Multi-Agent Area Coverage Control Using Reinforcement Learning

Simon Hu

December 11, 2019

***Abstract*—In this paper, an area coverage control law is combined with a reinforcement learning algorithm to solve the optimal deployment problem. Traditional methods proposed for solving the method, from classical control theory, use proportional or linear quadratic Gaussian control designs. These traditional methods lack the robustness and adaptivity to dynamic environments, especially when these dynamic environment contribute significantly to the motion of the agents. A delayed actor-critic deep deterministic policy gradient algorithm is proposed to learn the value function and optimal policy. Simulation results are presented for stationary and time-varying scalar fields.**

I. Introduction

The problem of coverage control for multi-agent systems is concerned with deploying agents over an environment to maximize sensor coverage for various tasks like data collection and surveillance missions. Applications of this problem are harbour control, search and rescue, data, and surveillance missions. Consider a group of N homogeneous agents moving in a compact environment $\Omega \subset \mathbb{R}^2$ where the dynamics of the i-th agent are given by

$$\ddot{p}_i = u_i \tag{1}$$

where $p_i = (x_i, y_i)$ represents the agent's location and $u_i = (u_{x_i}, u_{y_i})$ represents the control input, which in this case is the direct acceleration input. The goal of solving the coverage control problem is to find the optimal configuration of the agent positions, $p = (p_1, p_2, \dots, p_N)$, with respect to the cost index

$$\mathcal{H}(p,t) = \int_\Omega \max_{i=1,2,\dots,N} f_i(\|q-p_i\|)\phi(q,t)\ \mathrm{d}q \tag{2}$$

where the integral is understood in the sense of Lesbegue. In the above, $\phi : \Omega \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a probibility density function that represents the probability of an event occuring at point $q \in \Omega$ and $f : \mathbb{R}^2 \to \mathbb{R}$ is a non-increasing function. Intuitively, $\phi(q,t)$ represents a belief of an event occuring at or near point q at time t. The wider the spread around the point q, the more important it becomes for the agents to provide sensor coverage in that area. This density is either assumed to be known, or inferred using some other external algorithm. The function f encodes the task to be performed. For the maximum area coverage problem, the function $f(x) = -x^2$ is used, and is assumed to be the same for all agents. It is easily checked that f is measurable on Ω and is a non-increasing function.

A. Voronoi Partitions

A collection $S = (S_1, S_2, \dots, S_M)$ is a partition of Ω if each pair (S_i, S_j) has disjoint interior and the union of all the S_i is Ω. That is, the S_i cover Ω. One particular partition of Ω is the Voronoi partition. The Voronoi partition of Ω is given by $V_\Omega = (\mathcal{V}_1, \mathcal{V}_2, \dots, \mathcal{V}_M)$ where each V_i is given by

(a) (b)

Fig. 1: (a) An example of a Voronoi partition, assuming that ϕ is uniform and stationary. (b) An example of a centroidal Voronoi partition, assuming that ϕ is uniform and stationary.

$$\mathcal{V}_i = \left\{ q \in \Omega : \|q-p_i\| \leq \|q-p_j\|, \forall j \neq i \right\}. \tag{3}$$

We say that agents i and j are Voronoi neighbors if $\partial V_i \cap \partial \mathcal{V}_j \neq \emptyset$. That is, two agents are Voronoi neighbors if their Voronoi cells share a boundary. The mass and center of mass of a Voronoi cell are given by

$$m_{\mathcal{V}_i} = \int_{\mathcal{V}_i} \phi(q,t)\ \mathrm{d}q \tag{4}$$

$$c_{\mathcal{V}_i} = (m_{\mathcal{V}_i})^{-1} \int_{\mathcal{V}_i} q\phi(q,t)\ \mathrm{d}q \tag{5}$$

respectively. The importance of the Voronoi partition framework is summarized in the following proposition.

Proposition 1. *Let V_Ω be the Voronoi partition of Ω. Then the cost index (2) can be rewritten as*

$$\mathcal{H}(p,t) = \sum_{i=1}^N \int_{\mathcal{V}_i} f(\|q-p_i\|)\phi(q,t)\ \mathrm{d}q. \tag{6}$$

Proof. Since $\mathcal{V}_\Omega$ is a covering of Ω, we write (2) as

$$\mathcal{H}(p,t) = \sum_{i=1}^N \int_{\mathcal{V}_i} \max_{i=1,2,\dots,N} f_i(\|q-p_i\|)\phi(q,t)\ \mathrm{d}q.$$

The result then follows from the definition of $\mathcal{V}_i$ and the non-increasing property of f. □

The implications of proposition 1 are important to note. In many applications, agents must communicate with each other for various reasons. For example, consider a scenario where agents must communicate with each other for the purpose of avoiding collisions. Communications can be energy-intensive, unreliable, or impossible when the distance between agents is very large. However, proposition 1 implies that agent i only needs to communicate with its Voronoi neighbors which reduces computational cost for communications at least, and makes the problem tractable at best. In this manner, any algorithm that utilizes the principle behind proposition 1 is said to be spatially distributed with respect to the underlying distribution.

B. Optimal Policy

It is shown in [1] that the optimal partition of Ω is the centroidal Voronoi partition and the optimal sensor placement p are located at the centers of the Voronoi cells. Thus, the optimal policy is to drive the agents towards their Voronoi centers. In [1], the authors consider a simple proportional-derivative controller of the form

$$u_i = -k_p(c_{\mathcal{V}_i} - p_i) - k_d(\dot{c}_{\mathcal{V}_i} - \dot{p}_i)$$

where $k_p, k_d > 0$. This controller performs reasonably well. However, classical control methods requires either tuning sensitive hyperparameters, using integral control to account for different inputs, or a combination of both. Reinforcement learning methods are introduced to train agents that learn dynamics of themselves and the environment. This is especially inmportant in scenarios where agents are deployed over hydrodynamic or aerodynamic environments.

II. Technical Approach

Designing a controller that realizes the control law in [1] is difficult, even for simple dynamics. When the dynamics become more intricate and complex, like an underwater glider dynamics for example, classical controller design becomes exponentially hard. To reduce this burden, reinforcement learning methodologies are employed. We first describe an actor-critic neural network approximation, and then an improvement to it, which is used in the final implementation.

A. Actor-Critic DDPG

Define $e_i(t) = c_{\mathcal{V}_i}(t) - p_i(t)$ as the centroid error for agent i and the Value function at the k-th step as

$$V(e_i(k)) = \sum_{\kappa=k}^{\infty} e_i^T(\kappa) Q e_i(\kappa) + u_i^T(\kappa) R u_i(\kappa) \tag{7}$$

where $Q, R \in \mathbb{R}^{2\times 2}$ are positive definite matrices. Note that if the process noise is assumed to be Gaussian then this is just the linear quadratic regulartor design from classical control theory. The second term in (7) encodes a measure of energy expenditure for physical systems so the design of the R matrix is important for long-term surveillance and data collection tasks. For notational simplicity, we write $V_{i,k} = V_i(e_i(k))$. The value function is rewritten in the form of a Bellman equation,

$$V_{i,k} = e_i^T(k) Q e_i(k) + u_i^T(k) R u_i(k) + V_{i,k+1} \tag{8}$$

so that the minimization problem to be solved is

$$V_{i,k}^* = \min \left[e_i^T(k) Q e_i(k) + u_i^T(k) R u_i(k) + \gamma V_{i,k+1}^* \right]$$

where γ is the discount factor. The end goal is to use this value function to obtain a policy π that ends up minimizing the value function. The authors in [2] propose to use a neural network to approximate both the value function and the policy. They are assumed to take the structure

$$\hat{V}_j(e_i(k)) = \omega_{c,j}^T \rho(e_i(k)), \tag{9}$$
$$\hat{\pi}_j(e_i(k)) = \omega_{a,j}^T \sigma(e_i(k)). \tag{10}$$

In the above, $\hat{V}_j$ and $\hat{\pi}_j$ are respectively the estimates of the value function and policy, $\omega_{c,j}, \omega_{a,j}$ are network weights at iteration j and ρ, σ are a series of activation functions from the neural network. The authors in [2] consider a least-squares loss for both the actor and critic networks, but we use the MSE loss instead.

A target and source network are initialized and updated for both the actor and critic. The weights are synched using a soft update rule to improve numerical stability. A vanilla actor-critic DDPG algorithm (ACDDPG) is given in algorithm 1.

Algorithm 1 Actor-Critic DDPG (ACDDPG)

```
1:  Randomly initialize the critic network $Q(s,a|\theta^Q)$ and
    actor network $\pi(s|\theta^\mu)$ with weights $\theta_Q, \theta^\pi$.
2:  Initialize a target network $Q'$ and $\pi'$ with the same weights
    as the networks above.
3:  Initialize an empty replay buffer $R$.
4:  for episode = 1 to M do
5:      Initialize a random process $\mathcal{N}$ for exploration.
6:      Obtain an initial observation of the state, $s_1$.
7:      for t = 1 to T do
8:          Select action $a_t = \pi(s_t|\theta^\mu) + \mathcal{N}_t$ according to the
            current policy and exploration noise.
9:          Execute action $a_t$ and observe the reward $r_t$ and new
            state $s_{t+1}$.
10:         Store the transition dictionary $(s_t, a_t, r_t, s_{t+1})$ to $R$.

11:         Sample a minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$
            from $R$.
12:         Set $y_i = r_i + \gamma Q'(s_{i+1}, \pi'(s_{i+1}|\theta^{\pi'})|\theta^{Q'})$.
13:         Update the critic network by minimizing the loss,
            given by equation (??).
14:         Update the actor network using the sampled policy
            gradient given by
            $$\nabla_{\theta^\pi} J \approx \frac{1}{N} \sum_i \nabla_a Q(s,a|\theta^Q)|_{s-s_i, a=\pi(s_i)} \nabla_{\theta^\pi} \pi(s|\theta^\pi)|_{s_i}.$$
15:         Perform a soft update of the target networks:
            $$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
            $$\theta^{\pi'} \leftarrow \tau\theta^\pi + (1-\tau)\theta^{\pi'}$$
16:     end for
17: end for
```

B. Twin-Delayed Soft Actor-Critic DDPG

The ACDDPG algorithm, while good, ends up most of the time overestimating the value function. The ACDDPG algorithm suffers from an overestimation of the value function. This overestimation propagates throughout the learning process and has a negative effect on the learned policy. This motivates the development of double Q-learning algorithms, where the action selection and Q-value updates are decoupled through the use of two value networks. In a setting like the ACDDPG algorithm, we keep track of two deterministic actors $\pi_{\theta_1}, \pi_{\theta_2}$ and two corresponding critics $Q_{\omega_1}, Q_{\omega_2}$ so that the Bellman updates are given by

$$\begin{aligned} y_1 &= r + \gamma Q_{\omega_2}(s', \pi_{\theta_1}(s')), \\ y_2 &= r + \gamma Q_{\omega_1}(s', \pi_{\theta_2}(s')). \end{aligned}$$

Clipped Q-learning, which uses the minimum estimated update between the two above, is applied so that the estimation of the value function is an underestimate. Biasing towards underestimation is preferred, since the underestimation of the value function is much harder to propagate through the learning process. The authors in [5] also consider a delayed soft-update of the network parameters, where the parameters of the policy are soft-updated at a lower frequency than the Q-function, so that the value function doesn't diverge when the estimate of the policy is poor. Finally, regularization is performed by adding a small amount of clipped noise to the action and updating is done in batches. With the clipping, the Bellman update takes the form

$$y = r + \gamma Q_\omega(s', \pi_\theta(s') + \epsilon)$$

where $\epsilon \sim \operatorname{clip}(\mathcal{N}(0,\sigma), -c, +c)$ is the clipped noise. The final twin-delayed actor-critic DDPG algorithm (TD3) is given in algorithm (cite here).

Algorithm 2 Twin-Delayed Actor-Critic DDPG (TD3)

```
    Initialize critic networks $Q_{\omega_1}, Q_{\omega_2}$ and actor network $\pi_\theta$ with random parameters $\omega_1, \omega_2, \theta$.
 2: Initialize the target networks $\omega_1' \leftarrow \omega_1, \omega_2' \leftarrow \omega_2, \theta' \leftarrow \theta$.
    Initialize a replay buffer $R$.
 4: for $t = 1$ to $T$ do
        Select action with exploration noise $a \sim \pi(s) + \epsilon, \epsilon \sim \mathcal{N}(0,\sigma)$ and record the reward $r$ and new state $s'$.
 6:     Store the tuple $(s, a, r, s')$ into $R$.
        Sample minibatches of $N$ transitions $(s, a, r, s')$ from $R$.
 8:     Smooth the target policy according to $\tilde{a} \leftarrow \pi_{\theta'}(s) + \epsilon, \epsilon \sim \operatorname{clip}(\mathcal{N}(0,\tilde{\sigma}), -c, c)$.
        Clip the results according to the followng rule $y \leftarrow r + \gamma \min_{i=1,2} Q_{\omega_i'}(s', \tilde{a})$.
10:     Update the critics according to $\omega_i \leftarrow \min_{\omega_i} N^{-1} \sum (y - Q_{\omega_i}(s,a))^2$.
        if $t \mod d$ then
12:         Update $\theta$ by the deterministic policy gradient according to the following rule
                $\nabla_\theta J = N^{-1} \sum \nabla_a Q_{\omega_1}(s,a)\big|_{a=\pi_\theta} \nabla_\theta \pi_\theta.$
            Update the target networks according to
                $\omega_i' \leftarrow \tau\omega_i + (1-\tau)\omega_i'$
                $\theta' \leftarrow \tau\theta + (1-\tau)\theta'.$
14:     end if
    end for
```

C. Multi-Agent Coverage Control Algorithm

Recall that the goal of area coverage is to maximize (2) by finding the optimal locations of the agents, p. Under the Voronoi partitions framework, it is shown in [1] that the optimal configuration is obtained by iteratively having agents move to the centroids of their Voronoi cells. The proposed area coverage control algorithm uses the Voronoi framework and uses the TD3 algorithm to learn suitable controls for each agent. In essense, we combine the power of reinforcement learning with the Voronoi partitions framework to converge to the centroidal Voronoi configuration, which solves (2), in an optimal manner.

References

[1] J. Cortés, S. Martínez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," *IEEE Transactions on Robotics and Automation*, vol. 20, pp. 243–255, April 2004.

[2] A. Adepegba, S. Miah, and D. Spinello, "Multi-agent area coverage control using reinforcement learning," 2016.

[3] C. Nowzari and J. Cortés, "Self-triggered coordination of robotic networks for optimal deployment," pp. 1039–1044, June 2011.

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015.

[5] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," 2018.

[6] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. TB, A. Muldal, N. Heess, and T. Lillicrap, "Distributional policy gradients," in *International Conference on Learning Representations*, 2018.

[7] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," 2017.