Path Planning Using Reinforcement Learning and Neural Radiance Fields

Gaurav Kothamachu Harish
Northeastern University
360 Huntington Ave, Boston, MA 02115
kothamachuharish.g@northeastern.edu

Abstract

Deep Reinforcement Learning (DRL) has been adopted in diverse applications due to its ability to inherently learn optimal policies in environments with high-dimensional states and actions. Deep Deterministic Policy Gradient is a DRL algorithm that uses an actor-critic architecture. The actor learns which action to take in a given state, and the critic evaluates those actions. This manuscript explores the implementation of DDPG and its performance in various tasks performed by a 7 DoF Robotic Manipulator. We also discuss ways to improve the performance of DDPG. Experimental results demonstrate that our method achieves faster convergence and more stable learning compared to standard DDPG implementations. The architecture's effectiveness is particularly evident in dense reward settings, where it achieves consistent performance with reduced training time and computational resources. In the process of improving DDPG behavior, we also compare the performance against PPO implemented by Stable-Baselines3 and trained using RL Baseline3 Zoo. We also explore the use of Neural Radiance Fields for generating Latent Space in a 3D environment. Our work also compares the performance of DDPG with and without Neural Radiance Fields. We also explore the use of the Whale Optimization Algorithm for hyperparameter tuning. The manuscript highlights the role of the above algorithms in the Reinforcement Learning training process and its consequential impact on the quality of the solution. The results show that the proposed method outperforms the standard DDPG implementation in terms of convergence speed and solution quality. The proposed method is also more robust to local minima and achieves better performance in complex path planning scenarios.

Code — https://github.com/khgaurav/RL_DDPG_Nerf

Introduction

Robots are widely deployed across numerous industries, including picking up and placing objects, welding, surgery, agriculture, and many more. Industrial robots operate in complex environments with inherent uncertainties. Path planning is a problem that is fundamental in the goal of achieving autonomy in robots. Traditional methods like A* and Dijkstra's are very fast but are not suitable for complex and high-dimensional state spaces. Alternatives like inverse kinematics(Agrawal, Garimella, and Desmier 1996) are simple and satisfy angular momentum constraints, but the unpredictable emergence of dynamic environments frequently prevent it from ensuring the best path, which can impair manipulator performance. Deep Reinforcement Learning (DRL) methods have shown promise in solving such problems. Recent advances in control systems have taken advantage of machine learning, with particular emphasis on Reinforcement Learning (RL), to address complex control challenges. RL introduces an innovative approach in which agents develop decision-making policies in dynamic and uncertain environments through iterative interactions. The system operates by allowing agents to execute state-altering commands on the robotic manipulator, receiving reward-based feedback within a Markov Decision Process (MDP) framework, independent of predetermined samples or rules.

DRL algorithms have been shown to be effective in addressing path planning challenges in continuous action space(Al Ali, Shi, and Zhu 2024). This paper presents an optimized implementation of the Deep Deterministic Policy Gradient (DDPG) algorithm for robotic arm path planning. The proposed method is evaluated on a 7 DoF robotic manipulator in a 3D environment performing various tasks. Our approach combines several innovative elements: a prioritized experience replay buffer for efficient learning, the Whale Optimization Algorithm (WOA) for hyper-parameter tuning, and a dense reward structure based on goal distance metrics. Although traditional DDPG implementations often struggle with convergence and local minima issues in path planning scenarios, our methodology addresses these challenges through careful architectural choices and parameter optimization. We also compare the integration of NeRF for environmental mapping provides a rich representation of the 3D space, enabling more sophisticated path planning capabilities than conventional approaches.

Background

Markov Decision Processes (MDP): An MDP is the traditional setting used for reinforcement learning, providing classical formalizations of sequential decision-making where actions taken by a decision-making agent can impact not only immediate, but also future rewards. MDPs solve the issue of learning through interactions to achieve a goal, where the problem arises from a series of interactions between an agent and an environment. MDP is represented as the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, H, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T}$ is the transition function $\mathcal{T}(s_{t+1}|s_t, a_t)$, $\mathcal{R}$ is the reward function $\mathcal{R}(s_t, a_t, s_{t+1})$, H is the horizon, and γ is the discount factor. More specifically, in a finite horizon case, at each time step over a set of discrete time steps $t = 0, 1, 2, 3, ..., H-1$, the agent receives a state s_t from the environment and based on that state selects an action a_t. Upon attempting this action, the environment returns a reward r_{t+1} based on the reward function and transitions to a next state s_{t+1} based on the transition function. The objective of the agent is to learn to take the best actions (a policy, π) to maximize the expected sum of discounted rewards $E_\pi[\sum_{t=0}^{H} \gamma^t r_t]$[1].

Deep Deterministic Policy Gradient (DDPG) : DDPG is an off-policy actor-critic algorithm designed for continuous action spaces. Unlike traditional actor-critic methods, DDPG employs deterministic policies and incorporates several key innovations from Deep Q-Networks (DQN). The algorithm maintains four neural networks: an actor network, a critic network, and their corresponding target networks used for stable learning. The critic evaluates actions using the Q-function while the actor aims to maximize this Q-value through gradient ascent.

Actor-Critic Methods : Actor-critic methods represent a fundamental architecture in reinforcement learning that combines value-based and policy-based approaches. The architecture consists of two main components working in tandem: an actor and a critic(Haarnoja et al. 2018).
The actor is responsible for learning and executing the policy $\pi_\theta(a|s)$, which maps states to actions. In DDPG, the actor implements a deterministic policy $\mu_\theta(s)$ that directly outputs actions rather than a probability distribution. The actor network is trained to maximize the expected return by following the policy gradient:

$$\nabla_\theta J(\theta) = E_{s\sim\rho^\beta}[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)}]$$

The critic evaluates the actor's actions by learning a Q-value function $Q(s, a)$ that estimates the expected cumulative reward. It is trained using temporal difference learning to minimize the Bellman error:

$$L(\phi) = E_{(s,a,r,s')\sim D}[(Q_\phi(s, a)-(r+\gamma Q_{\phi'}(s', \mu_{\theta'}(s'))))^2]$$

where ϕ' and θ' represent target network parameters.
The actor and critic are implemented as deep neural networks to handle high-dimensional state spaces. The networks typically consist of:

- Multiple hidden layers with ReLU activations
- Batch normalization for stable training
- Target networks updated using soft updates with parameter τ: $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$

This deep learning architecture enables the algorithm to learn complex non-linear policies while maintaining stable training through experience replay and target networks.

Prioritized Experience Replay : Experience replay is a fundamental mechanism in reinforcement learning that enables agents to store and reuse past experiences for more efficient learning. Each experience is stored as a transition tuple (s_t, a_t, r_t, s_{t+1}) in a replay buffer. Prioritized experience replay (PER) enhances this basic mechanism by intelligently sampling transitions based on their temporal-difference (TD) error magnitude, ensuring that important experiences are replayed more frequently during training. The sampling probability for a transition i is given by $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$ where p_i represents the priority of transition i and α controls the degree of prioritization. To correct for the bias introduced by non-uniform sampling, importance sampling weights are applied using $w_i = (\frac{1}{N} \cdot \frac{1}{P(i)})^\beta$ where β is annealed from an initial value to 1 over time. This prioritization scheme helps accelerate learning by focusing the agent's updates on the most informative transitions while maintaining stable training through bias correction.

Whale Optimization Algorithm : The Whale Optimization Algorithm is a nature-inspired optimization technique that models the hunting behavior of humpback whales to solve complex optimization problems(Mirjalili and Lewis 2016). In DDPG hyperparameter optimization, WOA operates through three main mechanisms:

- Encircling Prey (Exploitation): When whales detect prey (local optimal), they encircle it (hyperparameters are varied nearby) using:

  $$D = |C \cdot X^*(t) - X(t)|, X(t+1) = X^*(t) - A \cdot D$$

  where X^* is the current best solution, X is the current solution, A and C are coefficient vectors which keep decreasing.

- Bubble-Net Attacking (Intensification): This mechanism mimics the spiral updating position of whales:

  $$X(t+1) = D \cdot e^{bl} \cdot \cos(2\pi l) + X^*(t)$$

  where b defines the spiral shape, l is a random number in $[-1, 1]$, D represents the distance between the whale and prey
- Search for Prey (Exploration): During exploration, whales search randomly using large values of $|A|$ (when $|A| > 1$), while during exploitation they use small values of $|A|$ (when $|A| < 1$).

Neural Radiance Fields (NeRF) Neural Radiance Fields (NeRF) represent a groundbreaking deep learning method for reconstructing three-dimensional scenes from two-dimensional images (Driess et al. 2022). The technique employs a neural network that maps spatial coordinates (x, y, z) and viewing directions (θ, ϕ) to color and volume density values. At its core, NeRF uses a multilayer perceptron (MLP) architecture to learn the scene's geometry and appearance, enabling the generation of novel viewpoints through volume rendering. The network learns both the spatial distribution of objects and their view-dependent appearance properties, allowing for photorealistic rendering of complex scenes. Through ray casting and volume rendering techniques, NeRF can synthesize new views by sampling points along camera rays and accumulating color and density predictions. This approach has revolutionized 3D scene representation by offering a compact, continuous representation that can reconstruct high-fidelity scenes from a limited set of input images.

Related Work

Classical Path Planning : Traditional path planning methods like A* and Dijkstra's algorithm provide deterministic solutions for robot navigation. While these methods are computationally efficient for low-dimensional spaces, they become intractable for high-dimensional robotic manipulators due to the curse of dimensionality. These graph-based approaches also struggle with continuous action spaces and dynamic environments.

Proximal Policy Optimization (PPO) : Proximal Policy Optimization (PPO) is a policy gradient method that addresses the limitations of traditional policy gradient algorithms by introducing a surrogate objective function. PPO optimizes the policy by maximizing the probability ratio between new and old policies while constraining the policy update to a specified threshold. This approach ensures stable learning by preventing large policy updates that can lead to catastrophic forgetting. While PPO offers more stable training through trust region optimization and clipped objective functions, it's on-policy nature makes it less sample efficient, which is crucial when training physical robotic systems. While we include PPO comparisons using Stable-Baselines3, our focus remained on DDPG due to its better sample efficiency

Particle Swarm Optimization (PSO) Alternative approaches like Particle Swarm Optimization (PSO) and singular-free terminal sliding mode control present optimization solutions for robotic path planning, but face significant practical limitations. PSO algorithms, while effective for static path planning, struggle with premature convergence and computational inefficiency in dynamic environments(Yao et al. 2024). Terminal sliding mode control approaches offer robust solutions for trajectory tracking but require substantial computational resources for real-time implementation and heavily depend on prior knowledge of system uncertainties. These methods, though theoretically sound, often sacrifice real-time performance for optimization quality, making them less suitable for robotic applications requiring quick decision-making under uncertain conditions

Project description

The first phase of the project focused on implementing and optimizing the Deep Deterministic Policy Gradient (DDPG) algorithm for path planning of robotic arms in 3D environments. DDPG algorithm is a no model, off-policy actor-critic algorithm that can learn policies in high-dimensional and continuous action space which is necessary when the actions available are the joint angles.

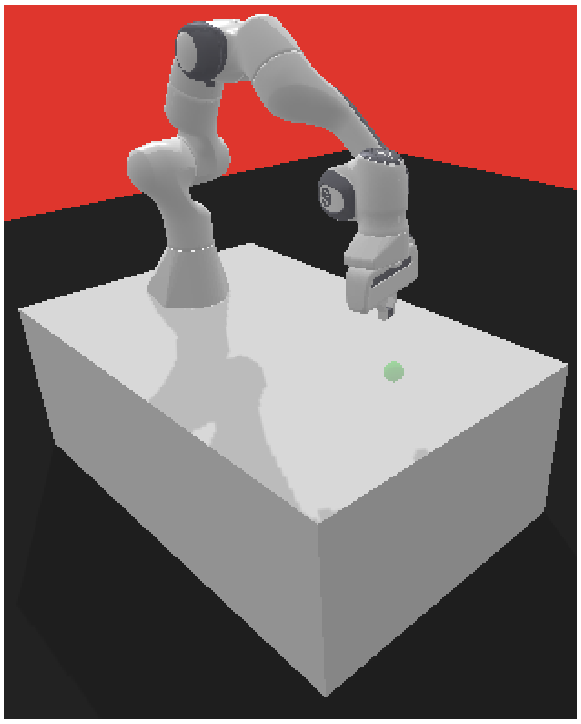
Figure 1: Franka Emika Panda Robot Environment

Simulator : Initially for constructing and enhancing DDPG algorithm, the Franka Emika Panda 7-DoF Manipulator was used(1). An gym environment(Gallouédec et al. 2021) already exists for this robot with predefined tasks. The below experiment was done for the PandaReach task where the goal for the end effector to reach a target position in 3D space.

The second environment used was the Meta-World Drawer-Open-v2 task(Otto et al.). It is a robotic manipulation task within the Meta-World benchmark suite, featuring a simulated Sawyer robotic arm that must learn to open a drawer. The robot must successfully grasp and pull open a drawer, with the drawer's initial position being randomized at the start of each episode. An episode is considered successful when the handle is less than 3cm from the target position or the gripper is less than 3cm from the handle5

Environment

- **State Space**:
  - Joint angles of Robotic Manipulator:
$$q = [q_1, q_2, q_3, q_4] \in R^4$$
  - End-effector position:
$$p = [x, y, z] \in R^3$$
  - Target position:
$$p_t = [x_t, y_t, z_t] \in R^3$$
- **Action Space**: The action space consists of the joint angles of the robot.
$$a = [\Delta q_1, \Delta q_2, \Delta q_3, \Delta q_4] \in R^4$$

where Δq_i is the change in joint angle q_i.

- **Reward Function**: +1 reward will be provided when the end effector p reaches the target position p_t. 0 reward will be given in all other steps

Neural Network Architecture

Actor Network: The actor network maps states to deterministic actions with the following structure:

- Input Layer: Takes state dimensions representing joint angles, end-effector position and target position.
- Hidden Layers: Two fully connected layers [256, 400] neurons with ReLU activation
- Output Layer: Produces action dimension with tanh activation to bound actions in [-1, 1]
- Xavier initialization for all weights

Critic Network: The critic network evaluates state-action pairs through a specialized architecture:

- State Processing Branch: Initial layer processes state input separately
- Action Integration: Action input is concatenated after the first hidden layer
- Hidden Dimensions: [256, 400] neurons with ReLU activation
- Output: Single neuron producing Q-value estimation

The implementation employs the Adam optimizer with distinct learning rates for the actor and critic networks, ensuring optimal parameter updates for each network independently. Target network parameters are smoothly updated using a soft update mechanism with a τ parameter, calculated as

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

, where θ' represents target network parameters and θ represents the current network parameters. To maintain training stability, batch normalization is applied throughout the network layers, normalizing the inputs to each layer and reducing internal covariate shift. Additionally, gradient clipping is implemented to prevent gradient explosions during back-propagation, ensuring stable learning even with large parameter updates.

DDPG Algorithm

DDPG algorithm (2) was implemented based on the original paper(Lillicrap et al. 2019). The environment state will be observed and preprocessed. The actor network selects an action, where $\mathcal{N}$ is Ornstein-Uhlenbeck noise with parameters:

$$a_t = \mu_\theta(s_t) + \mathcal{N}_t$$

The first improvement made to the original paper was to use a Prioritized Replay Buffer. This is implemented using TorchRL's specialized buffer and LazyMemmapStorage for efficient memory management. The replay buffer stores the transitions (s_t, a_t, r_t, s_{t+1}). The transitions are sampled based on their temporal difference (TD) error, with higher priority given to transitions with larger errors. Priority is calculated using:

$$p_i = |\delta_i| + \epsilon$$

where δ_i is the TD error and ϵ is a small constant to ensure all transitions have a non-zero probability of being sampled. The replay buffer is updated with new transitions and the actor and critic networks are trained using the sampled minibatches. Then an Importance sampling weight of

$$w_i = (\frac{1}{N} \cdot \frac{1}{P(i)})^\beta$$

is used to samples a minibatch of transitions for training. This mechanism ensures that the agent learns more from informative transitions, accelerating the learning process.

The replay buffer is updated with new transitions using the loss function and the actor and critic networks are trained using the sampled minibatches. The critic network is trained to minimize the Bellman error

$$L(\theta^Q) = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$

where $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$ and the actor network is trained to maximize the critic's Q-value

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s=s_i}$$

.

Training

The DDPG algorithm was trained on the PandaReach task in a 3D environment(1). The training process involved iteratively updating the actor and critic networks using the replay buffer and sampled minibatches. The networks were trained for a fixed number of episodes, with the target networks updated at each time step. The training process was optimized using the Whale Optimization Algorithm (WOA) (Ashraf et al. 2021) for hyperparameter tuning.The algorithm was trained using a dense reward structure based on goal distance metrics, enabling faster convergence and more stable learning. The trained networks were then evaluated on the PandaReach task to assess their performance in path planning scenarios.

During this process, we also did reward shaping to implement a dense reward function based on the normalized distance to completion. This approach provides continuous feedback during training, helping the agent learn more efficiently than sparse rewards would allow. The reward is calculated as the negative Euclidean distance between the end-effector and target position, with an additional bonus for successful completion:

$$r_t = -\|p_t - p\|_2 + \text{bonus}_{\text{success}}$$

where p_t is the target position, p is the end-effector position, and $\text{bonus}_{\text{success}}$ is a reward bonus for reaching the target position. This dense reward structure encourages the agent to minimize the distance to the target, accelerating learning and improving convergence speed.

Once a satisfactory performance was achieved, the trained networks were evaluated on the PandaReach task to assess their path planning capabilities. The results were compared

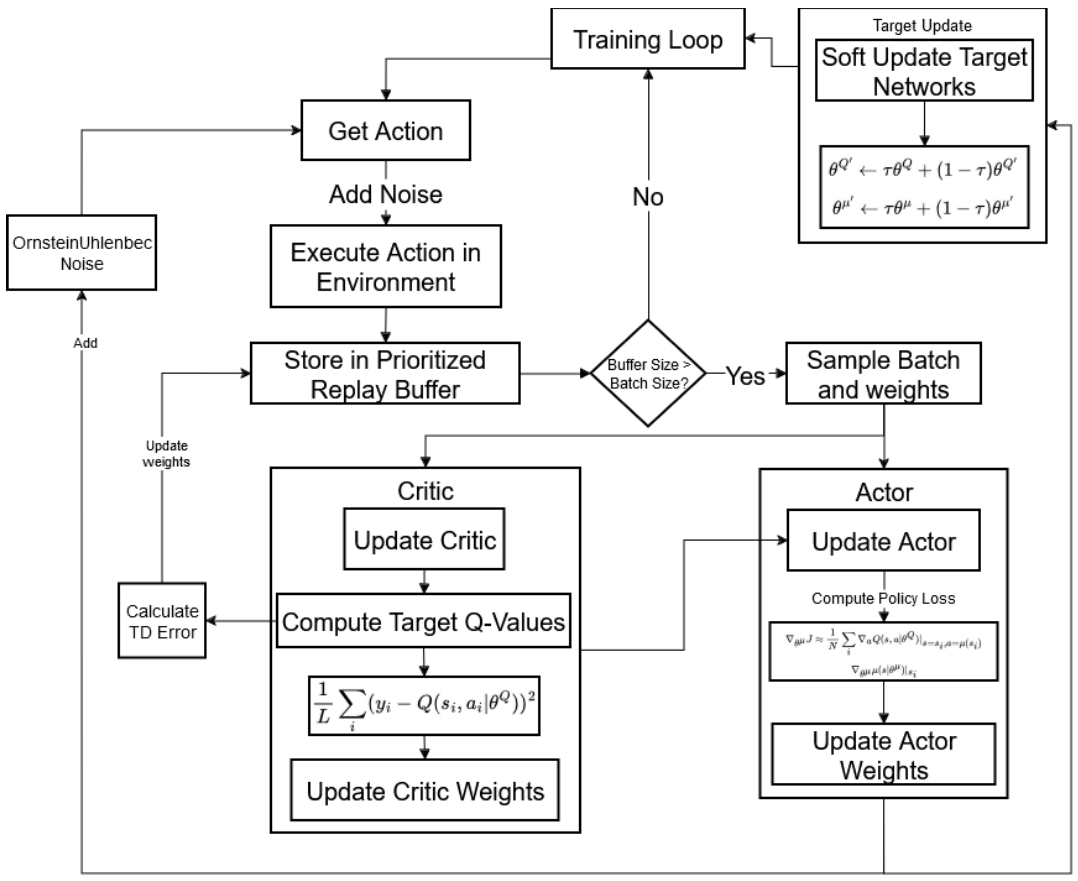
Training Loop
Target Update
Soft Update Target Networks
$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$
$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$
Get Action
Add Noise
No
OrnsteinUhlenbec Noise
Execute Action in Environment
Add
Store in Prioritized Replay Buffer
Buffer Size > Batch Size?
Yes
Sample Batch and weights
Update weights
Critic
Update Critic
Calculate TD Error
Compute Target Q-Values
$\frac{1}{L}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
Update Critic Weights
Actor
Update Actor
Compute Policy Loss
$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)}$
$\nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
Update Actor Weights

Figure 2: Flowchart of our DDPG implementation

against standard PPO implementations to highlight the improvements achieved through our optimized methodology. The evaluation process involved testing the agent's performance in various path planning scenarios, including complex tasks like opening a drawer.

Neural Radiance Fields Integration

Using the work of (Shim, Lee, and Kim 2023), I tried to reproduce their results. SNeRL (Semantic-aware Neural Radiance Fields) integrates NeRF with reinforcement learning by providing rich semantic representations of the environment. SNeRL operates in two distinct stages:

1. Pre-training stage with semantic-aware NeRF
2. Downstream RL task training with the pre-trained encoder

Multi View Encoder The multi-view encoder Ω combines observations from multiple camera perspectives into a unified latent representation z for RL applications. The encoder takes as input RGB images $o^i \in R^{H\times W\times 3}$ and their associated camera projection matrices $K^i \in R^{3\times 4}$ from V different viewpoints, indexed by $i = 1\cdots V$. To construct $z \in Z$, the system first employs a CNN encoder E_{CNN} to extract view-independent features from each image. These extracted features are then combined through channel-wise concatenation with their respective (flattened) camera projection matrices to encode viewpoint information. The resulting concatenated vectors are transformed through MLP layers, denoted as g_{MLP}, to create intermediate view-aware representations. The system then averages these representations across all views to obtain a unified encoding, which is finally mapped to the latent space Z using another MLP network h_{MLP}, expressed mathematically as:

$$z = \Omega(o^{1:V}, K^{1:V}) = h_{MLP}(\frac{1}{V}\sum_{i=1}^{V} g_{MLP}(E_{CNN}(o^i), K^i))$$

Neural Rendering Function From the environment, we first generate 3 images:

- RGB images (c)
- Semantic labels (s)
- Depth images (f)

The Loss Functions uses a combination of three loss components:

$$L = L_{RGB} + \lambda_{sem}L_{sem} + \lambda_{feat}L_{feat} \tag{1}$$

where L_{RGB} is the RGB loss, L_{sem} is the semantic loss, and L_{feat} is the feature loss. The loss computation consists of two components: a semantic component and a feature component. The semantic component employs cross-entropy to measure the discrepancy between predicted and ground truth semantic labels. For the feature component, the L2 norm quantifies the distance between predicted and ground truth feature vectors. The neural rendering function's parameters are then iteratively refined through gradient descent optimization to minimize this combined loss function.

The architecture effectively combines geometric understanding with semantic awareness, making it particularly suitable for reinforcement learning tasks that require both spatial and semantic understanding of the environment.

Then a trained decoder is used to generate the semantic feature for current environment and then it is added to the existng state space of the DDPG algorithm. This integration allows the agent to learn policies that are aware of both geometric and semantic properties of the environment, enabling more sophisticated interaction strategies. The agent is then trained with the new state space and the results are compared with the previous implementation.

Experiments

We have chosen 2 environments for this manuscript:

- PandaReach: The goal is for the position End effector-Franka Emika Panda robot to reach a target position in 3D space.
- metaworld/drawer-open-v2: The goal is to open a drawer in the metaworld environment.

PandaReach

PandaReach was chosen as the primary environment for evaluating the DDPG algorithm's performance in path planning scenarios. Its simple nature allows faster evaluation during the initial stages of training.
Using parameters from 1 and using sparse rewards, the agent was unable to learn the optimal policy even after 100,000 time steps/200 episodes as seen in 3.

Table 1: Initital Hyperparameters

Parameter	Value
Actor Learning Rate	0.001
Critic Learning Rate	0.001
Discount Factor (γ)	0.99
Soft Update Rate (τ)	0.005
Exploration Noise (ϵ)	0.1
Batch Size	100
Buffer Size	1,000,000

Dense Rewards Shifting to dense rewards significantly enhance DDPG's performance by providing immediate feedback after each action, making the credit assignment problem more manageable and reducing blind exploration. This continuous feedback helps prevent the common deadlock situations where DDPG can get stuck with sparse rewards, as both the critic and actor networks receive regular learning signals to evolve. The immediate nature of dense

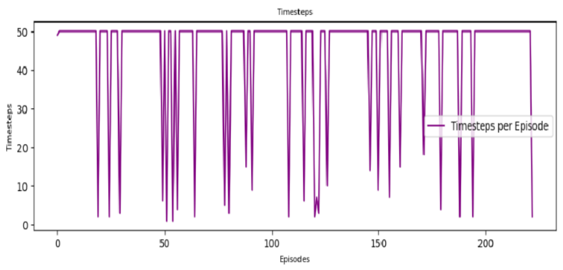

Figure 3: Episode length during training of DDPG using sparse rewards

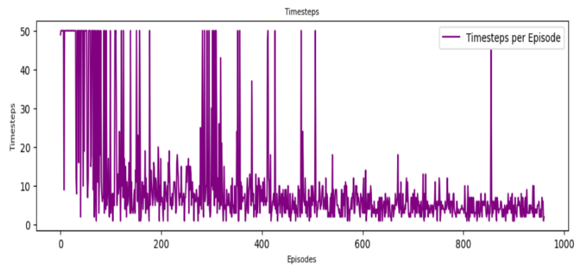

Figure 4: Timesteps per episode while training DDPG using dense rewards

rewards enables the agent to understand which actions contribute positively to the task without waiting for sparse terminal rewards, leading to faster convergence rates. The optimal policy was learned in less than 600 episodes as seen in 4.

Whale Optimization Algorithm Using WOA before training the agent provides a significant boost in performance in environments with continuous action spaces. WOA effectively optimizes five critical DDPG parameters: actor learning rate, critic learning rate, discount factor, target network learning rate, and batch size. Its simple operation and minimal control parameters, combined with a robust ability to escape local optima, make it particularly effective for this optimization task. The algorithm was able to find the optimal hyperparameters for the environment in less than 100 iterations. The hyperparameters found by WOA are shown in 2.

Table 2: Whale Optimized Hyperparameters

Parameter	Value
Actor Learning Rate	7.08×10^{-4}
Critic Learning Rate	5.32×10^{-3}
Discount Factor (γ)	0.958
Soft Update Rate (τ)	7.57×10^{-3}
Exploration Noise (ϵ)	0.477
Batch Size	174
Buffer Size	60,167

Prioritized Experience Replay The integration of Prioritized Experience Replay (PER) further enhances the DDPG

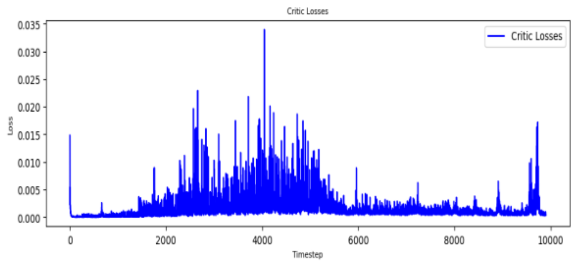

Figure 5: Critic Loss while training DDPG using dense rewards

algorithm's performance by prioritizing informative transitions during training. By focusing on high TD error transitions, PER accelerates learning by replaying critical experiences more frequently. This mechanism helps the agent learn from its mistakes and explore more efficiently, leading to faster convergence and improved performance. The agent was able to learn the optimal policy in less than 300 episodes as seen in 6. Comparing 5 and 6, the most significant improvement was seen in the actor and critic losses, which converged very fast and remained stable throughout training.

PPO Comparison

Using the same environment, the agent was trained using PPO using the RL Baseline3 Zoo implementation (Raffin 2020). This implementation uses Optuna for hyperparameter training and uses the Stable-Baselines3 implementation of PPO. The agent was trained for 100,000 time steps with the same environment.

Comparing the results of DDPG and PPO, DDPG outperformed PPO in terms of sample efficiency and convergence speed. DDPG was able to learn the optimal policy in less than 300 episodes, while PPO required more than 1000 episodes to achieve similar performance (7 and 8). Specifically, DDPG achieved an average reward near to 0 in approximately 250 episodes, whereas PPO took around 2000 episodes to reach the same reward level. Additionally, the average episode length for DDPG stabilized around 10 steps after 800 episodes, while PPO required over 3000 episodes to achieve similar stability. However, with optimizing parameters for PPO similar results can be achieved for this environment. The results highlight the importance of the changes we have done to overcome the inherent disadvantages of DDPG like getting stuck in local optima and hyperparameter training while continuing to exploit its advantages due to its off-policy nature.

Meta-World Drawer-Open-v2

While setting up Nerf, we faced a lot of issues. The code provided by (Shim, Lee, and Kim 2023) did not contain the code to generate dataset for the encoder training. After writing the code to generate the dataset9, we faced issues with the training of the encoder. The training needed almost 30GB of memory and took 30 hrs for training and the results were not as expected. We were not able to integrate the trained encoder with the DDPG algorithm.

However, we were able to run our DDPG code without SNeRL on the Meta-World Drawer-Open-v2 task. The agent was highly sensitive, sometimes learning the policy within 500 episodes but most of the time getting stuck in a local minima 10. Here we can see DDPG's hyper sensitivity and how it's performance is highly environment-dependent. This output agrees with the results of the original paper 11 where the agent was able to learn the optimal policy in 800,000 timesteps while using SNeRL but gets stuck in a local minima without it. This shows the importance of semantic information in the environment for the agent to learn the optimal policy.

Conclusion

This research demonstrates significant improvements in DDPG's performance for robotic path planning through several key optimizations. Comparing the results of the three experiments in Panda environment, it is evident that the combination of dense rewards, WOA, and PER significantly enhances the DDPG algorithm's performance in path planning scenarios. The agent was able to learn the optimal policy in less than 300 episodes, demonstrating the effectiveness of these optimization techniques in accelerating learning and improving convergence rates. The results highlight the importance of reward shaping, hyperparameter optimization, and experience replay in enhancing the DDPG algorithm's performance in complex robotic tasks.

The dense reward structure proved particularly effective in providing continuous feedback, enabling the agent to learn more efficiently than with traditional sparse rewards. The comparative analysis between our optimized DDPG implementation and PPO revealed superior sample efficiency and faster convergence rates for our approach.

However, experiments with the Meta-World Drawer-Open-v2 task exposed DDPG's sensitivity to environmental complexity, highlighting the algorithm's tendency to get trapped in local optima when dealing with more intricate manipulation tasks. DDPG can fall into a deadlock situation. Once the actor policy drifts too far from optimal regions, the critic's gradient becomes zero, preventing further updates to either network. This creates a persistent suboptimal state that continues even when given ideal training samples.

We observed that when DDPG finds rewarding states early in training, it has a much higher chance of converging to the optimal policy. If the reward is discovered later, the success rate drops significantly and the agent is more likely to get stuck in suboptimal behaviors.

While our attempts to integrate Neural Radiance Fields (NeRF) faced computational challenges, the preliminary results from the Meta-World environment align with previous research suggesting the importance of rich semantic information in complex manipulation tasks. Future work should focus on developing more computationally efficient methods

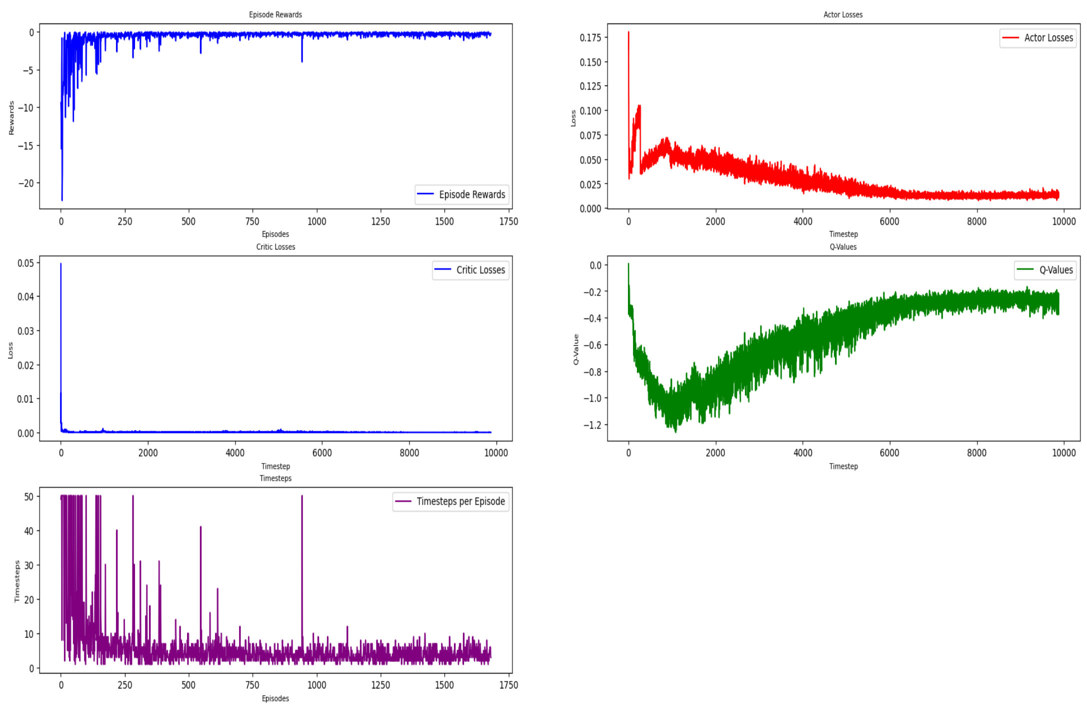

Figure 6: DDPG using Dense Rewards, WOA and Prioritized Experience Replay

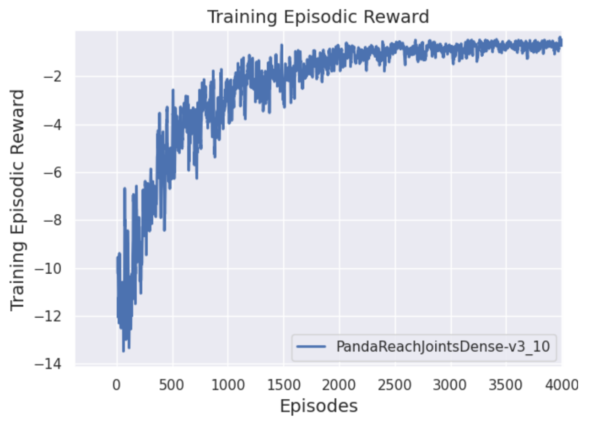

Figure 7: Average Reward per episode during training of PPO

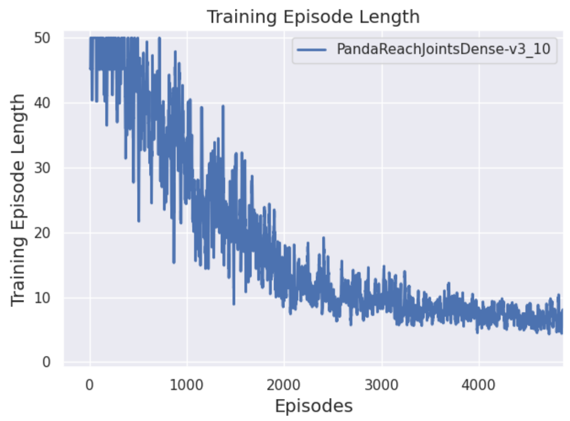

Figure 8: Length per episode during training of PPO

for incorporating semantic information into the learning process, potentially through lightweight alternatives to NeRF that maintain the benefits of semantic awareness while reducing computational overhead.

References

Agrawal, S.; Garimella, R.; and Desmier, G. 1996. Free-floating closed-chain planar robots: Kinematics and path planning. *Nonlinear Dynamics*, 9: 1–19.

Al Ali, A.; Shi, J.-F.; and Zhu, Z. H. 2024. Path planning of 6-DOF free-floating space robotic manipulators using reinforcement learning. *Acta Astronautica*, 224: 367–378.

Ashraf, N. M.; Mostafa, R. R.; Sakr, R. H.; and Rashad, M. Z. 2021. Optimizing hyperparameters of deep reinforcement learning for autonomous driving based on whale optimization algorithm. *PLOS ONE*, 16(6): 1–24.

Driess, D.; Schubert, I.; Florence, P.; Li, Y.; and Toussaint, M. 2022. Reinforcement Learning with Neural Radiance Fields. arXiv:2206.01634.

Gallouédec, Q.; Cazin, N.; Dellandréa, E.; and Chen, L. 2021. panda-gym: Open-Source Goal-Conditioned Environments for Robotic Learning. *4th Robot Learning Workshop: Self-Supervised and Lifelong Learning at NeurIPS*.

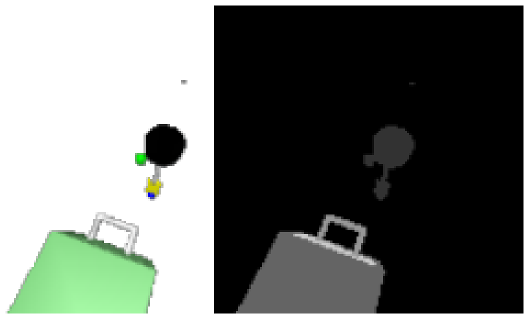

Figure 9: Generated Nerf Dataset containing Semantic and RGB view

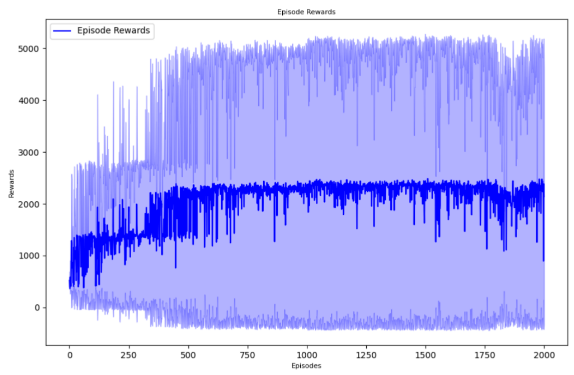

Figure 10: DDPG performance on Meta-World Drawer-Open-v2 task

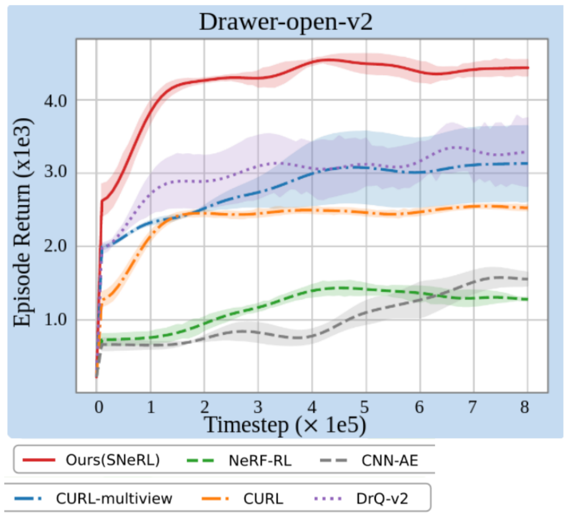

Figure 11: DDPG performance on Meta-World Drawer-Open-v2 task

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *ArXiv*, abs/1801.01290.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2019. Continuous control with deep reinforcement learning. arXiv:1509.02971.

Mirjalili, S.; and Lewis, A. 2016. The Whale Optimization Algorithm. *Advances in Engineering Software*.

Otto, F.; Celik, O.; Roth, D.; and Zhou, H. ???? Fancy Gym.

Raffin, A. 2020. RL Baselines3 Zoo. https://github.com/DLR-RM/rl-baselines3-zoo.

Shim, D.; Lee, S.; and Kim, H. J. 2023. SNeRL: Semantic-aware Neural Radiance Fields for Reinforcement Learning. arXiv:2301.11520.

Yao, J.; Luo, X.; Li, F.; et al. 2024. Research on hybrid strategy Particle Swarm Optimization algorithm and its applications. *Scientific Reports*, 14: 24928.