Appendices

A. Hyperparameters Description

Table 1. Hyperparameters used for Predator-Prey, Keep-away and Physical Deception

Hyperparameter	MERL	MATD3/MADDPG
Population size	10	N/A
Rollout size	10	10
Target weight	0.01	0.01
Actor Learning Rate	0.01	0.01
Critic Learning Rate	0.01	0.01
Discount Factor	0.95	0.95
Replay Buffer Size	$1e^6$	$1e^6$
Batch Size	1024	1024
Mutation Prob	0.9	N/A
Mutation Fraction	0.1	N/A
Mutation Strength	0.1	N/A
Super Mutation Prob	0.05	N/A
Reset Mutation Prob	0.05	N/A
Number of elites	4	N/A
Exploration Policy	$\mathcal{N}(0, \sigma)$	$\mathcal{N}(0, \sigma)$
Exploration Noise	0.4	0.4
Rollouts per fitness	10	N/A
Actor Architecture	$[100, 100]$	$[100, 100]$
Critic Architecture	$[100, 100]$	$[300, 300]$
TD3 Noise Variance	0.2	0.2
TD3 Noise Clip	0.5	0.5
TD3 Update Freq	2	2

Table 1 details the hyperparameters used for MERL, MATD3, and MADDPG in tackling predator-prey and cooperative navigation. The hyperparmaeters were inherited from (Lowe et al., 2017) to match the original experiments for MADDPG and MATD3. The only exception to this was the use of hyperbolic tangent instead of Relu activation functions.

Table 2 details the hyperparameters used for MERL, MATD3, and MADDPG in the rover domain. The hyperparameters themselves are defined below:

- **Optimizer = Adam**
  Adam optimizer was used to update both the actor and critic networks for all learners.
- **Population size** M
  This parameter controls the number of different actors (policies) that are present in the evolutionary population.
- **Rollout size**
  This parameter controls the number of rollout workers (each running an episode of the task) per generation.

  Note: The two parameters above (population size k and rollout size) collectively modulates the proportion of exploration carried out through noise in the actor's *parameter* space and its *action* space.

Table 2. Hyperparameters used for Rover Domain

Hyperparameter	MERL	MATD3/MADDPG
Population size	10	N/A
Rollout size	50	50
Target weight	$1e^{-5}$	$1e^{-5}$
Actor Learning Rate	$5e^{-5}$	$5e^{-5}$
Critic Learning Rate	$1e^{-5}$	$1e^{-5}$
Discount Factor	0.5	0.97
Replay Buffer Size	$1e^5$	$1e^5$
Batch Size	512	512
Mutation Prob	0.9	N/A
Mutation Fraction	0.1	N/A
Mutation Strength	0.1	N/A
Super Mutation Prob	0.05	N/A
Reset Mutation Prob	0.05	N/A
Number of elites	4	N/A
Exploration Policy	$\mathcal{N}(0, \sigma)$	$\mathcal{N}(0, \sigma)$
Exploration Noise σ	0.4	0.4
Rollouts per fitness ξ	10	N/A
Actor Architecture	$[100, 100]$	$[100, 100]$
Critic Architecture	$[100, 100]$	$[300, 300]$
TD3 Noise variance	0.2	0.2
TD3 Noise Clip	0.5	0.5
TD3 Update Frequency	2	2

- **Target weight** τ
  This parameter controls the magnitude of the soft update between the actors and critic networks, and their target counterparts.
- **Actor Learning Rate**
  This parameter controls the learning rate of the actor network.
- **Critic Learning Rate**
  This parameter controls the learning rate of the critic network.
- **Discount Rate**
  This parameter controls the discount rate used to compute the return optimized by policy gradient.
- **Replay Buffer Size**
  This parameter controls the size of the replay buffer. After the buffer is filled, the oldest experiences are deleted in order to make room for new ones.

- **Batch Size**
  This parameters controls the batch size used to compute the gradients.
- **Actor Activation Function**
  Hyperbolic tangent was used as the activation function.
- **Critic Activation Function**
  Hyperbolic tangent was used as the activation function.
- **Number of Elites**
  This parameter controls the fraction of the population that are categorized as elites. Since an elite individual (actor) is shielded from the mutation step and preserved as it is, the elite fraction modulates the degree of exploration/exploitation within the evolutionary population.
- **Mutation Probability**
  This parameter represents the probability that an actor goes through a mutation operation between generation.
- **Mutation Fraction**
  This parameter controls the fraction of the weights in a chosen actor (neural network) that are mutated, once the actor is chosen for mutation.
- **Mutation Strength**
  This parameter controls the standard deviation of the Gaussian operation that comprises mutation.
- **Super Mutation Probability**
  This parameter controls the probability that a super mutation (larger mutation) happens in place of a standard mutation.
- **Reset Mutation Probability**
  This parameter controls the probability a neural weight is instead reset between $\mathcal{N}(0, 1)$ rather than being mutated.
- **Exploration Noise**
  This parameter controls the standard deviation of the Gaussian operation that comprise the noise added to the actor's actions during exploration by the learners (learner roll-outs).
- **TD3 Policy Noise Variance**
  This parameter controls the standard deviation of the Gaussian operation that comprise the noise added to the policy output before applying the Bellman backup. This is often referred to as the magnitude of policy smoothing in TD3.
- **TD3 Policy Noise Clip**
  This parameter controls the maximum norm of the policy noise used to smooth the policy.
- **TD3 Policy Update Frequency**
  This parameter controls the number of critic updates per policy update in TD3.

B. Expected Selection Rate

We use a multi-step selection process inside MERL. First we select e top-performing individuals as elites sequentially from the population without replacement. Then we conduct a tournament selection (Miller and Goldberg, 1995) with tournament size t with replacement from the entire population including the elites. t is set to 3 in this paper. The candidates selected from the tournament selection are pruned for duplicates and the resulting set is carried over as the offsprings for this generation. The combined set of offsprings and the elites represent the candidates selected for that generation of evolution.

The expected selection rate $P(s)$ is defined as the probability of selection for an individual if the selection was random. This is equivalent to conducting selection using a random ranking of the population where the fitness scores were ignored and a random number was assigned as an individual's fitness. Note that the selection rate is **not** the probability of selection for a random policy (individual with random neural network weights) inserted into the evolutionary population. The probability of selection for such a random policy would be extremely low as the other individuals in the population would be ranked significantly higher.

In order to compute the expected selected rate, we need to compute it for the elite set and the offspring set. The expected selection rate for the elite set is given by e/M. The expected selection rate for the offspring set involves multiple rounds of tournament selection with replacement followed by pruning of duplicates to compute the combined set along with the elites. We computed this expectation empirically using an experiment with 1000000 iterations and found the expected selection rate to be 0.47.

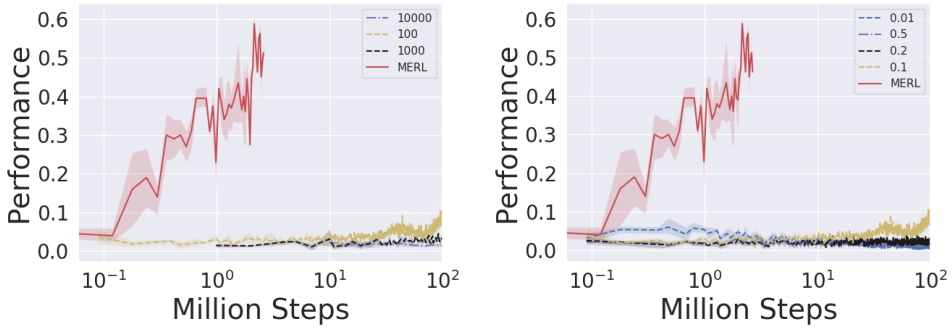
C. Extended EA Benchmarks

We conducted some additional experiments by varying the population sizes used for the EA baseline. The purpose of these experiments is to investigate if larger population sizes (as is the norm for EA algorithms) can alleviate the need for policy-gradient module within MERL.

Additionally, we also investigated Evolutionary Strategies (ES) (Salimans et al., 2017), which has been widely adopted in the community in recent years. We perform hyperparameter sweeps to tune this baseline. All results are reported in the rover domain with a coupling of 3.

C.1. Evolutionary Strategies (ES)

ES Population Sweep: Figure 11(left) compares ES with varying population sizes in the rover domain with a coupling of 3. Sigma for all ES runs are set at 0.1. Among the ES runs, a population size of 100 yields the best results converging to



(a) ES Population Sweep (b) ES Sigma Sweep

Figure 11. (a) Evolutionary Strategies population size sweep on the rover domain with a coupling of 3. (b)Evolutionary Strategies Noise magnitude (sigma) sweep on the rover domain with a coupling of 3

0.1 in 100-millions frames. MERL (red) on the other hand is ran for 2-million frames and converges to 0.48.

ES Noise Sweep: Apart from the population size, a key hyperparameter for ES is the variance of the perturbation factor (sigma). We run a parameter sweep for sigma and report results in Figure 11(right). We do not see a great deal of improvement with the change of sigma.

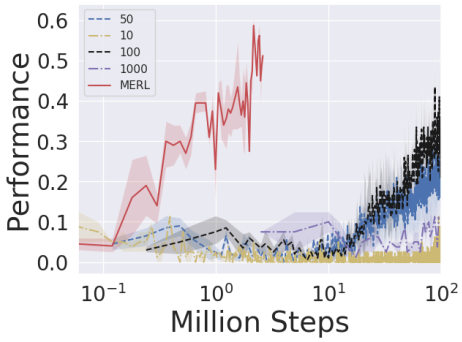


Figure 12. Evolutionary Algorithm Population size sweep on the rover domain with a coupling of 3. MERL was run for 2-million steps while the other EA runs were ran for 100-million steps.

C.2. EA Population Size

Next, we conduct an experiment to evaluate the efficacy of different population sizes from 10-1, 000 for the Evolutionary algorithm used in the paper. All results are reported for the rover domain with a coupling factor of 3 and are illustrated in Figure 12. The best EA performance was found for a population size of 100 reaching 0.3 in 100-million time steps. Compare this to MERL which reaches a performance of 0.48 in only 2 million time steps. This demonstrates a key thesis behid MERL - the efficacy of the guided evolution approach over purely evolutionary approaches.

D. Rollout Methodology

Algorithm 2 describes an episode of rollout under MERL detailing the connections between the local reward, global reward, and the associated replay buffer.

Algorithm 2 Rollout

function Rollout(π, $\mathcal{R}$, noise, ξ)
  $fitness = 0$
  for j = 1:ξ **do**
    Reset environment and get initial joint state js
    while env is not done **do**
      Initialize an empty list of joint action ja = []
      for Each agent (actor head) $\pi^k \in \pi$ and s_k *in* js **do**
        $ja \Leftarrow ja \cup \pi^k(s_k|\theta^{\pi^k}) + noise_t$
      end for
      Execute ja and observe joint local reward jl, global reward g and joint next state js'
      for Each Replay Buffer $\mathcal{R}_k \in \mathcal{R}$ and s_k, a_k, l_k, s'_k in js, ja, jl, js' **do**
        Append transition (s_k, a_k, l_k, s'_k) to R_k
      end for
      $js = js'$
      if env is done: **then**
        $fitness \leftarrow g$
      end if
    end while
  end for
  Return $\frac{fitness}{\xi}$, $\mathcal{R}$
end function