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. Details for CS229 Project

I am planning on making some improvements to a project that I'm working on for my honors thesis. The other sections of this paper are part of an early draft of the paper I'm writing up, which represent the basis for the CS229 project I want to do. Currently, I'm running experiments to finish this paper and submit it somewhere. I don't plan on using this *exact* project for CS229, but instead explore some possible improvements to this I could incorporate into another paper down the line. I have a well maintained simulation package for drone seeking problems that I will be using for this project, found here [1].

The gist of this problem is as follows: a seeker drone is trying to localize a target drone. The target drone is localized when the seeker drone has a proper idea of where the target drone is – the way this is done in this work is with a particle filter. When the particles are concentrated on the target drone's actual position, the target is localized. The seeker makes measurements in a search environment and uses these measurements to make decisions about where to move. These measurements are noisy relative bearing measurements of the target's position. The goal is to find a policy for the seeker that localizes the target quickly and with high accuracy and precision.

The work this is based on used MCTS for planning [2], and my mentor had the idea to use reinforcement learning to improve performance. Prior work suggests this is possible for fixed-wing drones [3] and drones in source-seeking applications [4]. I already have an implementation of DQN that can solve this problem, so I want to try different deep reinforcement learning algorithms. I primarily want to try either a recurrent network-based algorithm like DRQN or a continuous control algorithm like DDPG. The motivation behind DRQN would be that this problem is partially observed, and integrating information over time with an RNN might improve planning performance over DQN. Increased training time with the RNN scares me, so maybe this is where other CS229 concepts could help – some sort of dimenionality reduction or other preprocessing on the particle filter input would help keep training time realistic. DDPG would perhaps not be a strict improvement on performance but would instead allow the seeker drone to take finer actions than the finite limit I've currently set. Since it might take longer for a DDPG-based planner to converge to a good solution here, it might also make sense to use some sort of dimensionality reduction on the filter input.

The evaluation of the agent is based off that in [2]. The seeker should move to gather information from its sensors and concentrate particles in the filter to the true position of the target. This can be quantified by minimization of entropy and tracking error. In practice, maintaining distance from the target is useful – you avoid detection from the target, and prevent in-air collisions. Then a 'good' agent minimizes these three things. The goal is to show that for different collision rates, a learning-based planner will have a lower tracking error than an MCTS-based planner.

2. Introduction

The use of drones has become widespread in recent years. In the civil sector, drones can be used for disaster relief or surveillance. In the commercial sector, drones can be used for package delivery, photography or film production, or agriculture. Drones are also often used in the military for reconnaissance and as weapons.

The popularity of drones brings with it several challenges. Drones near airports pose a threat to aircraft operations, and carry the potential for a terrorist attack [5]. This year to date, the UK Airprox Board has recorded 18 "Airprox" events in which drones may have compromised the safety of aircraft [6]. Currently, the FAA prohibits drones from operating near rescue efforts in natural disasters like hurricanes or wildfires [7]. In these situations, it would be

beneficial to localize the position of the offending drone. Then it'd be possible to intercept and deal with the drone in a timely and safe manner.

Tracking a drone can be done by monitoring its radio telemetry. Drones usually emit radio signals to their operators, like speed, position, battery life, and other vital data. A seeker drone outfitted with radio receivers and antennas can home in on these radio signals and find the target drone's position. The sensor modality we consider is a directional antenna and an omnidirectional antenna, which has been shown to provide accurate bearing measurements on drones [8]. This combination of sensors allows for accurate bearing estimates of the target signal. While it is possible to track drones on the ground or with stationary trackers, a drone-based tracker would be as mobile as the target and be able to follow it for longer distances.

This work builds heavily off of [2] and [9] in that it has the same goal – improve drone tracking performance over greedy, one-step planners. A greedy solution would execute the best myopic action at every time step, without planning ahead to the next. The drone's path planning system aims to find a policy that chooses the best overall actions, perhaps sometimes making suboptimal myopic decisions in exchange for better hyperopic path planning.

This formulation of the problem has two drones: a seeker drone and a target drone. The seeker drone's objective is to track the moving target drone by capturing emissions by the target drone's radio. Existing solutions using Monte Carlo tree search (MCTS) often require large amounts of memory and computational power that might be lacking on drone avionics boards. Previous work shows how neural networks can be used for guidance of drones to waypoints with much smaller memory footprints than traditional Markov decision process (MDP) solutions in obstacle avoidance and waypoint finding [9]. This work applies the same methodology to the drone localization problem. We show that it is possible to train a deep reinforcement learning (RL) algorithm, namely Deep Q-Network (DQN), to perform path planning in a bearing-only case.

Our main contribution is the development of a learning-based planner that achieves a lower tracking error and near-collision rate compared to other planners.

3. Related Work

An MCTS solution to drone localization is presented by Dressel and Kochenderfer in [2]. The sensor modality used in this work consists of two directional antennas that are used to compare signal strength in front of or behind the drone. Our work differs in that we use a directional antenna and an omnidirectional antenna to provide pseudo-bearing measurements to the controller.

Using neural networks for path planning in drones is explored in [9]. In this work, the lookup table for the discrete value iteration solution is represented by a neural network.

Reinforcement learning for unmanned aerial vehicle (UAV) control is applied in [3]. Here, a fixed-wing UAV used for wilfire monitoring is controlled using deep RL.

Reinforcement learning for controlling multirotor UAVs is also explored in [4].

4. Mathematical Models

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

4.1. Drone Dynamics

The drone dynamics are exactly those described in [2]. 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

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

4.2. 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.

4.3. 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.

4.4. 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 [2]. 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.

4.5. Formulation

4.5.1 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.

4.5.2 Actions

While the belief-MDP framework is general enough to work with a continuous action space, this work focuses on the simpler discrete action space. The seeker drone is allowed to travel with a constant velocity in 24 directions equally spaced in a radial pattern.

4.5.3 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

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

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.

5. Planning Algorithm

We use deep reinforcement learning to approximate the optimal solution to this belief-MDP.

5.1. Deep Q-Networks

In the Deep Q-network (DQN) algorithm, state-action pairs are assigned a value $Q(s, a)$ representing the value of taking that action from that state [10]. This value is defined by the Bellman equation

$$Q(s, a) = R(s) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) \max_{a' \in \mathcal{A}} Q(s', a') \tag{11}$$

where $p(s'|s, a)$ is the probability of transitioning to state s' from state s after taking action a. Because computing the true value of Q for every state-action pair is intractable, we use a neural network to approximate Q. We train this neural network by minimizing the Bellman error

$$E_{Bellman} = R + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \mathbf{w}) - Q(s, a; \mathbf{w}) \tag{12}$$

with gradient descent. The optimal policy derived from the Bellman equation is then

$$\pi^*(s) \approx \arg\max_{a \in \mathcal{A}} Q(s, a; \mathbf{w}) \tag{13}$$

In practice, we apply the double Q-learning modification from [11] and dueling network architecture from [12].

5.2. Network Architecture

A neural network is used to approximate the Q value function. In a style similar to [3], a two-stream architecture (Figure 1)is used.

The input to the neural network at time t is the belief b_t and the seeker's pose x_t. The particle filter representing belief is first discretized to a 2d histogram. Convolutional layers are then used to extract spatial information from this downsampled belief. Only particle positions are used in this downsampling – the mean of the velocity of all particles is concatenated to x_t. The downsampling allows us to take advantage of the spatial relationships that particles have.

6. Preliminary Results

6.1. Simulation Parameters

Training the planner on the physical system is infeasible because of time constraints – the training environment must be reset every episode, human intervention is required to replace batteries, and weight updates are limited by the frequency of actions. Thus, the network is trained on a simulator that captures the essential aspects of the system described in section 4. The simulator code may be found at [1].

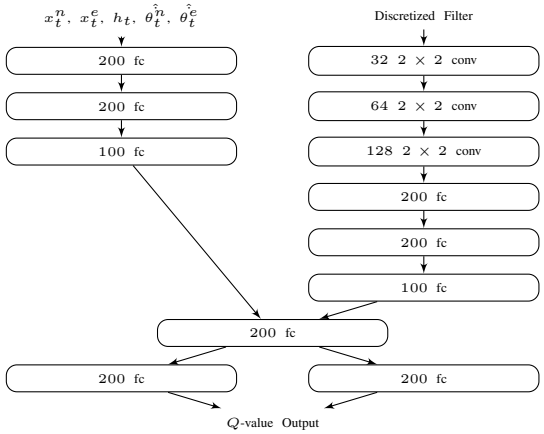

Figure 1. A two-stream dueling architecture is used to approximate the Q value function. `fc` denotes a fully-connected layer while `conv` denotes a convolutional layer.

The seeker and target drones are modeled in a 200 $m\times$ 200 m area, where the seeker begins each episode at the center of the area and the target begins at a randomly selected corner and travels to an adjacent corner at 1.7 m/s. The seeker drone can move at 5 m/s in 36 equally-spaced directions or take no action. The particle filter has 2000 particles, uniformly distributed at initialization and pruned using systematic sampling. The values of λ_1, λ_2, and λ_3 were found empirically.

6.2. Baseline

The planner is compared against two baselines found in [2], a UCT-based MCTS planner and a greedy planner. Both these methods are online and require no training. For this reason, the cost function for agents using these planners do not use the tracking error term, as this would involve incorporating information into the solution method that the planner does not have access to.

6.3. Results

7. Conclusion

References

[1] Cedrick Argueta. Python filter exploration for bearing only localization, 2019. Available at https://github.com/cdrckrgt/PyFEBOL.

[2] Louis Dressel and Mykel J. Kochenderfer. Hunting drones with other drones: Tracking a moving radio target.

[3] Kyle D. Julian and Mykel J. Kochenderfer. Distributed wildfire surveillance with autonomous aircraft using deep reinforcement learning. *CoRR*, abs/1810.04244, 2018.

[4] Bardienus P. Duisterhof, Srivatsan Krishnan, Jonathan J. Cruz, Colby R. Banbury, William Fu, Aleksandra Faust, Guido C. H. E. de Croon, and Vijay Janapa Reddi. Learning to seek: Autonomous source seeking with deep reinforcement learning onboard a nano drone microcontroller, 2019.

[5] Associated Press. Worried about drones flying too close to airports? here are answers to your questions, Jan 2019.

[6] UK Airprox Board. Monthly meeting march 2019, Mar 2019.

[7] Federal Aviation Administration. Airspace restrictions, Dec 2018.

[8] 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.

[9] Kyle D. Julian and Mykel J. Kochenderfer. *Neural Network Guidance for UAVs*. 2017.

[10] 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.

[11] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[12] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015.