Learning to Seek: Deep Reinforcement Learning for Phototaxis of a Nano Drone in an Obstacle Field

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. While visual servoing of nano drones can achieve complex tasks, state of the art solutions have significant impact on endurance and cost. The primary goal of our work is to demonstrate phototaxis in an obstacle field, by adding only a lightweight and low-cost light sensor to a nano drone. We deploy a deep reinforcement learning model, capable of direct paths even with noisy sensor readings. 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 onboard the nano drone. We verify our approach with simulation and in-field testing on a Bitcraze CrazyFlie, achieving 94% success rate in cluttered and randomized test environments. 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 random walker 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. State of the art visual servoing approaches [1, 2] have proven to be capable of complex tasks, but also decrease endurance and increase cost.

Several complex tasks can be completed without a camera, as bug algorithms have demonstrated [3, 4]. An understudied application is source seeking on nano drones, where the objective is to navigate safely to a source. 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 during a potential collision.

We focus on light seeking, as it is a fundamental mechanism in nature, where an organism responds either positively or negatively to a light source, moving towards 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 [5]. 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.

[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.

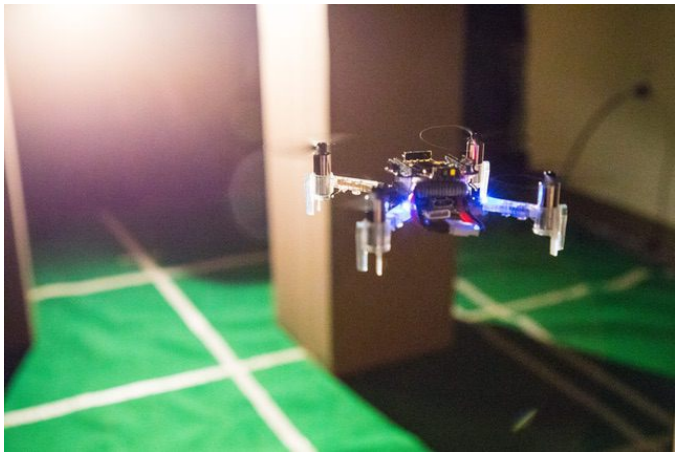


Fig. 1. CrazyFlie nano drone running a deep reinforcement learning policy fully *onboard*. It performs the computation online using a low-power Cortex-M4 microcontroller. It uses a light sensor to locate the source while avoiding obstacles with a multiranger and an optical flow sensor for flight stability.

In this paper, we present a light-seeking nano drone, performing robust obstacle avoidance using an ultra-lightweight ($0.2\,\mathrm{g}$) light sensor. While a human-designed method like the I-Bug [6] and random-walker [7] algorithms may be feasible solutions, we implement and train a deep reinforcement learning agent in simulation. Deep RL has the potential to learn how to deal with noise [8] and implicit information, like shadows. To the best of our knowledge, we introduce the first deep reinforcement learning (deep RL) based source-seeking nano drone that is fully autonomous and capable of avoiding obstacles. 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.

When designing a fully autonomous and deep RL based nano drone, the fundamental challenges fall into three dis-

tinct, but not completely independent, categories: (1) application modeling, (2) system design and (3) 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. System design means the design of lightweight and low-cost light sensor, but also the implementation of a lightweight custom inference library. Algorithm design consists of designing a neural network model and developing a deep RL based policy that successfully navigates the drone to the source while avoiding obstacles.

For application modeling, we extend the Air Learning simulation environment for training the drone to find the light source [9]. Air Learning is an AI-research platform that models a task, such as source seeking, in a randomly generated environment with 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 system design, we design an ultra-lightweight light sensor to minimize weight and maximize endurance. For inference of the deep RL policy, we implement a lightweight C-library onboard the CrazyFlie and arrive at a $100\,\mathrm{Hz}$ inference frequency, leading to robust obstacle avoidance.

For algorithm design, we use a deep reinforcement learning based source seeking solution, to use the limited and noisy inputs to find an efficient path to the source. Deep RL has shown to perform well in the presence of sensor noise [8], while capable of finding high-performance solutions to complex tasks [10]. We train a Deep Q-Network (DQN) [11] to locate the source. We feed two source terms, $s1$ and $s2$, constructed from source strength c and its low-pass version c_f. These features are designed to maximize performance within the tiny neural network that we are able to fit on the nano drone.

In summary, we make the following contributions:

- We introduce the first deep reinforcement learning algorithm capable of light seeking and obstacle avoidance. By designing relevant source features, the algorithm shows robust source seeking and obstacle avoidance.
- We implement the deep reinforcement learning policy onboard a nano drone, and, to the best of our knowledge, present the first source-seeking nano drone. We provide a series of flight results, demonstrating robust behavior by end-to-end navigation by deep reinforcement learning.

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, analyze the agent's behavior, 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 [12]. Not only have robotic manipulation [10] and locomotion [13] 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 [14, 15, 16] and was recently used to perform model-based reinforcement learning (MBRL) with a CrazyFlie [17]. High-level control using deep reinforcement learning for obstacle avoidance has been shown with different sets of sensory input [18, 9]. Although light seeking has been demonstrated before using Q-learning [19], *we present a deep reinforcement learning-based model for end-to-end navigation of a source seeking nano drone, including obstacle avoidance*.

Traditional source seeking algorithms can be divided into four categories [20]: 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. The I-Bug algorithm [6], for example, requires the level sets to be concentric images of simple closed curves, which is not true in many source seeking applications.

Previous bio-inspired algorithms have yielded promising results, but often 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 design a bio-inspired deep RL approach for source seeking, and demonstrate robust behavior on a nano drone.*

In a multi-agent setup, particle swarming [21, 22] has shown to be successful in simulation. However, just like the other gradient-based methods, it is likely to lack performance in flight tests due to sensor noise. Finally, probabilistic and map-based algorithms are more flexible but require high computational cost and accurate sensory information. In contrast to traditional methods, deep reinforcement learning can learn to deal with noisy inputs [8] 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.*

III. Method

We start by explaining the characteristics of our nano drone (Section III-A), including its microcontroller unit (MCU) and

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.

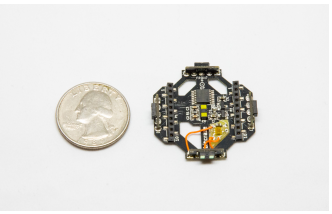


Fig. 2. Close-up of a BitCraze multiranger deck, fused with our custom TSL2591 light sensor board.

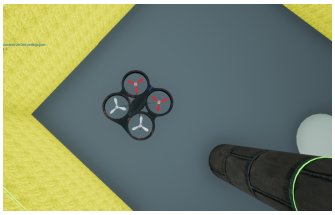


Fig. 3. Air Learning simulation with obstacle(s) and a light source. The picture is illuminated for clarity.

light sensor. 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. 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 [23]. 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 [24]. 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.

State of the art approaches using a camera and the GAP8 MCU [1] can solve complex tasks, thanks to its increased computational power, these solutions significantly affect endurance endurance and cost. In this work, we exploit the stock hardware when developing learning-based algorithms.

In contrast, 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 runtime has to be configured to meet the real-time constraints of both (1) the flight control stack for stability and (2) inference for obstacle avoidance. As the input to our network is low-dimensional, and the network small, we are able to run the network at 100 Hz.

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 hence 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 [9], which couples with Microsoft AirSim [25] 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` and rotating `left` or `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 in simulation, meaning the rate controller will command $\dot{\psi} = 54°/s$ or $v_x = 0.5\ m/s$ (body 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

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.

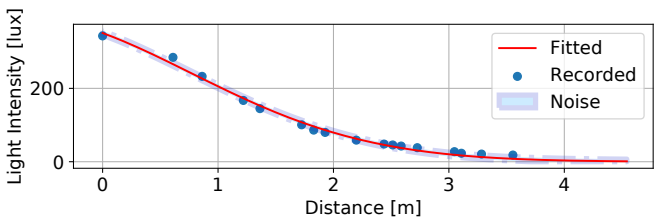


Fig. 4. Light intensity describing the function as used in training with 3σ (standard deviations) of the noise.

the light intensity in a two-dimensional grid with our light sensor. We have a 50 W LED 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 may be adopted to improve performance.

C. Policy Selection and Reward Shaping

We train and deploy a Deep-Q-Network (DQN) [11] algorithm on the CrazyFlie. Lower-level control is carried out by a set of PID controllers, while higher-level actions ($\dot{\psi}, v_x$, body frame) 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,

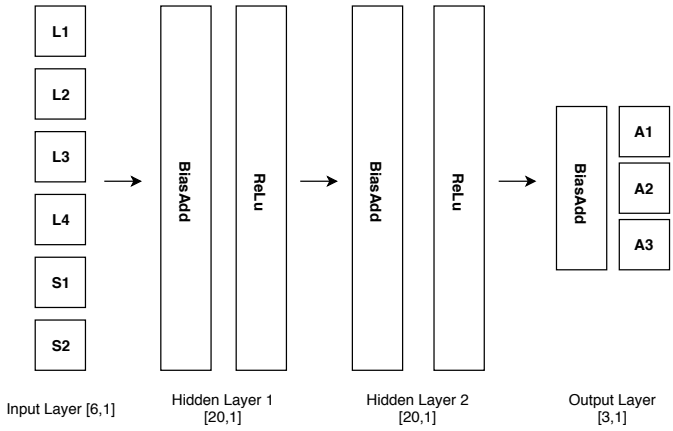


Fig. 5. Visualization of the general network architecture.

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).

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

We then add a low-pass filter and compute c_f.

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

Finally, inspired by [26], we compute terms s_1 and s_2.

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

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

Figure 13 shows all variables on a source-seeking nano drone. Term s_1 is effectively a normalized and low-pass version of the gradient of c (i.e., it is the light gradient over time). The low-pass filter has a high cutoff frequency, but is useful in filtering outliers. Normalization is crucial when far away from the source, when the gradient is small and finding the source is hard. Term s_2 is a transformation of c_f. If c_f is 0.5, and s_2 0, the agent is 1.5 m from the source (Figure 4). The signal therefore provides the agent with a proxy of distance to the source, which it can use to alter behavior. In general, we observed it flying `forward` more when it was close to the source.

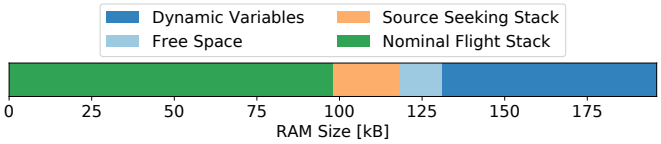


Fig. 6. RAM usage on the Bitcraze CrazyFlie, using a custom float inference stack. Total free space: 12.5 kB

We have trained policies with non-filtered gradient inputs, leading to poor results. As discussed in [20], 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.

The agent succeeds when it reaches within one meter of the light source in the simulation environment. To teach the nano drone to seek the light source, the following reward is computed at each step (instantaneous reward):

$$r = 1000 \cdot \alpha - 100 \cdot \beta - 20 \cdot \Delta D_s - 1 \tag{5}$$

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.

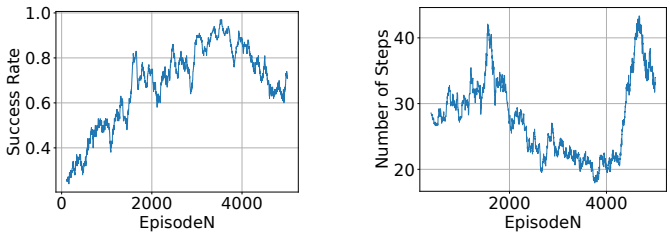
D. Deploying on the Nanodrone

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.

[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.



(a) Training success rates. (b) Steps to locate source.

Fig. 7. Quality metrics during training of the source seeking nano drone.

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), evaluate flight tests (Section IV-D) and analyze the agent's behavior (Section IV-E). Finally, we analyze endurance and power in Section IV-F.

A. Baseline Algorithm

To evaluate how well our deep reinforcement learning based approach performs, we compare our approach to more traditional solutions. The baseline should be able to work with our sensors and source. While I-Bug [6] uses similar sensory inputs, it requires the level sets to be concentric images of simple closed curves. In the presence of shadows and noise this requirement remains unsatisfied.

We implement a random walker [7] baseline algorithm, that performs obstacle avoidance without any specific information about the light source. The random walker algorithm moves in a straight line until it encounters an obstacle. As demonstrated in [27], this approach is effective in exploration and often used in autonomous cleaning robots.

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.

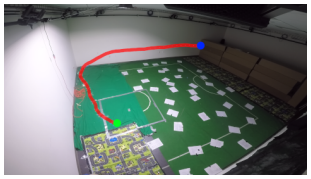
B. Training in Simulation

To evaluate the learning process, we present quality metrics (success rate and number of steps) during training. As shown in Figure 7, we train our policy up to convergence at around 3,600 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.

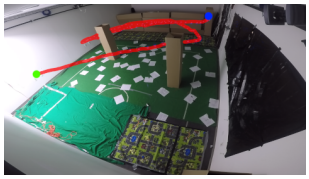
Model description	Success	# of Steps	Distance [m]	SPL
Our deep RL algorithm	96%	30.51	4.21	0.37
Random walker	84%	87.73	10.40	0.16
Random actions	30%	42.13	3.55	0.13

TABLE III
MODELS EVALUATED IN SIMULATION.

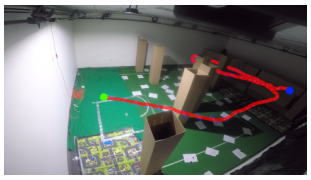
Fig. 8. Number of steps in simulation over 100 runs of each algorithm.



(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. Out of scope of simulation environments.

Fig. 10. 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.

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 are captured only when the agent succeeds. Additionally, we add the 'SPL' metric: Success weighted by (normalized inverse) Path Length [28]. 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 random walker baseline, and fully random actions in simulation. Table III and Figures 8, 9 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.

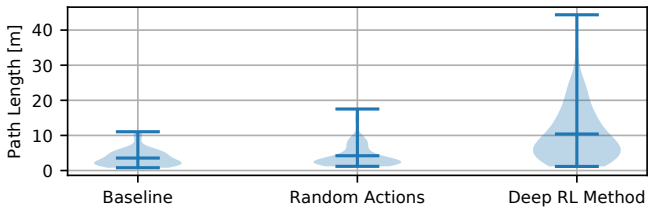


Fig. 9. Path length (i.e. traveled distance) in simulation over 100 runs of each algorithm.

D. Flight Tests

For flight tests, we use a room that is approximately $5\,\mathrm{m} \times 5\,\mathrm{m}$ in size (see Figure 10). 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 10 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 and $4.6\,\mathrm{m}$ from the source, resulting in an initial total source distance between $4.6\,\mathrm{m}$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 random walker

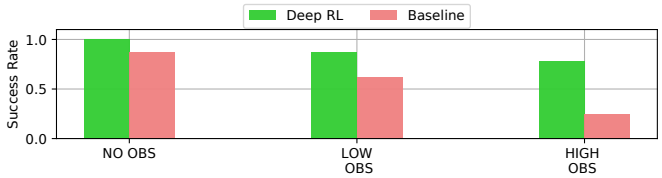


Fig. 11. Success Rate over 104 flight experiments, comparing our deep RL approach with the random walker baseline.

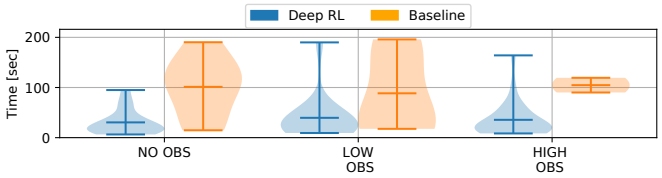


Fig. 12. Mission time in success over 104 flight experiments, comparing our deep RL approach with the random walker baseline.

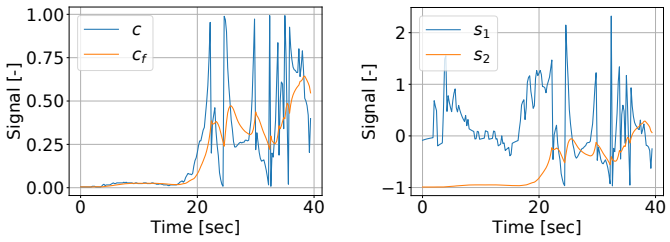
baseline model. As can be seen in Figure 11, our model beats the baseline in all three obstacle density groups. The baseline reaches a 75% success rate in a set of representative flight tests ('NO OBS' and 'LOW OBS'), compared to a 84% success rate in simulation.

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 important metric when evaluating the viability of an approach for real applications. Figure 12 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 [27], the random walker 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 13, 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 initial exploration.

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



(a) Normalized light measurements c and its low-pass version c_f.

(b) Policy inputs s_1 and s_2, as described in Equations 3 and 4.

Fig. 13. Source measurements on the nano drone while seeking the light source. The first 20 seconds the sensor provides little information, after which the source is found and high noise levels are observed.

First, the agent is trained in a sparse obstacle field, with only one obstacle. 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, 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. The agent has learned that the source never presents itself in the center (as in simulation) and hence usually 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. Behavior Analysis

To better understand the behavior of the nano drone, we record source measurements and network inputs during flight. A number of conclusions can be drawn from Figure 13:

- When far away from the source, extremely little information is present. In the first 20 seconds, almost no gradient is visible, forcing the agent to explore.
- The raw light readings are extremely noisy and unstable due to sensor noise, attitude changes and shadows.

- Once the agent 'sees' the source, it keeps traveling up the gradient and doesn't go back to the dark zone.
- The features work as imagined, s_1 is a normalized light gradient with a low-pass filter and s_2 is a transformation of c_f.

Figure 13 reveals the shortcomings of an ultra-lightweight and low-power light sensor. Considering how noisy the raw data is, we believe the custom design of our features was critical, especially when using such a tiny network.

The resolution and dimensionality of the sensor restrict more effective source seeking when far away from the source. The observed changes in source intensity are mostly noise, forcing the agent to explore. Similar to many bug algorithms [4], we observed wall-following during the exploration stage. It seems like the agent has learned to explore by following walls, so that it can get close enough to the source to 'see' it. An interesting direction for future work would be to further investigate the learnt behavior, and study the effects of training in different environments on behavior.

F. 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]. 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 that the state of the art methods for vision-based navigation on nano drones have large impact on endurance, and that different sensors are worth investigating.

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. By improving simulator fidelity, the advantages of our learning-based approach will be exacerbated, as it can, for example, learn to use shadows and turbulence to find a shorter path.

Additionally, 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 nano drones. Even though state of the art visual servoing strategies can achieve complex tasks, they have significant impact on endurance and cost. Instead, we have shown that we can enable complex tasks by end-to-end learning in size-, weight- and power-constrained robots. While traditional bug algorithms are mostly designed for ideal simulation environments, we trained a deep reinforcement learning policy that is robust to the noise present in the real world. We believe this methodology is useful in other real-world applications too, as robots learn to adapt to noise and other implicit information like shadows.

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.

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*, 6(5):8357–8371, Oct 2019. ISSN 2372-2541. doi: 10.1109/JIOT.2019.2917066.

[2] Daniele Palossi, Francesco Conti, and Luca 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*, pages 604–611, 2019. doi: 10.1109/DCOSS.2019.00111. URL https://doi.org/10.1109/DCOSS.2019.00111.

[3] K. N. McGuire, C. De Wagter, K. Tuyls, H. J. Kappen, and G. C. H. E. de Croon. Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment. *Science Robotics*, 4(35), 2019. doi: 10.1126/scirobotics.aaw9710. URL https://robotics.sciencemag.org/content/4/35/eaaw9710.

[4] K.N. McGuire, G.C.H.E. de Croon, and K. Tuyls. A comparative study of bug algorithms for robot navigation. *Robotics and Autonomous Systems*, 121:103261,

2019. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2019.103261. URL http://www.sciencedirect.com/science/article/pii/S0921889018306687.

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

[6] K. Taylor and S. M. LaValle. I-bug: An intensity-based bug algorithm. In *2009 IEEE International Conference on Robotics and Automation*, pages 3981–3986, May 2009. doi: 10.1109/ROBOT.2009.5152728.

[7] J. Evans. *Optimization Algorithms for Networks and Graphs*. Routledge, 2017.

[8] Aleksandra Faust, Oscar Ramirez, Marek Fiser, Kenneth Oslund, Anthony Francis, James Davidson, and Lydia Tapia. PRM-RL: long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. *CoRR*, abs/1710.03937, 2017. URL http://arxiv.org/abs/1710.03937.

[9] Srivatsan Krishnan, Behzad Boroujerdian, William Fu, Aleksandra Faust, and Vijay Janapa Reddi. Air learning: An AI research platform for algorithm-hardware benchmarking of autonomous aerial robots. *CoRR*, abs/1906.00421, 2019. URL http://arxiv.org/abs/1906.00421.

[10] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *CoRR*, abs/1806.10293, 2018. URL http://arxiv.org/abs/1806.10293.

[11] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992698. URL https://doi.org/10.1007/BF00992698.

[12] Aleksandra Faust, Anthony Francis, and Dar Mehta. Evolving rewards to automate reinforcement learning. *CoRR*, abs/1905.07628, 2019. URL http://arxiv.org/abs/1905.07628.

[13] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905, 2018. URL http://arxiv.org/abs/1812.05905.

[14] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *CoRR*, abs/1707.05110, 2017. URL http://arxiv.org/abs/1707.05110.

[15] William Koch, Renato Mancuso, Richard West, and Azer Bestavros. Reinforcement learning for uav attitude control. *ACM Trans. Cyber-Phys. Syst.*, 3(2):22:1–22:21, February 2019. ISSN 2378-962X. doi: 10.1145/3301273. URL http://doi.acm.org/10.1145/3301273.

[16] Artem Molchanov, Tao Chen, Wolfgang Hönig, James A. Preiss, Nora Ayanian, and Gaurav S. Sukhatme. Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors. *CoRR*, abs/1903.04628, 2019. URL http://arxiv.org/abs/1903.04628.

[17] Nathan O. Lambert, Daniel S. Drew, Joseph Yaconelli, Roberto Calandra, Sergey Levine, and Kristofer S. J. Pister. Low level control of a quadrotor with deep model-based reinforcement learning. *CoRR*, abs/1901.03737, 2019. URL http://arxiv.org/abs/1901.03737.

[18] Katie Kang, Suneel Belkhale, Gregory Kahn, Pieter Abbeel, and Sergey Levine. Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight. *CoRR*, abs/1902.03701, 2019. URL http://arxiv.org/abs/1902.03701.

[19] Steve Dini and Mark Anthony Serrano. Combining q-learning with artificial neural networks in an adaptive light seeking robot, 2012.

[20] Xin xing Chen and Jian Huang. Odor source localization algorithms on mobile robots: A review and future outlook. *Robotics and Autonomous Systems*, 112:123 – 136, 2019. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2018.11.014. URL http://www.sciencedirect.com/science/article/pii/S0921889018303014.

[21] R. Zou, V. Kalivarapu, E. Winer, J. Oliver, and S. Bhattacharya. Particle swarm optimization-based source seeking. *IEEE Transactions on Automation Science and Engineering*, 12(3):865–875, July 2015. ISSN 1545-5955. doi: 10.1109/TASE.2015.2441746.

[22] Lauren Parker, James Butterworth, and Shan Luo. Fly safe: Aerial swarm robotics using force field particle swarm optimisation. *CoRR*, abs/1907.07647, 2019. URL http://arxiv.org/abs/1907.07647.

[23] 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)*, pages 37–42, Aug 2017. doi: 10.1109/MMAR.2017.8046794.

[24] 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*, 3(4):2799–2806, Oct 2018. doi: 10.1109/LRA.2018.2843445.

[25] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. *CoRR*, abs/1705.05065, 2017. URL http://arxiv.org/abs/1705.05065.

[26] G.C.H.E. de Croon, L.M. O'Connor, C. Nicol, and D. Izzo. Evolutionary robotics approach to odor source localization. *Neurocomputing*, 121:481 – 497, 2013. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2013.05.028. URL http://www.sciencedirect.com/science/article/pii/S0925231213005869. Advances in Artificial Neural Networks and Machine Learning.

[27] 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*, pages 4236–4241, April 2005. doi: 10.1109/ROBOT.2005.1570771.

[28] Peter Anderson, Angel X. Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, and Amir Roshan Zamir. On evaluation of embodied navigation agents. *CoRR*, abs/1807.06757, 2018. URL http://arxiv.org/abs/1807.06757.