Safe Learning for MADDPG with Control Barrier Certification for Long duration ground surveillance

Lokesh Bansal
Robert Bosch Center for Cyber
Physical Systems
Indian Insttute of Science
Bangalore, India
lokeshbansal@iisc.ac.in

***Abstract*—Reinforcement learning based control algorithms require large amount of training. During training the agent explore many possible safe and unsafe states. In the simulation environment exploration of unsafe states can be affordable. But many times, training in simulation world is not a feasible option because of issues of modeling of real-world scenarios. In such scenarios direct real-world training seems to be feasible solution. In such real-world training exploration of unsafe state is very dangerous for the agent as well as for the environment. In this work, safe learning is carried out for multi agent deep deterministic policy gradient (MADDPG) algorithm with the help of control barrier functions (CBF). This MADDPG augmented with CBF is applied for an application of long duration autonomy. Battery driven multiple ground robots are deployed for surveillance of a given region with fix charging stations. The robots are trained for surveillance considering safe learning i.e. robots will not drain out of battery during training and testing for surveillance.**

Keywords—Reinforcement Learning, multiagent deep deterministic policy gradient, control barrier functions, safe learning, long duration autonomy

I. INTRODUCTION

Safe learning is a requirement for reinforcement learning algorithms to deploy them for real-world applications. Robotics applications are safety critical where agents as well as environment both are safety critical stakeholders, and we cannot afford any unsafe state during training. Control barrier functions (CBF) can be used to provide safety guarantees to remain in safe set. In the given paper I have used this property of CBF to make safe learning for a reinforcement learning algorithm: multi agent deep deterministic policy gradient algorithm.

The rest of this paper is organized as follows. Control Barrier Function and Deep deterministic policy gradient (DDPG) algorithm is discussed in section II- Background. Section III explains problem formulation, required notations and equations along with already proposed approach in literature. Section IV explains solution of the problem with proposed Approach using MADDPG with Barrier Certification modification. Section V presents the results. Conclusion and future work is presented in Section VI.

II. BACKGROUND

A. Control Barrier Function

Dynamics: Control-affine system

$$\dot{x}_i = f(x_i) + g(x_i)u_i$$

where, $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$

Safe set: We consider a set C defined as the superlevel set of a smooth function h: $R^n \rightarrow R$, yielding:

$$\mathcal{C} = \{x \in R^n \mid h(x) \geq 0\},$$
$$\partial\mathcal{C} = \{x \in R^n \mid h(x) = 0\},$$
$$Int(\mathcal{C}) = \{x \in R^n \mid h(x) > 0\}.$$

We refer to C as the safe set.
Control invariant set: A set is control invariant if there exists a control law that keeps any trajectory starting in the set within the set.
Let $C \subset R$ be the superlevel set of a smooth function h: $R \rightarrow R$, then h is a control barrier function (CBF) if there exists an extended class K_∞ function α such that for the control system:

$$\exists u \in \mathbb{R}^m, \forall x \in \mathbb{R}^m, L_f h(x) + L_g h(x) u \geq -\alpha(h(x))$$

Safety-Critical Control via Quadratic Program:

$$u(x) := \underset{u \in \mathbb{R}^m}{\operatorname{argmin}} \; u^T u$$
$$\text{s.t. } L_f h(x) + L_g h(x) u \geq -\alpha(h(x))$$

B. DDPG Algorithm

Deep deterministic policy gradient (DDPG) approach is closely connected to Q-learning. DDPG is an off-policy algorithm. DDPG can only be used for environments with continuous action spaces. DDPG can be thought of as being deep Q-learning for continuous action spaces. It uses off-policy data and the Bellman equation to learn the Q-function. It uses the Q-function to learn the policy. Computing the maximum over actions is a challenge in continuous action spaces. DDPG deals with this by using a target policy network to compute an action which approximately maximizes Q_{phi_targ}.

Network Schematics

DDPG uses four neural networks: a Q network, a deterministic policy network, a target Q network, and a target policy network.

Parameters:

θ^Q : Q network
θ^μ : Deterministic policy function
$\theta^{Q'}$: target Q network
$\theta^{\mu'}$: target policy network

The Q network and policy network is very much like simple Advantage Actor-Critic, but in DDPG, the Actor directly maps states to actions (the output of the network directly the output) instead of outputting the probability distribution across a discrete action space

The target networks are time-delayed copies of their original networks that slowly track the learned networks. Using these target value networks greatly improve stability in learning. Here's why: In methods that do not use target networks, the update equations of the network are interdependent on the values calculated by the network itself, which makes it prone to divergence.

For example:

This depends Q function itself (at the moment it is being optimized)

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a) + \gamma max\overbrace{Q(s',a')} - Q(s,a)]$$

So, here's the pseudo-code of the algorithm that we want to implement:

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

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

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

We are going to break this down into:

1. Experience replay
2. Actor & Critic network updates
3. Target network updates
4. Exploration

Replay Buffer

As used in Deep Q learning (and many other RL algorithms), DDPG also uses a replay buffer to sample experience to update neural network parameters. During each trajectory roll-out, we save all the experience tuples (state, action, reward, next state) and store them in a finite-sized cache — a "replay buffer." Then, we sample random mini batches of experience from the replay buffer when we update the value and policy networks. Why do we use experience replay? In optimization asks, we want the data to be independently distributed. This fails to be the case when we optimize a sequential decision process in an on-policy way, because the data then would not be independent of each other. When we store them in a replay buffer and take random batches for training, we overcome this issue.

Actor (Policy) & Critic (Value) Network Updates

The value network is updated similarly as is done in Q-learning. The updated Q value is obtained by the Bellman equation:

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$$

However, in DDPG, the next-state Q values are calculated with the target value network and target policy network. Then, we minimize the mean-squared loss between the updated Q value and the original Q value:

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

* Note that the original Q value is calculated with the value network, not the target value network.

For the policy function, our objective is to maximize the expected return:

$$J(\theta) = \mathbb{E}[Q(s,a)|_{s=s_t, a_t=\mu(s_t)}]$$

To calculate the policy loss, we take the derivative of the objective function with respect to the policy parameter. Keep in mind that the actor (policy) function is differentiable, so we have to apply the chain rule.

$$\nabla_{\theta^\mu} J(\theta) \approx \nabla_a Q(s,a) \nabla_{\theta^\mu}\mu(s|\theta^\mu)$$

But since we are updating the policy in an off-policy way with batches of experience, we take the mean of the sum of gradients calculated from the mini-batch:

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

Target Network Updates

We make a copy of the target network parameters and have them slowly track those of the learned networks via "soft updates," as illustrated below:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

$$\text{where} \quad \tau \ll 1$$

Exploration

In Reinforcement learning for discrete action spaces, exploration is done via probabilistically selecting a random action (such as epsilon-greedy or Boltzmann exploration). For continuous action spaces, exploration is done via adding noise to the action itself (there is also the parameter space noise, but we will skip that for now).

$$\mu'(s_t) = \mu(s_t|\theta_t^\mu) + \mathcal{N}$$

III. Problem Formulation

Safe Learning for MADDPG with Control Barrier Certification for Long duration ground surveillance.

Applications: Multi Agent Systems

Figure 1: Swarm of drones for surveillance

Figure 2: Autonomous electric vehicles

Figure 3: Battery driven warehouse robots

Proposed Environment: Ground robots deployed for surveillance for long duration autonomy

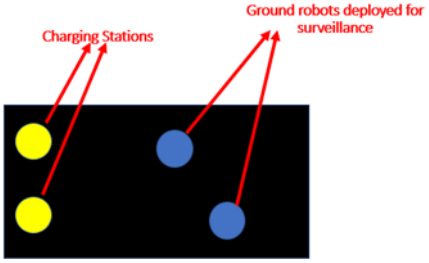

Figure 4: Long duration ground surveillance

Constraints:

1) Robot never gets stranded away from a charging station

2)Prevent overcharging

3)Collision avoidance

A. Notations and Equations:

- The dynamics of the augmented state X_i

$$\dot{\chi}_i = \begin{bmatrix} f(x_i) \\ \hat{f}(\chi_i) \end{bmatrix} + \begin{bmatrix} g(x_i) \\ \hat{g}(\chi_i) \end{bmatrix} u_i = F(\chi_i) + G(\chi_i)u_i.$$

Where:

- Dynamics of a robot is modelled by the following control affine system:

$$\dot{x}_i = f(x_i) + g(x_i)u_i$$

- And, the energy dynamics are given by (the energy E_i stored in robot i's battery)

$$\dot{E}_i = \hat{f}(\chi_i) + \hat{g}(\chi_i)u_i$$

- Static mapping from robot i's state to its position pi belongs to R^d, d = 2 for ground robots or d = 3 for aerial robots

$$p : x_i \in \mathbb{R}^n \mapsto p_i \in \mathbb{R}^d$$

- Function that evaluates the energy that robot i requires to reach a charging station starting from position p_i

$$\rho_i : p_i \in \mathbb{R}^d \mapsto \rho_i(p_i) \in \mathbb{R}_{\geq 0}$$

B. Given Approach in literature with Barrier Certification

Barrier Certification for Survivability constraints:

1) By ensuring that each robot never gets stranded away from a charging station:

$$h_{c,i}(\chi_i) = E_i - E_{min} - \rho_i(p(x_i)) \geq 0 \quad \forall i \in \{1, \ldots, N\}$$

2) To prevent overcharging:

$$h_{o,i}(\chi_i) = E_{max} - E_i \geq 0.$$

by defining the logical and of these constraints,

$$h_{e,i}(\chi_i) = \min\{h_{c,i}(\chi_i), h_{o,i}(\chi_i)\}$$

Barrier Certification for Environmental monitoring task:

- Reformulate the task itself using CBFs which can be then combined with the CBF for survivability in order to implement persistent environmental monitoring

- Consider N robots tasked with monitoring a compact and convex set omega is subset of Rd. We can define a measure of the coverage quality by defining a cost:

$$J(x) = \sum_{i=1}^{N} \int_{\Omega_i} \|p(x_i) - q\|^2 \phi(q) dq.$$

Barrier function related to the task as $h_t(x) = -J(x)$

Where,

- x is the ensemble state of the robots, $\{\Omega_1, \ldots, \Omega_N\}$ is the Voronoi tessellation of the set.

- The value $\phi(q) \in$ R; $\phi(q) \geq 0 \ \forall q \in \Omega$, , encodes the importance of the point q.

- Where the quality of the sensor coverage associated with the point q decreases quadratically with the distance $\|p(x_i) - q\|$.

Note: The further away the point to monitor is, the worse the coverage is, and the higher is the coverage cost J.

Barrier Certification for Collision avoidance:

$$h_s(\chi_i, \chi_j) = \|p(x_i) - p(x_j)\|^2 - \Delta^2 \geq 0$$

Combining the Barrier certificates for Collision avoidance and Survivability constraints:

$$h_i(\chi_i) = \min \left\{ \min_i \{h_{e,i}(\chi_i)\}, \min_{\substack{i,j \\ i \neq j}} \{h_s(\chi_i, \chi_j)\} \right\}$$

Each robot executes the input u_i solution of the following QP:

$$\begin{aligned} \min_{u_1, \ldots, u_N, \delta} \ & \sum_{i=1}^{N} \|u_i\|^2 + \kappa|\delta|^2 \\ \text{s.t. } & L_F h_i(\chi_i) + L_G h_i(\chi_i) u_i \geq -\alpha(h_i(\chi_i)), \ \forall i \\ & L_F h_t(\chi) + L_G h_t(\chi) u \geq -\alpha(h_t(\chi)) - \delta \end{aligned}$$

IV. Proposed Approach using MADDPG with Barrier Certification

Safety using Barrier Certification:

$$\begin{aligned} \min_{u_1, \ldots, u_N,} \ & \sum_{i=1}^{N} \|u_i\|^2 \\ \text{s.t. } & L_F h_i(\chi_i) + L_G h_i(\chi_i) u_i \geq -\alpha(h_i(\chi_i)), \ \forall i \end{aligned}$$

Surveillance using MADDPG:

$$J(x) = \sum_{i=1}^{N} \int_{\Omega_i} \|p(x_i) - q\|^2 \phi(q) dq.$$

A safe control input u_i safe can be determined by solving the following Quadratic program

$$u^i_{safe} = \underset{u}{\operatorname{argmin}} \frac{1}{2} \left\| u - \hat{u}^i \right\|$$

$$\text{subject to} \quad L_F h_i(\chi_i) + L_G h_i(\chi_i) u + \alpha(h_i(\chi_i)) \geq 0$$

MADDPG with Control Barrier Certification Algorithm

Initialize
Initialize replay buffer $\mathcal{D}$
For episode = 1 to m do:
Reset environment
Initialize state s
For time step $t = 1$ to max-episode-length do
Initialize a random process $\mathcal{N}_t$ for action exploration
For agent $i = 1$ to n do
Select action $a^i_t = \pi^i_{\theta_i}(s^i_t) + \mathcal{N}_t$
w.r.t the current policy and exploration
Obtain safety action $a^i_{t,safe}$ by solving the QP
based on the nominal action a^i_t
Update action $a^i_t \leftarrow a^i_{t,safe}$
End For

Execute actions $\{a^i_t\}_{i \in N}$
Observe reward r_t and new state s'
Store $(s, \{a^i_t\}_{i \in N}, r_t, s')$ in replay buffer $\mathcal{D}$
Update state $s \leftarrow s'$
For agent $i = 1$ to n do:
Sample a random minibatch of k samples (s_j, a_j, r_j, s'_j) from $\mathcal{D}$
Calculate the target state-action value:
$y_j = r_j + \gamma Q^i_{\pi'}(s'_j, a'^1, .., a'^n)|_{a'^i = \pi'^i(s^i_j)}$
Update critic by minimizing the loss:
$L(\theta_i) = \frac{1}{k} \sum_j (y_j - Q^i_\pi(s_j, a^1_j, .., a^n_j))^2$
Update actor using the gradient of loss:
$\nabla_{\theta_i} J \approx \frac{1}{k} \sum_j \nabla_{\theta_i} \pi^i(s^i_j) \nabla_{\theta_i} Q^i_\pi(s_j, a^1_j, .., a^n_j)_{a^i = \pi^i(s^i_j)}$
Update target network parameters:
$\theta'_i \leftarrow \xi \theta_i + (1 - \xi) \theta'_i$
End For **End For** **End For**

V. Results

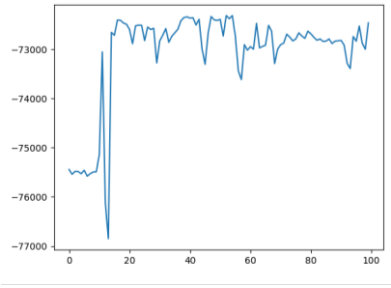

Figure 5. Training

Number of episodes: 10,000. Time steps per episode (episode length): 100. Save rate: after every 100 episodes

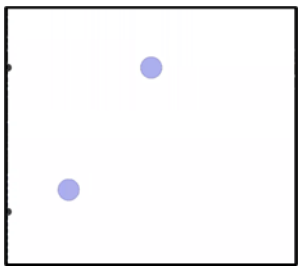

Figure 5. Agents doing surveillance

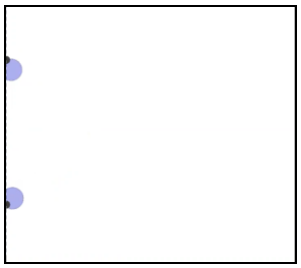

Figure 6. Agents getting charged

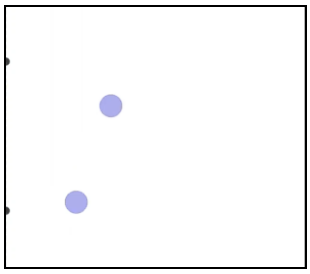

Figure 7. Agents doing surveillance

References

[1] Aaron D. Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada, “Control Barrier Functions: Theory and Applications”.

[2] Magnus Egerstedt, Jonathan N. Pauli, Gennaro Notomista, Seth Hutchinson, “Robot ecology: Constraint-based control design for long duration autonomy”.

[3] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver & Daan Wierstra, “Continuous control with deep reinforcement learning”.

[4] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, Yuval Tassa, “Safe Exploration in Continuous Action Spaces”.

[5] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, Igor Mordatch, “Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments”.