target policy. The lower bound is tight when the first term of the decomposition in Equation (28) is zero, so we choose q_k to minimize this term. We can optimize for each variational distribution q_k independently:

$$\begin{aligned} q_k(a|s) &= \operatorname*{argmin}_q \mathbb{E}_{\mu(s)}\Big[\mathrm{KL}(q(a|s) \,\|\, p_{\boldsymbol{\theta}'}(s,a|R_k=1))\Big] \\ &= \operatorname*{argmin}_q \mathbb{E}_{\mu(s)}\Big[\mathrm{KL}(q(a|s) \,\|\, \pi_{\boldsymbol{\theta}'}(a|s)) - \\ &\qquad \mathbb{E}_{q(a|s)} \log p(R_k = 1|s,a))\Big] . \end{aligned} \tag{30}$$

We can solve this optimization problem in closed form, which gives us

$$q_k(a|s) = \frac{\pi_{\boldsymbol{\theta}'}(a|s)\, p(R_k = 1|s,a)}{\int \pi_{\boldsymbol{\theta}'}(a|s)\, p(R_k = 1|s,a)\,\mathrm{d}a} .$$

This solution weighs the actions based on their relative improvement likelihood $p(R_k = 1|s,a)$ for each objective. We define the likelihood of an improvement event R_k as

$$p(R_k = 1|s,a) \propto \exp\left(\frac{Q_k(s,a)}{\alpha_k}\right),$$

where α_k is an objective-dependent temperature parameter that controls how greedy the solution $q_k(a|s)$ is with respect to its associated objective $Q_k(s,a)$. For example, given a batch of state-action pairs, at the extremes, an α_k of zero would give all probability to the state-action pair with the maximum Q-value, while an α_k of positive infinity would give the same probability to all state-action pairs. In order to automatically optimize for α_k, we plug the exponential transformation into (30), which leads to

$$\begin{aligned} q_k(a|s) = \operatorname*{argmax}_q \int \mu(s) \int q(a|s) Q_k(s,a)\,\mathrm{d}a\,\mathrm{d}s - \\ \alpha_k \int \mu(s)\,\mathrm{KL}(q(a|s) \,\|\, \pi_{\boldsymbol{\theta}'}(a|s))\,\mathrm{d}s . \end{aligned} \tag{31}$$

If we instead enforce the bound on KL divergence as a hard constraint (which we do in practice), that leads to:

$$\begin{aligned} q_k(a|s) = \operatorname*{argmax}_q \int \mu(s) \int q(a|s) Q_k(s,a)\,\mathrm{d}a\,\mathrm{d}s \\ \text{s.t.} \int \mu(s)\,\mathrm{KL}(q(a|s) \,\|\, \pi_{\boldsymbol{\theta}'}(a|s))\,\mathrm{d}s < \epsilon_k, \end{aligned} \tag{32}$$

where, as described in the main paper, the parameter ϵ_k defines preferences over objectives. If we now assume that $q_k(a|s)$ is a non-parametric sample-based distribution as in (Abdolmaleki et al., 2018b), we can solve this constrained optimization problem in closed form for each sampled state s, by setting

$$q(a|s) \propto \pi_{\boldsymbol{\theta}'}(a|s) \exp\left(\frac{Q_k(s,a)}{\eta_k}\right), \tag{33}$$

where η_k is computed by solving the following convex dual function:

$$\begin{aligned} \eta_k = \operatorname*{argmin}_\eta \eta\epsilon_k + \\ \eta \int \mu(s) \log \int \pi_{\boldsymbol{\theta}'}(a|s) \exp\left(\frac{Q_k(s,a)}{\eta}\right) \mathrm{d}a\,\mathrm{d}s . \end{aligned} \tag{34}$$

Equations (32), (33), and (34) are equivalent to those given for the first step of multi-objective policy improvement in the main paper (Sec. 4.2.1). Please see refer to Appendix D.2 in (Abdolmaleki et al., 2018b) for details on derivation of the dual function.

E.2. M-Step

After obtaining the variational distributions $q_k(a|s)$ for each objective k, we have found a tight lower bound for $\sum_{k=1}^N \log p_{\boldsymbol{\theta}}(R_k = 1)$ when $\boldsymbol{\theta} = \boldsymbol{\theta}'$. We can now obtain the parameters $\boldsymbol{\theta}$ of the updated policy $\pi_{\boldsymbol{\theta}}(a|s)$ by optimizing this lower bound. After rearranging and dropping the terms in (27) that do not involve $\boldsymbol{\theta}$, we obtain

$$\begin{aligned} \boldsymbol{\theta}^* = \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{k=1}^N \int \mu(s) \int q_k(a|s) \log \pi_{\boldsymbol{\theta}}(a|s)\,\mathrm{d}a\,\mathrm{d}s \\ + \log p(\boldsymbol{\theta}). \end{aligned}$$

This corresponds to the supervised learning step of policy improvement in the main paper (Sec. 4.2.2).

For the prior $p(\boldsymbol{\theta})$ on the parameters $\boldsymbol{\theta}$, we enforce that the new policy $\pi_{\boldsymbol{\theta}}(a|s)$ is close to the target policy $\pi_{\boldsymbol{\theta}'}(a|s)$. We therefore optimize for

$$\begin{aligned} \boldsymbol{\theta}^* = \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{k=1}^N \int \mu(s) \int q_k(a|s) \log \pi_{\boldsymbol{\theta}}(a|s)\,\mathrm{d}a\,\mathrm{d}s \\ \text{s.t.} \int \mu(s)\,\mathrm{KL}(\pi_{\boldsymbol{\theta}'}(a|s) \,\|\, \pi_{\boldsymbol{\theta}}(a|s))\,\mathrm{d}s < \beta. \end{aligned}$$

Because of the constraint on the change of the parametric policy, we do not greedily optimize the M-step objective. See (Abdolmaleki et al., 2018b) for more details.