**Exploration Strategy:**

* **Current Issue**: You're using Gaussian noise (GaussianExploration) to explore, but it may not be enough for stable exploration in a TD3 setup, especially when the environment is complex or has sparse rewards.
* **Solution**: You could try adjusting the noise parameters (e.g., higher max\_sigma or adjust min\_sigma), or explore other exploration techniques like Ornstein-Uhlenbeck noise, which is commonly used in TD3.

**Target Networks Update Frequency:**

* **Current Issue**: You're updating the target networks at every step (via soft\_update), which might be too frequent and cause instability.
* **Solution**: Update the target networks less frequently (e.g., every 2 to 5 iterations) to allow the Q-values to stabilize before transferring information to the target networks.

**Replay Buffer Size:**

* **Current Issue**: Your replay buffer is large (1000000), but it may not be fully utilized depending on how quickly new experiences are added. Sometimes, larger replay buffers may cause slower learning.
* **Solution**: Consider reducing the replay buffer size, or ensure the buffer has a good mix of exploration and exploitation. A common approach is to sample experiences more efficiently.
  + You can also prioritize experiences based on the TD-error to enhance learning from important transitions, which is implemented in **Prioritized Experience Replay**.

**Gradient Clipping:**

* **Current Issue**: Unstable gradients may lead to large weight updates, which can cause divergence.
* **Solution**: Apply **gradient clipping** to stabilize the training process, especially for the policy and value networks. You can clip gradients during the backward pass:

Try code:

torch.nn.utils.clip\_grad\_norm\_(policy\_net.parameters(), max\_norm=0.5)

**Learning Rate:**

* **Current Issue**: The learning rate for the actor (lr\_actor = 0.0003) and critic (lr\_critic = 0.001) might not be optimal for your specific problem.
* **Solution**: Try experimenting with different learning rates, possibly using **learning rate schedules** (decay over time) to allow the model to learn more effectively. Using a learning rate scheduler like torch.optim.lr\_scheduler can help fine-tune the learning process.

**Batch Normalization:**

* **Current Issue**: The model may not be sufficiently regularized or normalized, leading to instability.
* **Solution**: Adding **batch normalization** layers or **layer normalization** can improve the stability of the learning process, especially when training deep networks.

-------------------------------------------------

**1. Lowering the Exploration Noise:**

* **Noise Decay**: You are using Ornstein-Uhlenbeck noise, which has a decay rate over time. If the noise decay period is too long, the exploration may remain too high for too long, leading to erratic behavior.
* **Impact of High Exploration**: If the noise is too large, the agent might not learn meaningful patterns in the environment, as the actions taken are too random. Lowering the standard deviation (noise\_std) or the clipping value (noise\_clip) may encourage more deterministic action selection, enabling faster convergence.

**2. Hyperparameters to Check:**

* **Learning Rates**: If the learning rates for the actor (lr\_actor) or critic (lr\_critic) are too high, it can cause instability in training. In many cases, lowering the learning rate helps to reduce the oscillations and improve convergence.
* **Replay Buffer and Priority Sampling**: Make sure your replay buffer is appropriately sized and the prioritization is working as intended. Too much noise in the prioritized sampling can cause unstable training.
* **Target Network Updates**: The soft\_tau used for updating the target networks is often a small value (e.g., 1e-2). If this is too high, it could lead to the target networks drifting too far from the main networks, making training unstable.

**3. Exploration vs. Exploitation:**

* **Exploration Decay**: Consider introducing a more gradual decay to the exploration noise. You might start with a higher noise for exploration and then gradually decrease it as the agent becomes more confident in its actions.
* **Early vs. Late Training**: In the beginning, noise should be high enough to encourage exploration, but as the agent progresses, the noise should gradually reduce to allow the learned policy to dominate.

**4. Improving Stability:**

* **Action Normalization**: You are using normalized actions; ensure that this normalization process is done correctly for both exploration noise and the network output.
* **Critic Network Updates**: You are updating both value\_net1 and value\_net2. Make sure both critics are being updated in tandem and there is no significant difference between them, as this could introduce instability.
* **Target Network Smoothing**: You are doing soft updates to the target networks, which is important for stability. Make sure the soft\_tau value is not too large, as this will cause instability when propagating target values.

**5. Considerations:**

* If after lowering the exploration noise you still face issues, experiment with different values of gamma (discount factor), batch\_size, and other network architecture parameters like the number of hidden units or layers.
* Make sure your environment is sufficiently challenging but not too complex, as this can affect the learning dynamics.