Learning to Seek: Autonomous Source Seeking on a Nano Drone Microcontroller with Deep Reinforcement Learning

Bardienus P. Duisterhof[1,3] Srivatsan Krishnan[1] Jonathan J. Cruz[1] Colby R. Banbury[1] William Fu[1]

Aleksandra Faust[2] Guido C. H. E. de Croon[3] Vijay Janapa Reddi[1,4]

***Abstract*— Nano drones are uniquely equipped for fully autonomous applications due to their agility, low cost, and small size. However, their constrained form factor limits flight time, sensor payload, and compute capability, which poses a significant limitation on the use of source-seeking nano drones in GPS-denied and highly cluttered environments. The primary goal of our work is to demonstrate the effectiveness of deep reinforcement learning in fully autonomous navigation on highly constrained, general-purpose hardware and present a methodology for future applications. To this end, we present a deep reinforcement learning-based light seeking policy that executes, in conjunction with the flight control stack, on a commercially available off-the-shelf ultra-low-power microcontroller (MCU). We describe our methodology for training and executing deep reinforcement learning policies for deployment on constrained, general-purpose MCUs. By carefully designing the network input, we feed features relevant to the agent in finding the source, while reducing computational cost and enabling inference up to 100 Hz. We verify our approach with simulation and in-field testing on a Bitcraze CrazyFlie, achieving 94% success rate in a highly cluttered and randomized test environment. The policy demonstrates efficient light seeking by reaching the goal in simulation in 65% fewer steps and with 60% shorter paths, compared to a baseline 'roomba' algorithm.**

I. Introduction

In recent years, nano drones have gained traction in the robotics community. Their agility, maneuverability, and low price, make them suitable for a wide range of applications, especially in GPS-denied and cluttered environments. However, applications of nano drones have had limited success in the real world due to their form factor and size, which impose constraints on onboard compute and sensor payload. So far, only with specialized hardware [1], [2], inference of neural networks on nano drones has been achieved.

In this paper, we focus on source seeking, where the objective is to navigate safely and locate a source (e.g., gas, light, radiation). A source-seeking robot has many real-world applications, ranging from finding an exit in a collapsed building to locating a radiation or gas source. The ideal source-seeking robot is small, agile, and cheap, making it good at moving through narrow spaces while reducing the impact of a possible collision with obstacles in its path.

To the best of our knowledge, we introduce the first deep reinforcement learning (Deep-RL) based source-seeking nano drone that is fully autonomous. Our source-seeking nano drone, shown in Figure 1, seeks a light source. It uses a custom light sensor (Figure 2) to track light intensity, a Bitcraze laser-based multiranger to detect obstacles, a Bitcraze flow deck to track its motion on the z-axis and $x-y$ plane, and only an ultra-low-power Cortex-M4 microcontroller. The microcontroller does all the processing required in real-time without any external assistance with only 196 kB of RAM for the full flight stack and machine learning model.

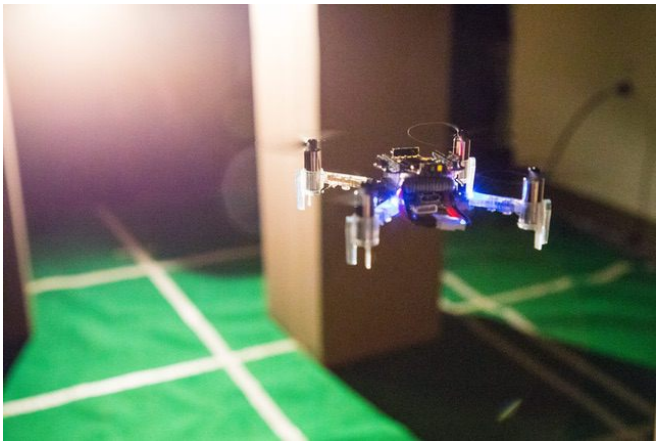


Fig. 1. CrazyFlie nano drone running a deep reinforcement learning policy fully *onboard*. It performs the computation online using a low-power Cortex-M4 microcontroller. It uses a light sensor to locate the source while avoiding obstacles with a multiranger and an optical flow sensor for flight stability.

Our light-seeking nano drone application is inspired by nature, where an organism responds either positively or negatively to a light source, moving toward or away from the source, respectively. Even though it is not entirely clear why insects are attracted to light, various theories exist. They use these sources as a means of navigation, learned light sources radiate heat in cold environments or find their fellow insects at the source [3]. Such learned behavior can be useful for aerial robots to find the exit out of a collapsed building or help a solar-powered nano drone recharge effectively.

When designing a fully autonomous and Deep-RL based nano drone, the fundamental challenges fall into two distinct, but not completely independent, categories: (1) application modeling and (2) algorithm design. Application-specific challenges include modeling the light source appropriately in a simulation where the agent (i.e., drone) learns to find the light source over repeated trials. Algorithm design consists of designing a neural network model and developing a deep reinforcement learning-based policy that successfully navigates the drone to the source while avoiding obstacles.

For application modeling, we extend the Air Learning

[1]Harvard University, [2] Robotics at Google, [3]Delft University of Technology, [4] The University of Texas at Austin - bduisterhof@g.harvard.edu. The work was done while Bart was a visiting student at Harvard.

simulation environment for training the drone to find the light source [4]. Air Learning is an AI-research platform that models a task, such as source seeking, in a randomly generated environment with (or without) a variety of obstacles. We modify the Air Learning platform to model the light source intensity as a function of distance from the source.

For algorithm design, we use a Deep-RL based source seeking solution. Our work is an implementation of Deep-RL for source seeking. Deep-RL has shown to perform well in the presence of sensor noise [5], while capable of finding high-performance solutions to complex tasks [6]. We train a Deep Q-Network (DQN) [7] to locate the source. DQN uses a neural network policy to approximate the Q-function whose objective is to maximize the long term reward.

In the source seeking application, the objective of the Q-function is to minimize the distance from the source (i.e., to seek the source) to maximize the reward. The input to the neural network policy is obstacle distance from the laser ranger and readings from the light sensor. The last layer of the neural network policy maps to the action space. For the source seeking application, the action space includes three actions (`rotate left`, `right`, or `move forward`). We feed two source terms, $s1$ and $s2$, constructed from source strength c. These features are designed to maximize performance within the tiny neural network that we are able to fit on the nano drone.

The neural network model is evaluated through both simulation and flight tests onboard a CrazyFlie nano drone. We present and discuss the reward, success rate, and the distance traveled. We present a wide range of flight tests, where success is defined as reaching close to the light source, and failure occurs when the nano drone runs out of battery or crashes into a wall or an obstacle. On average, we observe a success rate of 94 % during real-world flight tests in highly cluttered and randomized testing environments.

While we focus exclusively on light-seeking as our application in this paper, we believe that the general methodology we have developed for deep reinforcement learning-based source seeking (i.e., application modeling, algorithm design) can be readily extended to other applications as well.

In summary, we make the following contributions:

- We present the first application of deep reinforcement learning for source seeking using light as the source, which proves useful in situations such as finding an exit path out in a search and rescue situation. We present a series of results in simulation and flight tests to demonstrate robust source seeking behavior.
- We introduce a novel methodology for deploying deep reinforcement learning models onto microcontroller units that are typically found in nano drones, as opposed to relying on custom-designed AI hardware. We show how a microcontroller with only 196 kB of RAM and one low-power core can be used to perform inference of a deep reinforcement learning policy.
- Using this methodology, to the best of our knowledge, we introduce the first end-to-end navigation by deep reinforcement learning onboard a nano drone.

In the remainder of this paper, we go through the methodology and results. In Section III-A we lay down vehicle configuration, whereafter we describe our simulation environment in Section III-B. In Section III-C, we provide a detailed description of the algorithm with all hyperparameters. Finalizing our methodology, in Section III-D, we describe our strategy for deploying a deep reinforcement learning policy onboard a nano drone. In Section IV, we introduce a baseline algorithm, show training progress, test the different models in simulation, show a wide range of flight tests, and finally perform analysis of endurance and power. In Section V we present a discussion and conclude the paper in Section VI.

II. Related Work

From the algorithm point of view, two categories are relevant to our work: 1) deep reinforcement learning algorithms; and 2) previous source seeking algorithms. Deep reinforcement learning has proven to be a promising set of algorithms for robotics applications. The fast-moving deep reinforcement learning field is enabling more robust and accurate but also more effortless application [8]. Not only have robotic manipulation [6] and locomotion [9] benefited from deep reinforcement learning, but also UAV control is considered to be an area of application. Lower-level control has been demonstrated to be suitable to replace a rate controller [10], [11], [12] and was recently used to perform model-based reinforcement learning (MBRL) with a CrazyFlie [13]. High-level control using deep reinforcement learning for obstacle avoidance has been shown with different sets of sensory input [14], [4]. Although light-seeking has been demonstrated before using Q-learning [15], *we present a deep reinforcement learning-based model for source seeking running entirely onboard a nano drone microcontroller*.

Source seeking algorithms can be divided into four categories [16]: 1) gradient-based algorithms, 2) bio-inspired algorithms, 3) multi-robot algorithms, and 4) probabilistic and map-based algorithms. Even though gradient-based algorithms are easy to implement, their success in source seeking has been limited due to their unstable behavior with noisy sensors. Previous bio-inspired algorithms have yielded promising results, but excluded obstacle avoidance. Additionally, to the best of our knowledge, none of the bio-inspired source seeking algorithms have been successfully implemented on a robot. *We implement a learning-based algorithm capable of robust autonomous source seeking.*

In a multi-agent scenario, particle swarming [17], [18] has shown to be particularly successful. Downsides of multi-agent source seeking include higher price, required communication, and localization, and a higher risk of collision. Finally, probabilistic and map-based algorithms are more flexible but require high computational cost and accurate sensory information. However, in contrast to traditional methods, deep reinforcement learning can learn to deal with noisy inputs [5] and effectively learn (optimal) behavior for a combination of tasks. Hence, source seeking on a nano drone is a suitable task for deep reinforcement learning, as it can combine obstacle avoidance with source seeking and deal with extraordinary noise levels in all sensors. *We leverage these advantages of deep reinforcement learning to produce a robust and efficient algorithm for source seeking.*

Developer	Bitcraze	Parrot
Vehicle	CrazyFlie 2.1	Bebop 2
Takeoff weight	27 g	500 g
Max payload	15 g	70 g
Battery (LiPo)	250 mAh	2700 mAh
Flight time	7 min	25 min
Size (WxHxD)	9.2 cm x 9.2 cm	32.8 cm x 38.2 cm

TABLE I

CRAZYFLIE 2.1 VERSUS BEBOP2 DRONE PLATFORM SPECIFICATIONS.

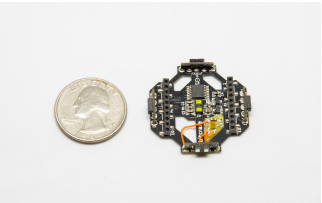


Fig. 2. Close-up of a BitCraze multiranger deck, fused with our custom TSL2591 light sensor board.

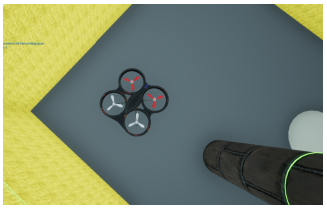


Fig. 3. Air Learning simulation with obstacle(s) and a light source. The picture is illuminated for clarity.

III. METHOD

We start by explaining the characteristics of our nano drone (Section III-A), which sets the constraints within which we must deploy a neural network model. Next, we describe our simulation setup (Section III-B) used to train deep reinforcement learning policies by randomizing the training environment. In Section III-C, we describe the Deep Q-Learning algorithm along with all of its inputs and hyperparameters. Finally, in Section III-D, we describe our strategy for deploying the deep reinforcement learning policy onboard the CrazyFlie.

A. Vehicle Configuration

The goal of our work is to demonstrate the effectiveness of deep reinforcement learning in fully autonomous navigation on highly constrained, general-purpose hardware and present a methodology that can be used to enable a range of applications, in addition to our source seeking application. To this end, we use the BitCraze CrazyFlie 2.1 as our research platform. It is agile yet physically and computationally limited. The physical characteristics of the CrazyFlie are captured in Table I, emphasizing its small size, weight, and power. For instance, the Bebop 2 has 4× more flight time and payload weight, and nearly 10× the battery capacity.

Similarly, the CrazyFlie's computational capabilities are shown in Table II [19]. We compare the CrazyFlie's compute system's capabilities (STM32F405) against a compute system (NVIDIA TX2) that is typically mounted on the Parrot Bebop 2 [20]. The former has a single low-power core, while the TX2 has six high-performance cores in addition to a GPU. The 50× power difference confirms this difference in computational capability. Additionally, we compare the STM32F405 to the Greenwaves GAP8 [1] co-processor. The GAP8 is a microcontroller unit, similar to the STM32F405 on the CrazyFlie, but it is coupled with a programmable 8-core accelerator that has specialized instruction support for digital signal processing. Compared to the GAP8, the STM32F405 has significantly less memory and compute power.

While specialized hardware such as GAP8 can lead to superior performance, the vast majority of nano drones typically come equipped with low-power general-purpose MCU's. Hence, it is beneficial to utilize existing hardware when developing learning-based algorithms. What makes this task especially challenging is that the MCU must not only run the neural network, but it must do so while time multiplexing the limited processing cycles with the execution of the flight control stack.

The limitations of the platform go beyond just computation, as the sensor payload is also constrained. Due to the lightweight nature of the CrazyFlie, every additional gram of payload significantly impacts power consumption and thereby endurance. With the impact on endurance in mind, we equip the CrazyFlie with three sensors: (1) the Bitcraze laser multiranger to measure the distance in the front/back/left/right directions, which we use to avoid obstacles, (2) the Bitcraze flow deck to track motion on the z-axis and $x-y$ plane to ensure stable flight, and (3) a custom light sensor to help locate the source by measuring the illuminance at a specific point in time and space. The light sensor sits on the Bitcraze multiranger deck, and is integrated with the CrazyFlie firmware via a deck software driver. The light sensor faces upwards, as shown in Figure 2.

B. Simulation Environment

The first building block of our methodology is the simulation or the training environment. We use the Air Learning platform [4], which couples with Microsoft AirSim [21] to provide a deep reinforcement learning back end. It allows us to generate a variety of different environments with randomly varying obstacle density, materials, and textures.

Using Air Learning, we simulate a training environment that is an arena. The arena is "spawned" at 5×5 m in size, similar to our flight test environment. The agent (i.e., drone) is initialized in the middle of the room, and the light source is spawned at a random location. In addition, Air Learning is configured to spawn obstacles in random locations within the room. The agent must learn to traverse the room without running into obstacles or running out of (300) steps before finding the light source. Figure 3 shows a screenshot from one of our simulation experiments.

The agent's actions space consists of moving `forward`, rotating `left` and rotating `right`. The forward-moving speed is 0.5 m/s, and the yaw rate is 54°/s in either direction. The length of each action is set to 0.25 s, meaning the rate controller will command $\dot{\psi} = 54°/s$ or $v_x = 0.5\ m/s$ (NED frame) for 0.25 s. The policy is trained on a computer with a powerful CPU and GPU (i.e., i9 9900K and RTX 2080 TI).

We predict light intensity as a function of the distance from the source. We generate this function by capturing data in the testing environment with the light source present. We capture the light intensity in a two-dimensional grid with our light sensor. In our testing environment, we have a 50 W LED

Name	STM32F405	GAP8	TX2
CPU	1-core @168MHz	1-core @ 250MHz (FC)	6-core @ 2GHz
GPU	None	8-core @ 250MHz (CL)	256-core @1300 MHz
Memory	196 kB	64 kB L1, 512kB L2, 8 MB L3(off chip)	8 GB
Flash	1 MB	16 MB(off chip)	32 GB
Power	0.14 W (max)	0.28 W (average@250MHz)	7.5 W

TABLE II

CRAZYFLIE MICROCONTROLLER SYSTEM VERSUS A GREENWAVES GAP8 AND NVIDIA TX2.

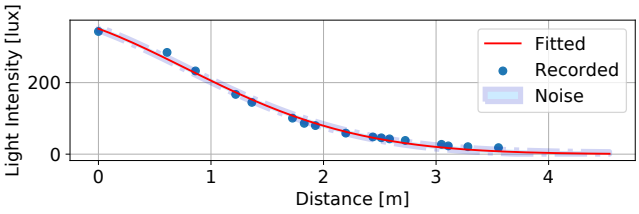


Fig. 4. Light intensity describing the function as used in training with 3σ (standard deviations) of the noise.

light, hanging from the roof, which creates a spotlight on the ground or at an angle.

Once captured, we use the data to fit a function with two requirements: 1) $\lim_{dist \to 0} < \infty$ and 2) $\lim_{dist \to \infty} = 0$. In other words, the function must be bounded when the agent is placed under the source, while also reaching zero at infinite distance from the source. A Gaussian function meets both requirements and is shown in Figure 4. The function has the form: $f(x) = a \cdot e^{-\frac{(x-b)^2}{2c^2}}$ with $a = 399.0, b = -2.6, c = 5.1$. The R-squared error, measuring the goodness-of-fit, is 0.007, implying a high-quality fit. Additionally, we inject Gaussian noise with a standard deviation of 2. The noise observed in recordings had a standard deviation of 2; however, in flight with less ideal power supply and unstable attitude, we expect more noise. To account for that, we inject more noise than recorded. In flight tests, we present the robustness of this function when shadows and reflections are present. In future work, a more sophisticated model incorporating shadows and reflections may be incorporated.

C. Policy Selection and Reward Shaping

We train and deploy a Deep-Q-Network (DQN) [7] algorithm on the CrazyFlie. Lower-level control is carried out by a set of PID controllers, while higher-level actions ($\dot{\psi}, v_x$, NED-coords) are chosen by the policy.

Within Deep Q-learning, a variety of neural network architectures can be used to predict Q-values for the possible states. The policy architecture is shown in Figure 5. Terms $L_1 - L_4$ are the multi-ranger readings in all four directions (i.e. right, front, left, back).

Additionally, we add two source-related terms. The challenge is to design features useful for a tiny neural network, maximizing navigation robustness and efficiency. We first compute a normalized version of the sensor readings, as sensor readings are dependent on sensor settings (e.g. integration time, gain).

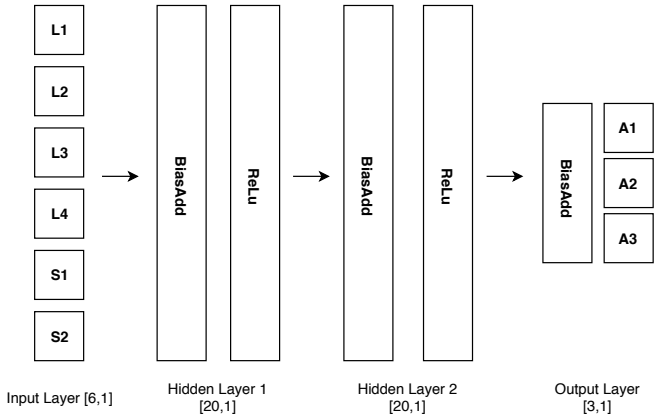


Fig. 5. Visualization of the general network architecture.

$$c = \frac{s_{\text{meas}} - s_{\text{min}}}{s_{\text{max}} - s_{\text{min}}} \tag{1}$$

We then create a low-pass version of c, c_f.

$$c_f \leftarrow 0.9 \cdot c_f + 0.1 \cdot c \tag{2}$$

Finally, inspired by [22], we compute terms s_1 and s_2.

$$s_1 = \frac{c - c_f}{c_f} \tag{3}$$

$$s_2 = 2 \cdot c_f - 1 \tag{4}$$

Term s_1 provides information on the change of source intensity compared to previous readings, similar to a gradient measurement. Term s_2 gives the agent a measure of how close it is to the source. If c_f is 0.5, and s_2 0, the agent is about $1.5\,\text{m}$ from the source (Figure 4). This means the agent has a clear signal on it being close to the source, altering its behavior. In general, we observed it flying `forward` more when it was close to the source.

We have trained policies with gradient inputs, leading to poor results. As discussed in [16], gradient-based algorithms are susceptible to disturbances. In the source seeking application, an ample amount of noise is present, explaining why a gradient-based approach may lead to poor results in this specific application. Our two features are an approximation of gradient (s_1) and distance to the source (s_2), leading to robust obstacle avoidance and source seeking.

The agent succeeds when it reaches within one meter of the light source in the simulation environment. To teach the

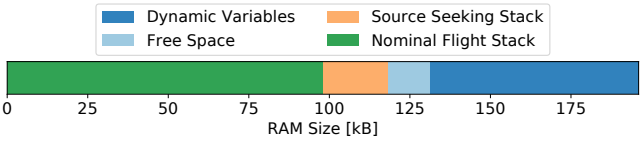


Fig. 6. RAM usage on the Bitcraze CrazyFlie, using a custom float inference stack. Total free space: 12.5 kB

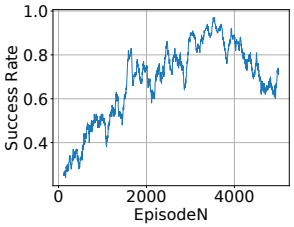


Fig. 7. Training success rates.

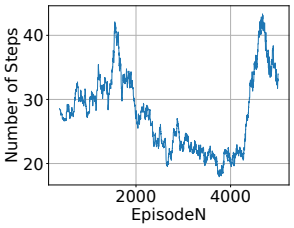


Fig. 8. Steps to locate source.

nano drone to seek the light source, the following reward is computed at each step (instantaneous reward):

$$r = 1000 \cdot \alpha - 100 \cdot \beta - 20 \cdot \Delta D_s - 1 \quad (5)$$

Here, α is a binary variable whose value is '1' if the agent reaches the goal else its value is '0'. β is a binary variable which is set to '1' if the nano drone collides with any obstacle or runs out of the 300 steps.[1] Otherwise, β is '0', effectively penalizing the agent for hitting an obstacle or not finding the source in time. ΔD_s is the change in distance, compared to the previous step. A negative ΔD_s means the agent got closer to the source, rewarding the agent to move closer to the source.

D. Deploying on the Nanodrone

Our implementation consists of a C library, capable of performing the necessary inference operations. The advantage of this approach is its small memory footprint and overhead, compared to a general inference framework like TFLite.

The Crazyflie has 1 MB of flash storage. The stock software stack occupies 192 kB of the available storage, while the custom source seeking stack takes up an additional 6 kB. So the total flash storage used is 198 kB, which leaves an ample amount of free storage space (over 75%).

However, the memory constraints are much more severe. RAM availability during execution is shown in Figure 6. Of the 196 kB of RAM available on the Cortex-M4 microcontroller, only 131 kB is available for static allocation at compile time. The rest is reserved for dynamic variables (i.e., heap). During normal operation, the Bitcraze software stack uses 98 kB of RAM, leaving only 33 kB available for our purposes. The entire source seeking stack takes up 20.5 kB, leaving 12.5 kB of free static memory. The policy as shown in Figure 5, runs up to 100 Hz in flight with this approach.

IV. Results

In this section we evaluate simulation and flight results of the models considered. We introduce our baseline model (Section IV-A), discuss simulation results (Section IV-B, IV-C) and evaluate flight tests (Section IV-D). Finally, we analyze endurance and power in Section IV-E.

A. Baseline Algorithm

To evaluate how well our deep reinforcement learning-based approach performs, we compare our approach to more traditional solutions. We implement a baseline algorithm that performs obstacle avoidance without any specific information about the light source. It is similar to an autonomous cleaning robot that randomly scans a room until it finds dirt to focus its vacuuming effort [23]. We will therefore refer to the baseline as the 'Roomba' approach, which is a relatively effective exploration algorithm [23]. The algorithm simply rotates the drone at a random angle when the front laser ranger detects an obstacle under 2 meters. This threshold is a trade-off between exploration and obstacle avoidance.

[1]We set the maximum allowed steps in an episode as 300. This is to make sure the agent finds the source within some finite amount of steps and penalize a large number of steps.

Model description	Success	# of Steps	Distance [m]	SPL
Our deep-rl algorithm	96%	30.51	4.21	0.37
Roomba Approach	84%	87.73	10.40	0.16
Random actions	30%	42.13	3.55	0.13

TABLE III
MODELS EVALUATED IN SIMULATION.

The baseline algorithm's results are shown in Table III, along with the other results. The success rate is 84%. From observations, we see that the front-ranger is unable to detect all of the obstacles and hereby fails to find the source in 16% of the runs. Additionally, we have tested fully randomized actions. This algorithm decides randomly between `moving forward`, `right` or `left`. This model serves to put our baseline and deep reinforcement learning algorithm into perspective.

B. Training in Simulation

To evaluate the learning process, we present quality metrics (success rate and number of steps) during training. As shown in Figures 7 and 8, we train our policy up to convergence at around 3600 episodes (or 100,000 steps). A consistent upward trend in success rate is visible, while number of steps shows a consistent decrease after an initial spike. The initial spike is caused by the agent becoming more successful, hence reaching more distant targets, instead of only finding close targets. After continued training success rate quickly drops, i.e., it over-trains after around 3600 episodes. We continued training to over 8,000 episodes and never saw an improvement in performance.

C. Inference in Simulation

Training data provide limited information as the model is continuously changing. So, we evaluate our policy after training (as shown in Table III). We compare it in terms of success rate, the average number of steps, and average traveled distance. The number of steps and traveled distance

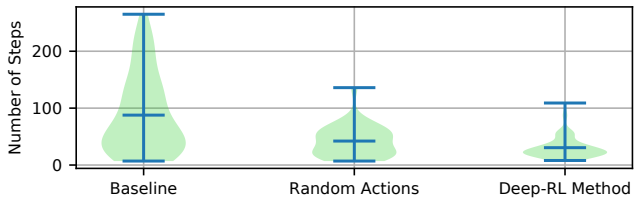


Fig. 9. Number of steps in simulation over 100 runs of each algorithm.

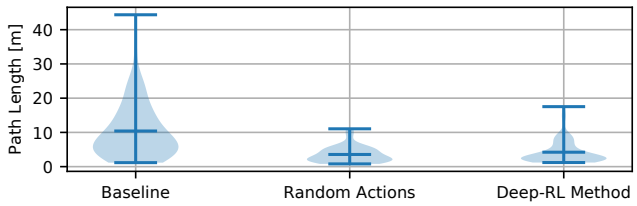
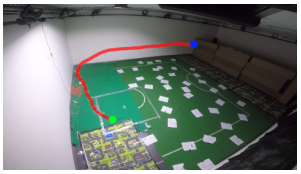
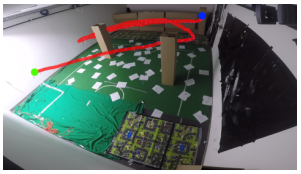


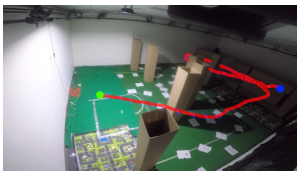
Fig. 10. Path length (i.e. traveled distance) in simulation over 100 runs of each algorithm.



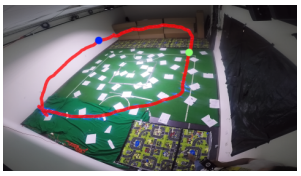
(a) Zero obstacles trajectory, finds source in direct trajectory.



(b) Three obstacles trajectory, finds source by avoiding obstacles.



(c) Seven obstacles trajectory, finds source by avoiding obstacles.



(d) Source position in center, does not find source. Out of scope of simulation environments.

Fig. 11. Four distinct trajectories in flight tests. The blue dot means the start of the trajectory, while the green dot highlights the end of the trajectory.

are captured only when the agent succeeds. Additionally, we add the 'SPL' metric: Success weighted by (normalized inverse) Path Length [24]. We compute the SPL metric as follows:

$$SPL = \frac{1}{N} \sum S_i \frac{l_i - 1}{\max(p_i, l_i - 1)} \tag{6}$$

Here N is the number of runs considered, l_i the shortest direct path to the source, p_i the actual flown path and S_i a binary variable, 1 for success and 0 for failure. We subtract l_i by 1, as the simulation is terminated when the drone is within 1 meter of the source. We do not take into account obstacles in the path, making the SPL displayed a conservative estimate of the actual value.

We evaluate our approach, the Roomba baseline, and fully random actions in simulation. Table III and Figures 9, 10 show the results of testing each method for 100 runs. Our deep reinforcement learning model outperforms the baseline in every metric. It finds the source in 65% fewer steps, with a 14% higher success rate, with 60 % shorter paths and a 131% higher SPL. Random actions yield shorter successful paths, as shorter paths have a higher chance of survival with random actions. In fact, the average path length over all (successful and failed) attempts for the random approach is $5.7\,\mathrm{m}$, while $4.19\,\mathrm{m}$ for our approach.

D. Flight Tests

For flight tests, we use a room that is approximately $5\,\mathrm{m} \times 5\,\mathrm{m}$ in size (see Figure 11). We use a $50\,\mathrm{W}$ light source attached to the roof, radiating a $120°$ beam onto the ground, as the light source. We count a distance under $0.7\,\mathrm{m}$ as a successful run, while the drone is flying at $1\ \mathrm{m/s}$ and performs inference between $50\,\mathrm{Hz}$ and $100\,\mathrm{Hz}$. Figure 11 show four distinct trajectories during the extensive testing of the algorithm.

We conduct 114 flight tests in a variety of different scenarios, involving highly cluttered environments. Across a set of representative flight tests, we get an average success rate of 94%. This number is measured over a set of 32 experiments, 16 with no obstacles, and 16 with 3 obstacles in the room. This is representative to simulation as it alternates between no obstacles and sparse obstacles. All agents were initialized at different positions, at the same $4.6\,\mathrm{m}$ from the source on one axis. On the other axis, we varied drone position between $0\,\mathrm{and}\ 4.6\,\mathrm{m}$ from the source, resulting in an initial total source distance between $4.6\,\mathrm{and}\ 6.5\,\mathrm{m}$.

Obstacle configuration, density, and source position were randomized. We classify three distinct obstacle densities: 'NO OBS', 'LOW OBS', and 'HIGH OBS', featuring zero, three and seven obstacles respectively.

To better understand the behavior of our algorithm, we decompose the results into three categories: 1) success rate; 2) mission time, and 3) failure cases.

1) Success Rate: Over a total of 104 flight tests, we compared success rate of our model against the Roomba baseline model. As can be seen in Figure 12, our model beats the baseline in all three obstacle density groups. The baseline reaches a 75% success rate in a set of representative flight tests ('NO OBS' and 'LOW OBS'), compared to a 84% success rate in simulation.

These results demonstrate that obstacle avoidance using solely a multiranger is challenging, as drift and limited visibility are the most prominent causes for crashing. In most crashes, the drone would either not see the obstacle or keep rotating close to an obstacle and eventually crash into the obstacle due to drift. The baseline serves to put our algorithm into perspective, i.e., it shows the robustness of our obstacle avoidance and source seeking strategy. A deteriorated success rate in the 'HIGH OBS' scenario is expected, as it has never seen such a scenario in simulation. Despite some loss in success rate, our approach demonstrates greater resilience to increased task complexity when compared to the baseline.

2) Mission Time: Our objective is not only to perform successful obstacle avoidance, but also to find the sourced in as little time as possible. Nano drones are characterized by their limited battery life. Therefore, efficient flight is an

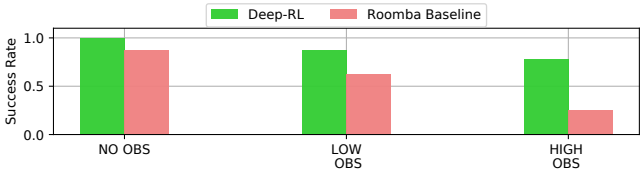


Fig. 12. Success Rate over 104 flight experiments, comparing our deep-rl approach with the Roomba baseline.

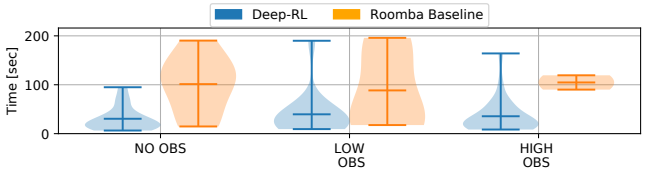


Fig. 13. Mission time in success over 104 flight experiments, comparing our deep-rl approach with the Roomba baseline.

important metric when evaluating the viability of an approach for real applications. Figure 13 shows the distribution of the mission time of successful runs, demonstrating an impressive advantage for our algorithm. Over the different obstacle densities, from low to high, our policy found the source 70%, 55%, and 66% faster respectively.

The baseline is again used to put our model into perspective. As demonstrated in [23], the Roomba approach is effective in exploring an area without any source information. Because of its random character, the baseline shows a more even distribution of mission times. The deep-reinforcement learning approach has a small number of outliers with high mission time, often caused by the agent getting stuck in a certain trajectory in the dark. As shown in Figure 4, the light gradient is limited far away from the source. The presence of noise makes it extremely hard for the agent to retrieve source information, at a great distance. We often observed a more direct path in the last $2.5\,\mathrm{m}$, compared to the first $2\,\mathrm{m}$ towards the source.

3) Failure cases: On top of our extensive mission time and success rate analysis, we also provide an analysis of failure cases. We describe three distinct failure cases: 1) dense obstacles in a certain configuration, 2) shadows, and 3) a source centered in the room.

First, the agent is trained with either zero or one obstacle in each episode. Therefore, when the agent encounters a dense wall of obstacles, it avoids the entire area. The agent has never encountered such a scenario in simulation, and thus rarely explores narrow passages between obstacles.

Second, shadows are not modeled in simulation and hence lead to unpredictable results. Occasionally, we saw the agent 'turning back' when moving into a shadow. By term s_1 in Equation 4, the agent sees a steep negative gradient when moving into a shadow. Therefore, it turns around, as this would be an effective approach when gradients are largely negative.

Third, a source in the center of the room significantly deteriorates the success rate and mission time. In simulation, we spawned the agent in the center of the room, while randomizing the environment, with the source at least $1\,\mathrm{m}$ from the center. For this reason, the model seemed to adopt a tendency to keep a certain distance to the center and rarely explore the center of the room. With 'LOW OBS', we saw a success rate of 50 %, while we did not see success at all without obstacles. The agent has learned that the source never presents itself in the center (as in simulation) and hence does not explore that region.

These failure cases can be resolved by altering the simulation environment. For future work, we propose randomization of the agent's initial position as well, while training and testing in larger and more complex environments. Additionally, modeling shadows and curriculum learning with more obstacles would benefit the algorithm.

E. Endurance and power

Finally, we consider endurance as a performance metric for our solution. By performing source seeking on the CrazyFlie, we add weight and CPU cycles. We determine endurance in hover and compare a stock hovering CrazyFlie with our source seeking solution. We swap the multiranger deck with light sensor for the battery holder, lowering the weight from $33.4\,\mathrm{g}$ to $31.7\,\mathrm{g}$ ($-1.7\,\mathrm{g}$). The endurance observed with the stock CrazyFlie is 7:06.3, while our solution hovers for 6:30.8, reducing endurance by $35.5\,\mathrm{s}$. With a battery capacity of 0.925 Wh, the average power consumption increased from $7.81\,\mathrm{W}$ to $8.52\,\mathrm{W}$(+$0.71\,\mathrm{W}$). It is expected that the vast majority of the extra power consumption comes from the extra weight, as the maximum consumption of the Cortex-M4 is $0.14\,\mathrm{W}$.

To put these numbers into perspective, we compare them to a CrazyFlie with a camera and additional compute [1]. While specialized hardware has the potential to support bigger networks, their extra weight and power consumption have a more significant impact on endurance. As shown in [1], endurance is reduced by $100\,\mathrm{s}$ when adding the PULP-Shield, almost three times more than in our experiments. This shows another advantage of our configuration for source seeking.

V. Discussion

In this work, we presented a fully autonomous source seeking nano drone, based on a deep reinforcement learning policy trained in simulation. Reflecting on our work, we realize that a higher fidelity simulation would likely improve results. For instance, modeling of shadows and reflections would provide the agent with additional cues to identify the source position. Better modeling of aerodynamics and vehicle dynamics would be useful too, as most of the crashes were caused by prolonged rotation and drift. The model used in our simulation is a general-purpose UAV model, not necessarily representative for the Crazyflie's dynamics.

From an algorithm point of view, we believe it's likely that a better policy exists to fit into this form factor. By doing a neural architecture search (NAS), one can identify the single most optimal deep reinforcement learning policy. However, as training a model takes several days, the computational cost

of such an effort would be tremendous. A more direct method for improvement may be further optimization by techniques such as quantization, allowing to fit a larger size network on the CrazyFlie.

VI. Conclusion and Future Work

We show that deep reinforcement learning can be used to enable source seeking applications on microcontroller-based aerial robots without the need for additional compute. Even though specialized hardware can accelerate AI applications, manufacturing, and deploying them is costly and time-consuming. Instead, we have shown that we can enable deep-learning applications in size-, weight- and power-constrained robots. We believe our methodology, which involves careful application modeling and algorithm design, is sufficiently general that it can be used to perform not only light seeking but also other forms of source seeking or entirely different applications. We anticipate future work to build upon our work, extending it to those other application areas as well and improve on our algorithm design. From algorithm point of view, we anticipate future work to present novel traditional and AI-based algorithm for the task. From an application point of view, we anticipate future work to explore different sources and source seeking in a multi-agent setting.

ACKNOWLEDGMENTS

We thank Robert Wood, Jim MacArthur, Hassan Khawaja and Sharad Chitlangia (from Harvard) for their help in enabling this project. Rob's lab enabled the drone flight tests, Jim aided with the custom light sensor board, Sharad helped with debugging the deep-reinforcement learning pipeline and Hassan helped with the experimental setup. The research was funded by NSF award #1724197, whichhelped us procure research equipment and fund the students.

References

[1] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini, "A 64-mw dnn-based visual navigation engine for autonomous nano-drones," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8357–8371, Oct 2019.

[2] D. Palossi, F. Conti, and L. Benini, "An open source and open hardware deep learning-powered visual navigation engine for autonomous nano-uavs," in *15th International Conference on Distributed Computing in Sensor Systems, DCOSS 2019, Santorini, Greece, May 29-31, 2019*, 2019, pp. 604–611. [Online]. Available: https://doi.org/10.1109/DCOSS.2019.00111

[3] "Why are insects attracted to light?" https://www.forbes.com/sites/quora/2018/12/19/why-are-insects-attracted-to-light/59563b2e5f44, accessed: 9/5/2019.

[4] S. Krishnan, B. Boroujerdian, W. Fu, A. Faust, and V. J. Reddi, "Air learning: An AI research platform for algorithm-hardware benchmarking of autonomous aerial robots," *CoRR*, vol. abs/1906.00421, 2019. [Online]. Available: http://arxiv.org/abs/1906.00421

[5] A. Faust, O. Ramirez, M. Fiser, K. Oslund, A. Francis, J. Davidson, and L. Tapia, "PRM-RL: long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," *CoRR*, vol. abs/1710.03937, 2017. [Online]. Available: http://arxiv.org/abs/1710.03937

[6] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation," *CoRR*, vol. abs/1806.10293, 2018. [Online]. Available: http://arxiv.org/abs/1806.10293

[7] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992. [Online]. Available: https://doi.org/10.1007/BF00992698

[8] A. Faust, A. Francis, and D. Mehta, "Evolving rewards to automate reinforcement learning," *CoRR*, vol. abs/1905.07628, 2019. [Online]. Available: http://arxiv.org/abs/1905.07628

[9] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," *CoRR*, vol. abs/1812.05905, 2018. [Online]. Available: http://arxiv.org/abs/1812.05905

[10] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *CoRR*, vol. abs/1707.05110, 2017. [Online]. Available: http://arxiv.org/abs/1707.05110

[11] W. Koch, R. Mancuso, R. West, and A. Bestavros, "Reinforcement learning for uav attitude control," *ACM Trans. Cyber-Phys. Syst.*, vol. 3, no. 2, pp. 22:1–22:21, Feb. 2019. [Online]. Available: http://doi.acm.org/10.1145/3301273

[12] A. Molchanov, T. Chen, W. Hönig, J. A. Preiss, N. Ayanian, and G. S. Sukhatme, "Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors," *CoRR*, vol. abs/1903.04628, 2019. [Online]. Available: http://arxiv.org/abs/1903.04628

[13] N. O. Lambert, D. S. Drew, J. Yaconelli, R. Calandra, S. Levine, and K. S. J. Pister, "Low level control of a quadrotor with deep model-based reinforcement learning," *CoRR*, vol. abs/1901.03737, 2019. [Online]. Available: http://arxiv.org/abs/1901.03737

[14] K. Kang, S. Belkhale, G. Kahn, P. Abbeel, and S. Levine, "Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight," *CoRR*, vol. abs/1902.03701, 2019. [Online]. Available: http://arxiv.org/abs/1902.03701

[15] S. Dini and M. A. Serrano, "Combining q-learning with artificial neural networks in an adaptive light seeking robot," 2012.

[16] X. xing Chen and J. Huang, "Odor source localization algorithms on mobile robots: A review and future outlook," *Robotics and Autonomous Systems*, vol. 112, pp. 123 – 136, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889018303014

[17] R. Zou, V. Kalivarapu, E. Winer, J. Oliver, and S. Bhattacharya, "Particle swarm optimization-based source seeking," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 3, pp. 865–875, July 2015.

[18] L. Parker, J. Butterworth, and S. Luo, "Fly safe: Aerial swarm robotics using force field particle swarm optimisation," *CoRR*, vol. abs/1907.07647, 2019. [Online]. Available: http://arxiv.org/abs/1907.07647

[19] W. Giernacki, M. Skwierczyski, W. Witwicki, P. Wroski, and P. Kozierski, "Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering," in *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, Aug 2017, pp. 37–42.

[20] N. J. Sanket, C. D. Singh, K. Ganguly, C. Fermüller, and Y. Aloimonos, "Gapflyt: Active vision based minimalist structure-less gap detection for quadrotor flight," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 2799–2806, Oct 2018.

[21] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," *CoRR*, vol. abs/1705.05065, 2017. [Online]. Available: http://arxiv.org/abs/1705.05065

[22] G. de Croon, L. O'Connor, C. Nicol, and D. Izzo, "Evolutionary robotics approach to odor source localization," *Neurocomputing*, vol. 121, pp. 481 – 497, 2013, advances in Artificial Neural Networks and Machine Learning. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231213005869

[23] J. Palacin, T. Palleja, I. Valganon, R. Pernia, and J. Roca, "Measuring coverage performances of a floor cleaning mobile robot using a vision system," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, April 2005, pp. 4236–4241.

[24] P. Anderson, A. X. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, and A. R. Zamir, "On evaluation of embodied navigation agents," *CoRR*, vol. abs/1807.06757, 2018. [Online]. Available: http://arxiv.org/abs/1807.06757