Deep Reinforcement Learning for Bearing-only Radiolocation

Cedrick Argueta
cdrckrgt@stanford.edu

Abstract

Unauthorized drones present a danger to airports and disaster areas. Localization and tracking of these unauthorized drones reduces some of this danger. It is possible to use a drone outfitted with with commericial antennas and radios to autonomously localize other drones. In this work, we show preliminary results detailing how a drone with this equipment may use deep reinforcement learning to perform path planning in a localization task.

1. Motivation

Tracking a moving radio target, such as a drone, is a useful and non-trivial task. An unauthorized drone could be disrupting airport operations, causing delays or interfering with aircraft. This task is not limited to tracking other drones: a wildlife radio tag could help with studying migration patterns of migrant animals. In this work, we model this problem as a dynamic system and apply a continuous control reinforcement learning algorithm, deep deterministic policy gradient (DDPG), as a solution method.

This work is intended to be a variant to work that I am pursuing for my honors thesis. My honors thesis revolves around this problem in the discrete setting, where there is imperfect information received from the environment. My honors thesis also involves the use of a sensor that gives very little information on the environment, resulting in a difficult sensor integration problem.

Here I intend on using a sensor that provides much more information of the environment, resulting in an easier version of my honors thesis. This balances the difficulty of considering continuous control over discretized control.

2. Mathematical Models

This section described the mathematical models used to formalize the system.

2.1. Drone Dynamics

The drone dynamics are exactly those described in [1]. The seeker drone state at time t is $x_t = [x_t^n, x_t^e, x_t^h]^\intercal$, where x_t^n and x_t^e are the seeker's north and east coordinates and x_t^h is the seeker's heading measured east of north. The state described here does not contain velocity or altitude as simplifying assumptions. The drone follows a first-order motion model, so the state after applying a control input u_t for duration Δt the new state is

$$x_{t+\Delta t} = x_t + u_t \Delta t \tag{1}$$

The target drone state at time t is $\theta_t = [\theta_t^n, \theta_t^e]^\intercal$, where θ_t^n and θ_t^e are the target's north and east coordinates. The target drone is assumed to move with a constant velocity $\dot{\theta} = [\dot{\theta_t^n}, \dot{\theta_t^e}]^\intercal$. The drone follows a first-order motion model, so the state after Δt is

$$\theta_{t+\Delta t} = \theta_t + \dot{\theta}_t \Delta t \tag{2}$$

2.2. Sensor Model

The bearing from the seeker drone to the target drone is

$$\beta_t = \arctan \frac{\theta_t^e - x_t^e}{\theta_t^n - x_t^n} \tag{3}$$

when measured east of north. Configured properly, the directional antenna and omnidirectional antenna can give estimates of the relative bearing of the target drone.

At time t, the seeker drone makes measurement $z_t \sim \mathcal{N}(\beta_t - x_t^h, \sigma^2)$, which is a bearing in the interval $[0°, 360°]$. It is assumed that these measurements are normally distributed about the true relative bearing with some variance to account for sensor error.

My honors thesis work uses a sensor that only provides a (noisy) boolean representing whether the target is in front of or behind the seeker, which provides much less information than a relative bearing measurement. The sensor model used here was verified experimentally in [2].

2.3. Particle Filter

The seeker drone maintains a belief of the possible location of the target drone, which is modeled with a particle filter. Belief at time t is represented by a set of N particles, each representing a hypothesis of the target drone's pose. Updates are made to the particle filter at every timestep to improve the belief's accuracy.

The belief update consists of three steps. The first step is the prediction step, where each particle is propagated according to the dynamics described in equation 2. Noise is added to the dynamics to prevent particle deprivation, a situation that arises when all particles converge to a hypothesis that doesn't accurately represent the true state. The second step is the weighting step, where each particle is assigned a weight according to how probable an observation z_t is given the particle's position. The third step is resampling, where particles are sampled according to these weights with replacement. In this work, we use stratified resampling to aid in maintaining an accurate estimate of the target while ensuring resiliency to particle deprivation.

2.4. Belief Markov Decision Process

The planning algorithm uses the partially observable Markov decision process (POMDP) framework for analysis.

A POMDP comprises a state space $\mathcal{S}$, an action space $\mathcal{A}$, a reward function R, and a transition function T defining the transition between states. An agent is unable to observe the true state s_t directly and instead makes an observation $\omega \in \Omega$ conditioned on the true state.

Solving a POMDP consists of finding a policy π^* such that

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(\omega_t))\right] \tag{4}$$

where γ is a discount factor.

POMDPs have a significant disadvantage when formalizing localization tasks. Rewards that depend on belief of the true state of the system are often difficult to represent in the POMDP framework [1]. For this reason, we instead convert the POMDP to a belief-Markov decision process, or belief-MDP.

Belief-MDPs are similar to MDPs where the system state is instead a *belief* of the true system state. We hereafter model the problem as an MDP where each state is a tuple of the fully observable part of the true state and the belief of the partially observable part of the true state.

2.5. Formulation

2.5.1 States

Each state is a tuple $s_t = (b_t, x_t)$ where b_t is the seeker's belief and x_t is the seeker's pose. The seeker's belief is the particle filter mentioned in the previous section.

2.5.2 Actions

The seeker is allowed to travel with a constant velocity towards any bearing. The seeker is also allowed to take no action.

The baseline controllers are discrete methods, and thus can only travel towards some bearings. Current experiments are run over 24 directions with a null action, resulting in 25 total discrete actions.

2.5.3 Reward Function

Our reward function for radiolocation captures the desire to maintain an accurate and precise estimate of the target's location *while also* maintaining an acceptable distance from the target.

A precise belief is one that has low uncertainty over the target's position. Minimization of this uncertainty is equivalent to minimization of the entropy of the belief distribution. Particles in the filter are first discretized into M bins. Entropy can then be defined as:

$$H(b_t) = -\sum_{i=1}^{M} \tilde{b}_t[i] \log \tilde{b}_t[i] \tag{5}$$

where $\tilde{b}_t$ is the proportion of particles in each bin. We normalize this entropy to be between 0 and 1, arriving at

$$H_n(b_t) = 1 - \frac{H(b_t)}{\log M} \tag{6}$$

.

An accurate belief is one that has a low tracking error with respect to the true target's state. Tracking error is

$$E(b_t, \theta_t) = \|\mathbb{E}[b_t] - \theta_t\| \tag{7}$$

but we again normalize the error to be between 0 and 1. We choose to divide by the maximum error in the search domain, which for a domain of length l is $l\sqrt{2}$. Our normalized tracking error is then

$$E_n(b_t, \theta_t) = 1 - \frac{E(b_t)}{l\sqrt{2}} \tag{8}$$

.

Near-collisions are penalized to encourage the seeker to keep a safe distance. The penalty term contains only the belief of the belief of the target's position rather than the true target position. This is to encourage the seeker to maintain a distance from the particles during evaluation. If the belief is representative of the true state, then the seeker will maintain a safe distance. If the belief is not representative of the true state, then the seeker will at least maintain a distance from the belief, which still might contain a noisy or partially accurate model of the target's motion. Our near-collision penalty is

$$C(b_t, x_t) = 1 - \mathbb{E}_{b_t} \mathbb{1}(\|x_t - \theta_t\| < d) \tag{9}$$

where $\mathbb{1}$ is an indicator function and d is the safe distance we wish the seeker to maintain.

The terms are combined to produce our full reward function:

$$R(b_t, x_t, \theta_t) = \lambda_1 H_n(b_t) + \lambda_2 E_n(b_t, \theta_t) + \lambda_3 C(b_t, x_t) \tag{10}$$

where λ_1, λ_2, and λ_3 are coefficients controlling the importance of each term.

3. Preliminary Experiments

3.1. Baseline

There are two current baselines for this experiment, both discrete. The first is a greedy method, which iterates over all possible actions and chooses that which maximizes the reward function for that step. The second is the standard deep Q-networks (DQN) algorithm, which makes Q-learning tractable with continuous observation spaces through the use of neural networks.

The greedy method represents a simple first step towards a solution, while DQN represents a first foray into deep reinforcement learning as a solution space. Both rely on a discretized action space, which is often done with continuous control problems to make them tractable.

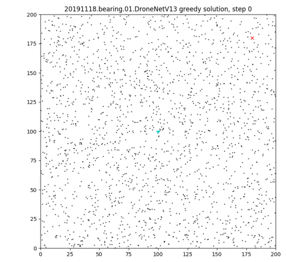
3.1.1 Greedy Method

The greedy method performs as expected. An example run is shown in figure 1. The greedy method has acculmated rewards from 8.0 to up to 45.0. The seeker will trail the target, concentrating belief behind the target with high tracking error. This is because the greedy method chooses the best action for that step, without taking into account that the target is moving with some unknown velocity.

The baseline is, however, still useful to illustrate the difficulty of this problem for learning-based systems – the controller must take into account that the target is moving, and figure out the target's approximate velocity by taking good actions that give the particle filter enough information.

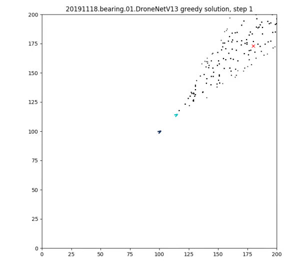
3.1.2 DQN

An exploration of DQN in this problem reveals that learning-based systems perform relatively better. It is well known that reinforcement learning models will exploit flaws in reward functions to get the maximum reward [3]. This is especially prevalent in the DQN baseline, where the seeker learns to minimize the entropy of the belief regardless of how good or bad the belief's accuracy is. Figure 3 shows how a higher reward is reached not necessarily by having better subjective performance but by maximizing the entropy reward from our cost function. Figure 2 shows the reward over all training episodes, representing a much higher reward than the greedy method.



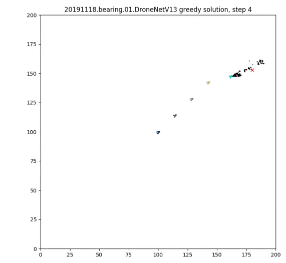
```
Entropy:   8.32, Max Eig:  3332.52, Max Prob: 0.000, Mean (vx, vy): (-0.0, -0.0)
Seeker Distance: 113.14, Tracking Error: 115.92, Step Reward: 0.00, Acc. Reward:   0.00
```

(a) Initialization.



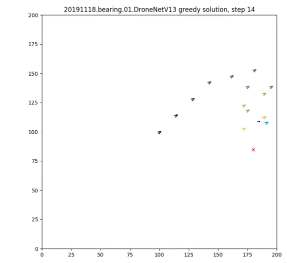
```
Entropy:   4.26, Max Eig:   876.00, Max Prob: 0.047, Mean (vx, vy): (0.1, -0.2)
Seeker Distance:  88.46, Tracking Error:  17.64, Step Reward: 0.17, Acc. Reward:   0.17
```

(b) After some steps.



```
Entropy:   1.44, Max Eig:    36.19, Max Prob: 0.549, Mean (vx, vy): (0.6, -1.3)
Seeker Distance:  18.98, Tracking Error:  12.18, Step Reward: 0.35, Acc. Reward:   1.68
```

(c) Belief mostly concentrated.



```
Entropy:   0.00, Max Eig:     0.00, Max Prob: 1.000, Mean (vx, vy): (0.7, -1.9)
Seeker Distance:  25.79, Tracking Error:  24.47, Step Reward: 1.40, Acc. Reward:  22.08
```

(d) Belief concentrated.

Figure 1: Progression of states for the greedy solver.

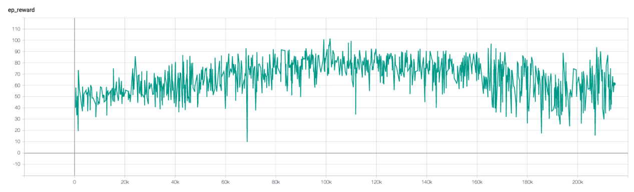
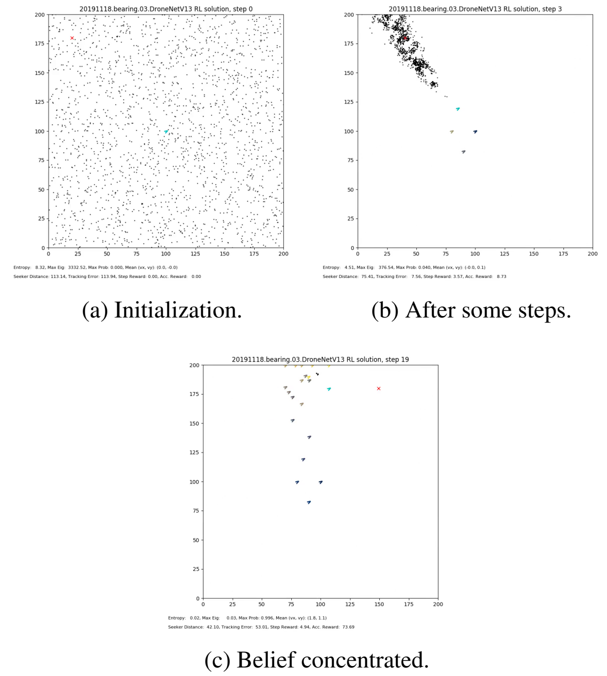


Figure 2: Reward curve over time.

The experiments run so far with DQN showcase how important it is to formulate the reward function in a way that incentivizes all the desired behavior without allowing for degenerate cases like this. While the DQN algorithm shows that a higher total reward is easily attainable by a reinforcement learning algorithm, it also reveals that the reward function must be reworked to incetivize the reduction of tracking error more. Further testing will need to be done with these baselines to determine a proper reward function that the continuous solution method can optimize well.

3.2. Secondary Metrics

Each solution method can be evaluated using the average accumulated reward over all episodes. However, a number of other metrics can be taken into account as well. Average time to reach a minimum level of performance (i.e. time to reach 50% confidence in our belief estimate) can be important when trying to take only approximately optimal steps. Also of interest is the execution times for different meth-



(a) Initialization. (b) After some steps.

(c) Belief concentrated.

Figure 3: Progression of states for the DQN solver.

ods: a continuous control algorithm that takes twice as long to produce a control input than a discrete control algorithm might not be worth the tradeoff for finer grained control. While I don't have these metrics baked into my simulator, I am considering them for the final report.

3.3. Solution Method

To solve this problem in the continuous setting, we use DDPG [4] and TD3 [5] and DQN [6]. This algorithm uses a neural network to approximate the value function and a neural network to approximate the policy.

3.3.1 Progress so far

I have written my own implementation of DDPG in PyTorch that is compatible with my simulator, but haven't yet tested it on the full problem. So far I have only solved the Pendulum environment from OpenAI Gym [7]. However, since I've written it in such a way that it interfaces well with my simulator, I will be able to run experiments in the continuous setting soon.

4. Contributions

I worked on this project alone.

Again, this project is derivative of my work for my honors thesis, but still represents different challenges. The sensor model used is different, specifically this project uses an 'easier' sensor model to work with. Also, my honors thesis so far has only considered discrete control, while the end goal of this project is to do continuous control. It will be useful to have a functioning implementation of DDPG that I can use in the future for this task.

References

[1] L. Dressel and M. J. Kochenderfer. Hunting drones with other drones: Tracking a moving radio target. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1905–1912, May 2019.

[2] L. Dressel and M. J. Kochenderfer. Pseudo-bearing measurements for improved localization of radio sources with multirotor uavs. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6560–6565, May 2018.

[3] Alex Irpan. Deep reinforcement learning doesn't work yet. `https://www.alexirpan.com/2018/02/14/rl-hard.html`, 2018.

[4] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[5] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477, 2018.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[7] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.