Published in: IEEE Transactions on Cognitive and Developmental Systems

Federated Reinforcement Learning for Collective Navigation of Robotic Swarms

Authors:
Seongin Na1 , Toma´s Roucek, Jir´
Ulrich , Jan Pikman, Toma´s Krajnk ,
Barry Lennox and Farshad Arvin

Presented By:
Ittehad Saleh Chowdhury
Reyadath Ullah Akib

Introduction

Nature-Inspired Swarm Robotics: Focuses on coordinating numerous robots to perform complex tasks, inspired by social insects.

Applications and Design Challenges: Highlights uses in challenging environments and the difficulties in creating individual robot behaviors.

Automatic Design Methods: Underlines the shift from manual to automatic methods, like Reinforcement Learning, for behavior optimization.

FLDDPG Strategy Introduction: Presents a Federated Learning-based strategy to enhance communication efficiency and data confidentiality in swarm robotics.

Challenges

Robot Behavior Design: Challenges in designing behaviors for individual robots to ensure cohesive swarm actions without centralized control.

Manual Design Limitations: Manual methods struggle with complexity and lack adaptability, unsuitable for dynamic environments.

Communication Constraints: The necessity for strategies that reduce reliance on central servers and cope with limited communication bandwidth in swarm robotics.

Applications

Learning for Navigation: Focus on navigation and collision avoidance, highlighting collective learning's role in improving swarm robotics.

Addressing Limited Communication: Showcases FLDDPG's capability in environments with restricted communication bandwidth.

Broad Application Prospects: Indicates FLDDPG's potential in autonomous operations in communication-constrained settings, like underground or marine exploration.

DRL Training Algorithm

- A Deep Deterministic Policy Gradient (DDPG) algorithm has been selected
- DDPG is an **actor-critic**, model-free DRL algorithm for agents with continuous observation and action spaces

Experience Replay Feature: DDPG uses experience replay to collect transition samples into memory, enabling efficient neural network training from shared memories across multiple agents.

Multi-Agent Applicability: The shared access to experience replay memory makes DDPG suitable for both multi-agent and single-agent scenarios.

Traditional DRL Training Strategies

Independent DDPG: There is no communications between the robots and the server for training.

Shared Experience DDPG: The agents have individual neural networks and a shared memory

Shared Network DDPG: Uses a shared neural network and a shared memory

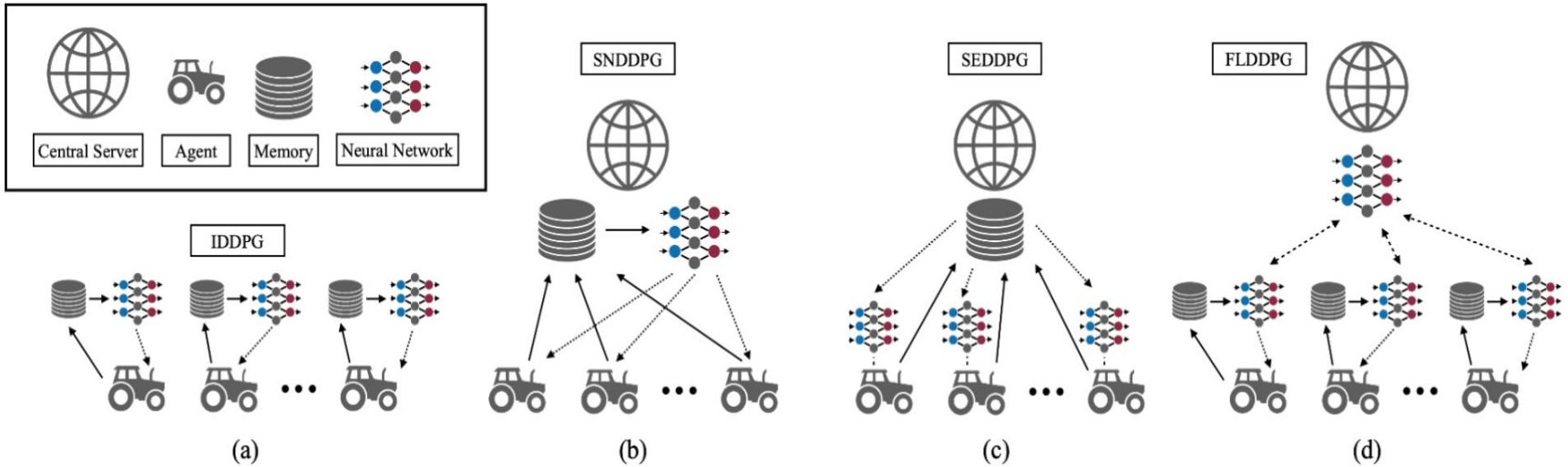


Fig. 1. Deep reinforcement learning training architectures for swarm robotic systems. (a) Individual DDPG (IDDPG), (b) Shared Network DDPG (SNDDPG), (c) Shared Experience DDPG (SEDDPG) and (d) proposed Federated Learning DDPG (FLDDPG).

Federated DRL Training Strategy

FLDDPG Strategy: Combines FL and DDPG to share only neural network weights, not local data, between robots and a central server, reducing communication needs.

Initial Setup: Initializes actor and critic networks and local memories for each robot, with data collection and parameter updates performed individually.

Transition Sample Collection: Collects transition samples for a set number of time steps, storing them in a replay buffer for episodic learning.

Continued.

Network Updates: Updates actor and critic networks by selecting samples from local memories, calculating targets, and minimizing loss for critic updates and using policy gradient for actor updates.

Soft Weight Update: Introduces a new method for efficiently updating local neural network weights with averaged weights, addressing the limitations of hard weight averaging by allowing fractional updates to maintain efficiency and adaptability of individual controllers.

Algorithm 1: DDPG with FL (FLDDPG)

Randomly initialise N critic neural networks, $Q_{1,\ldots,N}(s, a|\theta^{Q}_{1,\ldots,N})$ and N actor neural networks $\pi_{1,\ldots,N}(s|\theta^{\pi}_{1,\ldots,N})$ with weights $\theta^{Q}_{1,\ldots,N}$ and $\theta^{\pi}_{1,\ldots,N}$
Initialise N target networks $Q'_{1,\ldots,N}$ and $\pi'_{1,\ldots,N}$ with weights $\theta^{Q'}_{1,\ldots,N} \leftarrow \theta^{Q}_{1,\ldots,N}$, $\theta^{\pi'}_{1,\ldots,N} \leftarrow \theta^{\pi}_{1,\ldots,N}$
Initialise replay buffers $R_{1,\ldots,N}$
for *episode = 1, ..., M* **do**
    Initialise the states $s_t = s_1$
    for *t = 1, ..., T* **do**
        Run N actors and collect transition samples $D_t = (s_t, a_t, r_t, s_{t+1})$ into $R_{1,\ldots,N}$
        if *t = 0 mod* t_{train} **then**
            Sample l transitions (s_i, a_i, r_i, s_{i+1}) from local replay buffer memories, $R_{1,\ldots,N}$
            Set $y_i = r_i + \gamma Q'(s_i, \pi'(s_{i+1}|\theta^{\pi'})|\theta^{Q'})$
            Update critic networks by minimizing the loss: $L_Q = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^{Q}))^2$
            Update actor networks using the sampled policy gradient:
            $\nabla_{\theta^{\pi}} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^{Q})|_{s=s_i, a=\pi(s_i)}$
            $\cdot \nabla_{\theta^{\pi}} \pi(s_i|\theta^{\pi})|_{s=s_i}$
        end
        if *t = 0 mod* t_{target} **then**
            Update the target networks:
            $\theta^{Q'} \leftarrow \theta^{Q}$, $\theta^{\pi'} \leftarrow \theta^{\pi}$
        end
    end
    if *episode = 0 mod* T_{wa} **then**
        Perform *Soft Weight Update* for actor and critic networks for all robots
        $\theta_{wa} = \frac{1}{N}\sum_{k=1}^{N} \theta_k$
        for *i = 1, ... , N* **do**
            $\theta_i = \tau\theta_i + (1-\tau)\theta_{wa}$
        end
    end
end

Hard Weight vs Soft Weight update

Hard Weight Update: With the hard weight update method, the averaged weights are directly assigned to the local neural network weights.

$$\theta_{wa} = \frac{1}{N}\sum_{k=1}^{N}\theta_k \qquad (1)$$

$$\theta_{1,\ldots,N} = \theta_{wa},$$

Soft Weight Update: The local neural network weights are fractionally updated with the averaged weights.

$\theta_{wa} = \frac{1}{N}\sum_{k=1}^{N}\theta_k$
for $i = 1, \ldots, N$ **do**
  $\theta_i = \tau\theta_i + (1-\tau)\theta_{wa}$
end

Experiment

Four different training strategies were designed to evaluate the performance of:

- Individual DDPG
- Shared Experience DDPG
- Shared Network DDPG
- Federated Learning DDPG

DRL Implementation (Design Policy)

The four important design specifications are:

1.Observation Space: During the training, the robot collected observation to learn navigation and collision avoidance. the distance between the target and the current position of the robot in polar coordinates (d, θ_d) were collected.

2.Action Space: There are two action values in our setting:

1. Translational velocity (v)
2. Rotational velocity (ω)

   i.e a = [v, ω]

DRL Implementation (Design Policy)

3.Reward Function: The reward functions were designed to enable the learning of:

1. Navigation
2. Collision

$$r = r_g + r_p + r_c + r_a$$

$$r_g = \begin{cases} R_g, & \text{if arrived goal} \\ 0, & \text{otherwise} \end{cases},$$

$$r_p = \begin{cases} ad, & d > 0 \\ -ad, & \text{otherwise} \end{cases},$$

$$r_c = \begin{cases} R_c, & \text{if collision} \\ 0, & \text{otherwise} \end{cases},$$

$$r_a = \begin{cases} -e^{max(s_{laser})*\lambda}, & \text{any}(s_{laser}) > 0 \\ 0, & \text{otherwise} \end{cases}$$

r_g= goal arrival reward

r_p= goal approaching reward

r_c= collision penalty

r_a= approaching penalty

4.Neural Network Architecture: In DDPG, each robot requires two sets of neural networks:

1. Actor Network
2. Critic Network

Results

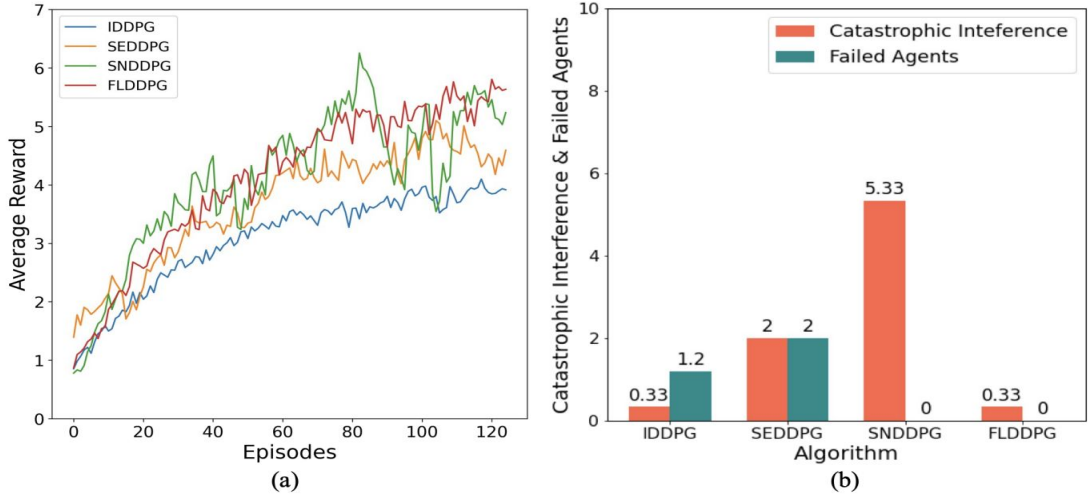


Fig. 6. The training results of four different algorithms: IDDPG, SNDDPG, SEDDPG and FLDDPG. (a) Average reward per episode over training. (b) Average number of catastrophic interference and failed agents per training.

Results

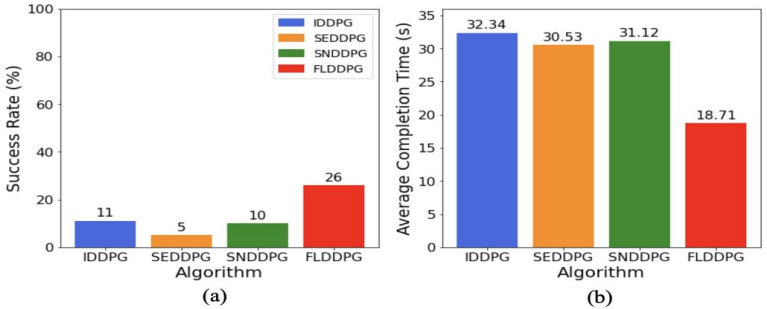
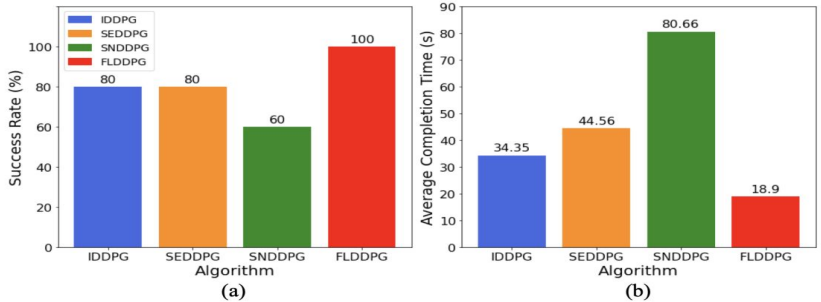


Fig. 7. The evaluation experiment results with the four different DRL training strategies. (a) and (b) show success rate and average completion time of 4 agents over 20 runs of the four algorithms respectively.



Thank You!