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

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*— Fully autonomous navigation using nano drones has numerous application in the real world, ranging from search and rescue to source seeking. Nano drones are well-suited for source seeking because of their agility, low price, and ubiquitous character. Unfortunately, their constrained form factor limits flight time, sensor payload, and compute capability. These challenges are a crucial limitation for the use of source-seeking nano drones in GPS-denied and highly cluttered environments. Hereby, we introduce a fully autonomous deep reinforcement learning-based light-seeking nano drone. The 33-gram nano drone performs all computation on-board the ultra-low-power microcontroller (MCU). We present the method for efficiently training, converting, and utilizing deep reinforcement learning policies. Our training methodology and novel quantization scheme allow fitting the trained policy in 3 kB of memory. The quantization scheme uses representative input data and input scaling to arrive at a full 8-bit model. Finally, we evaluate the approach in simulation and flight tests using a Bitcraze CrazyFlie, achieving 80% success rate on average in a highly cluttered and randomized test environment. Even more, the drone finds the light source in 29% fewer steps compared to a baseline simulation (obstacle avoidance without source information). To our knowledge, this is the first deep reinforcement learning method that enables source seeking within a highly constrained nano drone demonstrating robust flight behavior. Our general methodology is suitable for any (source seeking) highly constrained platform using deep reinforcement learning. Code & video: `https://github.com/harvard-edge/source-seeking`**

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 AI-hardware [1], inference of neural networks on nano drones has been achieved.

In this paper, we focus on nano drone applications for source seeking where the objective is to navigate safely and locate a source (e.g., gas, light, radiation). A robot capable of locating a gas leak, an exit in a collapsed building or a radiation source is useful in a wide range of applications. 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 (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 board (Figure 3) to track light intensity, a laser-based multiranger to detect obstacles, a 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.

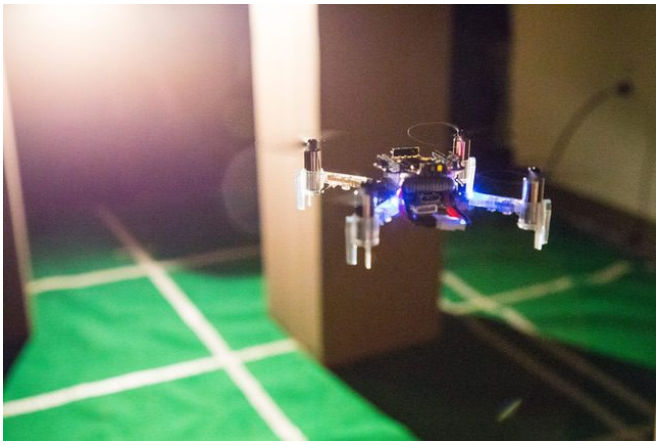

Fig. 1. CrazyFlie nano drone running a deep reinforcement learning policy fully *onboard*. It performs the computation online using a low-power Cortex-M4 microcontroller. It uses a light sensor to locate the source while avoiding obstacles with a multiranger and an optical flow sensor for flight stability.

When designing a fully autonomous and Deep-RL based nano drone, the fundamental challenges fall into three distinct, but not completely independent, categories: (1) application modeling; (2) algorithm design; and (3) system optimization. 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 picking a neural network model and developing a deep reinforcement learning based policy that successfully navigates the drone to the source while avoiding obstacles. System optimization involves fitting the trained network policy onto a severely memory and performance constrained onboard compute platform.

For application modeling, we extend the Air Learning simulation environment for training the drone to find the light source [2]. 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 obsta-

[1]Harvard University, [2] Robotics at Google, [3]Delft University of Technology, [4] The University of Texas at Austin - bduisterhof@g.harvard.edu

cles. 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 [3], while capable of finding high-performance solutions to complex tasks [4]. We train a Deep Q-Network (DQN) [5] 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 intensity sensor. The last layer of the neural network policy maps to the action space. For source seeking application, the action space includes three actions (`rotate left`, `right`, or `move forward`). We study two policies, one with a history of past inputs and one without any history. A comparison between the policies allows us to determine whether action history helps in successfully (or more effectively) locating the source.

We evaluate the quantized neural network model in 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 80 % 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, and system optimization) can be readily extended to other (source seeking) applications as well.

II. Related Work

Prior work in source seeking as it pertains to our work spans across 1) application; 2) algorithm; and 3) system design. As an *application*, source seeking was introduced by Braitenberg et. al [6]. It has many different uses. Gas seeking was explored in prior work [7], [8], as gas seeking can be used to localize and finally neutralize a dangerous leak. For similar reasons, radiation detection [9], [10] is useful as it can go where humans are at risk. Finally, light seeking was previously explored [11], [6] using a variety of algorithms. Work in obstacle avoidance while seeking a source is sparse [12], [13], and full autonomy is rarely achieved. *Our contribution is fully autonomous light-seeking while avoiding obstacles on a nano drone.*

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 not only more robust and accurate but also more effortless application [14]. Not only have robotic manipulation [4] and locomotion [15] 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 [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], [2]. Even though light-seeking has been demonstrated before using Q-learning [11], *we present a deep reinforcement learning-based model running onboard a nano drone with no specialized AI hardware needs*.

Source seeking algorithms can be divided into four categories [19]: 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. The previous bio-inspired algorithms have yielded promising results for single-agent scenarios, but failed to impress with a multi-agent set-up. In a multi-agent scenario, particle swarming [13], [20] 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 [3] 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 this opportunity to demonstrate a novel application of deep reinforcement learning for robust source seeking.*

From a *systems* perspective, quantization is an essential part of our solution and is novel to deep reinforcement learning. Previous work has shown compression of deep neural networks (DNNs) via techniques like pruning and quantization [21]. There is extremely minimal work on quantization for deep reinforcement learning [2], even though it has the potential to greatly speed-up inference of neural network-based policies trained using deep reinforcement learning [22], [23]. *We introduce a novel quantization strategy, specifically for the task of source seeking with deep reinforcement learning.*

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) to train deep reinforcement learning policies (Section III-C). Finally, we go over how we quantize the model to ensure it fits within the nano drone resource constraints (Section III-D) and explain how we deploy it onto the drone (Sections III-

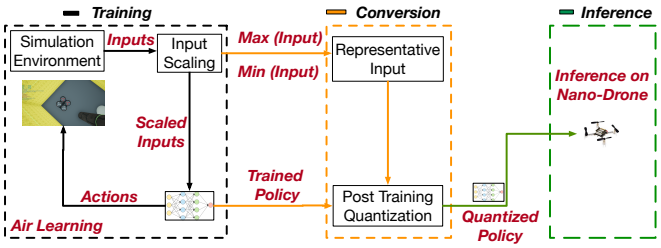

Fig. 2. Training, conversion (i.e., quantization) and inference methodology for source seeking on a nano drone using deep reinforcement learning.

E,III-F). Figure 2 shows an overview of how these tasks are distributed across (1) training; (2) conversion; and (3) inference deployment.

A. Vehicle Configuration

We use the BitCraze CrazyFlie 2.1 as our research platform. It is an agile, yet limited platform, from both a compute and physical point of view. The physical characteristics of the CrazyFlie are captured in Table I, emphasizing its small size, weight, and power. Similarly, the CrazyFlie's computational capabilities are shown in Table II.

To put the CrazyFlie into proper nano drone perspective, we compare the CrazyFlie's compute capabilities and its physical characteristics against a more conventional Parrot Bebop 2 drone that is frequently used for deep learning-based demonstrations while carrying an Nvidia Jetson TX2 [24]. In both form factor (Table I) and compute capabilities (Table II), the CrazyFlie is severely constrained. Alternatively, some nano drone demonstrations use custom-designed AI-hardware specialized to perform efficient inference [1]. Unfortunately, the vast majority of all chips on existing nano drones are low-power general-purpose CPU's. Hence, it is of greater benefit to utilize these chips for learning-based algorithms to unlock new applications.

Due to the limited capabilities of the CrazyFlie, we are limited on the sensor payload. So we equip the CrazyFlie with three sensors for light-seeking: (1) a laser multiranger to measure the distance in the front/back/left/right directions, which we use to avoid obstacles; (2) a flow deck to track motion on the z-axis and $x-y$ plane to ensure stable flight; and (3) a light sensor to help locate the source by measuring the illuminance at a specific point in time and space. The light sensor sits on a custom "deck" that we designed and integrated with the CrazyFlie firmware via a deck software driver. The light sensor faces upwards, as shown in Figure 3.

B. Simulation Environment

The first building block of our methodology is the simulation or the training environment. We use the Air Learning platform [2], which couples with Microsoft AirSim to provide a deep reinforcement learning back end.

Using Air Learning, we simulate a training environment that is an arena. The arena is "spawned" at `10x10` meters in size. 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

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
Flighttime	7 mins	25 mins
Size (WxHxD)	9.2x 9.2 cm	32.8 x 38.2 cm

TABLE I

CRAZYFLIE 2.1 VERSUS BEBOP2 DRONE PLATFORM SPECIFICATIONS.

Name	STM32F405	TX2
CPU	1-core @168MHz	6 cores @ 2 GHz+
GPU	None	256-core @1300 MHz
RAM	196 kB	8 GB
Storage	1 MB	32 GB
Power	0.14 W (max)	7.5 W

TABLE II

CRAZYFLIE MICROCONTROLLER SYSTEM VERSUS AN NVIDIA TX2.

in random locations within the room. The agent must learn to traverse the room without running into obstacles or run out of (750) steps to find the light source. Figure 4 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 `0.3` seconds, meaning the rate controller will command $\dot{\psi} = 54°/s$ or $v_x = 0.5m/s$ (NED frame) for 0.3 seconds. The training hardware (an Intel 9900K CPU and RTX 2080 TI GPU) is capable of fast inference. However, training with these capabilities would not be comparable to what the CrazyFlie microcontroller unit is capable of processing in real-time. We experimentally determined that during flight the network's output actions can be updated, approximately, once every 0.3 seconds. Hence, during training, we take this processing constraint into account. Especially for timely obstacle avoidance, it is critical to teach the agent what latency to expect from the onboard processing hardware.

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`

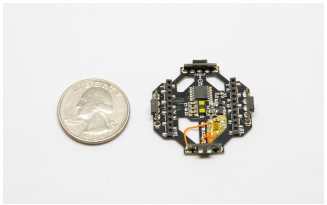

Fig. 3. Close-up of a BitCraze multiranger deck, fused with our custom TSL2591 light sensor board.

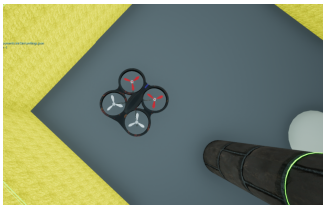

Fig. 4. Air Learning simulation with obstacle(s) and a light source. The picture is illuminated for clarity.

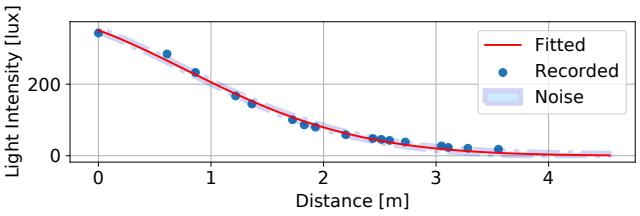

Fig. 5. Light intensity describing the function as used in training with 3σ (standard deviations) of the noise.

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

Once we capture this data, we 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 5. 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 0.993, 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 position, 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.

C. Policy Selection and Reward Shaping

We train and deploy a Deep-Q-Network (DQN) [5] 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. We consider two distinct feed-forward neural networks for the task. The distinction between them is the inclusion or exclusion of history states. We define a state as all four laser range readings and one light sensor reading (Figure 6). Both policies have two hidden layers of 20 nodes each and an output layer of length three nodes that map to the action space (move `forward`, turn `right` or `left`). The network contains a Softmax layer in training.

- **No-History Policy** The agent only has the current four laser rangers and light sensor values at its disposal. The challenge with this approach is that there is no way to find the source in a deterministic manner. If no history is apparent, determining the gradient is infeasible, presumably leading to worse mission performance.
- **History Policy** The agent has access to history. Hence, the agent has the potential to learn and explore more effectively based on its recent activity. For this policy, we use a history size of *n=4* (as depicted in Figure 6).
- **Other Policies** When considering a task that is highly dependent upon history information, recurrent and LSTM layers come to mind as a suitable policy architecture. However, there are two major problems with implementing these architectures. Firstly, as shown in Section III-F, resources are extremely limited, making it virtually impossible to implement the extra infrastructure for such layers (TFMicro stack). Secondly, the toolchain has limited support for such layers.

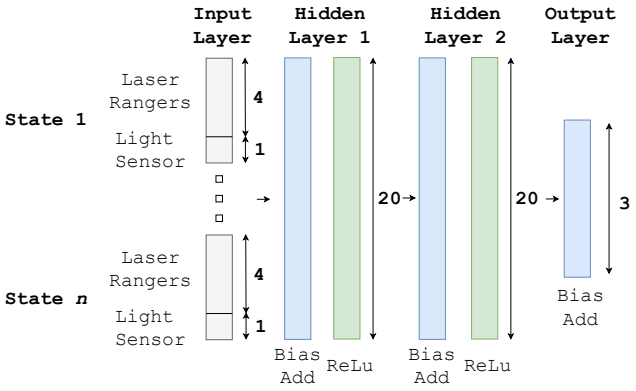

Fig. 6. Visualization of the general network architecture.

In either of the policies, the agent succeeds when it reaches within one meter of the light source in the simulation environment (`0.7 m` in the smaller flight test environment). To teach the nano drone to seek the light source, the following reward is computed at each step (instantaneous reward):

$$r = 1000 * \alpha - 100 * \beta - D_g - D_c * \delta - 1 \qquad (1)$$

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 750 steps.[1] Otherwise, β is '0', effectively penalizing the agent for hitting an obstacle or not finding the source in time. D_g is the distance to the source from the agent's current location, motivating the agent to move closer to the source. D_c is a penalty for moving away from the source when it almost reaches success. δ is a binary variable which is set to '1' if the agent is less than `2 m` away from the source, otherwise it is set to '0.'

D. Post-Training Quantization

The policy is trained using the `float32` or FP32 data format. Unfortunately, inference of a full `float32` network of this size on the CrazyFlie is infeasible due to the unacceptable processing delays. With a fully quantized network, we achieve a maximum inference rate of 4 Hz (Section III-F). Aggressively optimized `uint8` operations can be up to 4× faster than `float32` operations [22]. Based on our microbenchmarks running on the CrazyFlie's microcontroller, `float32` operations are 2.9× slower than `Int8` operations. In the context of running our neural network policy, this

[1]We set the maximum allowed steps in an episode as 750. This is to make sure the agent finds the source within some finite amount of steps.

would result in slowing down the inference rate to 1.3 Hz, making reliable obstacle avoidance infeasible.[2]

Therefore, quantization proves to be critical for success as it is directly tied to the performance of the agent. The quantization scheme is shown in Figure 2 and contains two essential innovations, compared to previous full 8-bit quantization [25]: 1) scaling of all the inputs to the same range for higher resolution and 2) usage of representative inputs to determine the min and max ranges inside the network. We discuss both of these steps independently now.

When feeding inputs into a quantized network, we need to determine a scaling range. An `uint8` can contain values from 0 to 255 while the `float` values do not coincide with this range. To convert the original `float` values to `uint8`, we must generate a conversion function that takes in a `float32` and generates a corresponding `uint8`.

However, we cannot use a naive conversion method that scales all the inputs into the network over the same range. For example, in our case, the laser ranger output values vary between 0 and 5 meters, whereas the light intensity output values vary between 0 and 300 lux. If a range of 0 to 300 is used for all variables, accuracy in the laser ranger readings is lost. Based on experimentation, we determine that it is more efficient to scale the individual inputs to the corresponding sensors maximum range for maximal resolution.

We then determine the minimum and maximum range for the tensors in the network. Initially, all tensors are in `float32` data type. While weight and activation functions have a known minimum and maximum value, the values of the tensors in the network depend on the input to the network. So, we feed a dataset of representative input values into the network and capture the minimum and maximum ranges of the tensors. The advantage is that now all multiplications can be carried out using the `uint8` data type, which leads to yet another significant boost in runtime performance.

E. Pre-Processing for the Quantized Model

To invoke the quantized model, at runtime, we pre-process the inputs before feeding them as input to the network. The laser rangers' values are available in `float32`. We convert these values from 0-5 meters in `float32` to 0-255 in `uint8`. The light sensor, however, requires a more complicated conversion for optimal performance. The light sensors outputs a `uint16` variable, which we have to convert to `uint8`. As the change in intensity between steps can be small, we need the highest possible resolution. For this reason, we measure the range of `uint16` values present in the room, determine the apparent range of readings, and finally scale our `uint8` within that range. In other words, let's say we measure values from 11,000 to 55,000 in our testing environment, we then scale `uint8` linearly between 11,000 and 55,000 ($0 = 11,000, 255 = 55,000$).

[2]A small `float32 Matrix-Vector` microbenchmark takes 113 milliseconds, whereas the same microbenchmark running on `uint8` data type takes 39.2 milliseconds. Hence, there is a 2.9× difference between a `float32` operation and an `uint8` operation running on the CrazyFlie microcontroller.

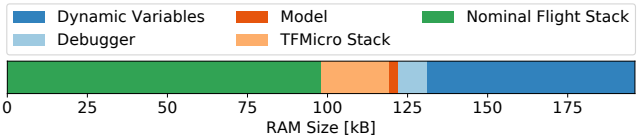

Fig. 7. RAM usage on the BitCraze CrazyFlie. Onboard memory is severely constrained and almost fully utilized with no room for anything else. Hence, the size of the model we can fit in is very small (∼3 kB). The "TFMicro Stack" in the legend is the TF-Lite stack for microcontrollers.

F. Deploying on the Nanodrone

We compile the quantized network and the TF-Lite interpreter for microcontrollers using the TF-Lite `C++` libraries. The code is fused with the rest of the Bitcraze Crazyflie software stack and "flashed" to the onboard microcontroller. The Bitcraze Crazyflie has 1 MB of flash storage. The nominal software stack occupies 194 kB of the available storage, while the TF-Lite stack takes up an additional 53 kB, of which the quantized models occupy only about 3 kB of storage. So the total flash storage used is 247 kB, which leaves an ample amount of free storage space (over 75%).

However, the runtime memory constraints are much more severe. We have only 9 kB of effective memory for running the model. RAM availability during execution is shown in Figure 7. We capture this data using the debugging interface. 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 nominal operation, the Bitcraze software stack uses 98 kB of RAM, leaving only 33 kB available for our purposes. At runtime, the TF-Lite interpreter, which facilitates model execution, consumes 24 kB of the 33 kB, thus leaving only 9 kB of RAM for the neural network model. Thanks to quantization, the model consumes only 3 kB of RAM. If it were an FP32 model, it would have taken 12 kB, which would not have fitted in the available memory.

During flight the greatest system constraint is access to the single-core CPU. TF-Lite has to share CPU time with higher priority processes (e.g. stabilization) to execute the model. Our model's response time (i.e., time for a single forward pass) was, on average, 46.4× longer than it would have been without interruption. In other words, the model spent only 2.16% percent of its 29.4 ms average response time executing on the CPU. The other 97.84% of the response time was spent waiting for a higher priority process to finish (e.g., stabilization) or waiting for sensitive data to be released by a lower priority process (e.g., multi-ranger). Additionally, there was significant overhead from context switching due to having only one (very slow) microcontroller for the CPU.

In summary, both CPU and RAM are operating at their absolute limit in the current configuration. Hence, running a `float32` based policy would have been infeasible from both memory capacity and performance perspective. Therefore, quantization is an enablement strategy for porting learned policy on a resource-constrained nano drone.

Model description	Success	# of Steps	Distance [m]
Baseline	59%	52.1	22.6
Float32 History	72%	73.3	33.2
No History Float32	35%	84.9	23.0
Uint8 History	42%	37.1	20.9

TABLE III

MODELS EVALUATED IN SIMULATION.

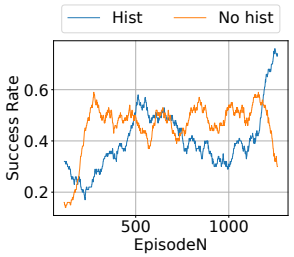

Fig. 8. Training success rates.

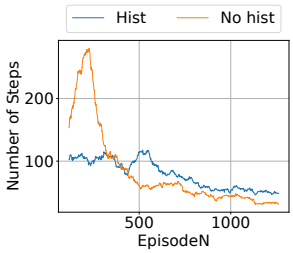

Fig. 9. Steps to locate source.

IV. RESULTS

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

A. Baseline Algorithm

To evaluate how well our deep reinforcement learning-based source seeking approach performs, we compare our approach to other 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 [26]. The algorithm simply rotates the drone at a random angle when the front laser ranger detects an obstacle under `2 m`.

The baseline algorithm's results are shown in Table III, along with the rest of our results. The success rate is 59%. From observations, we see that the front-ranger is unable to detect all of the obstacles and hereby fails to find the source.

B. Training in Simulation

To evaluate the learning process, we present quality metrics (success rate and the number of steps) during training. As the baseline algorithm obtains a 59% success rate (Table III), our goal is to obtain a model that outperforms the baseline. Hence, we terminate the training process when the success rate over the last 100 episodes exceeds 70%. Better success rates may be obtained from prolonged training, but we do not show that due to the computational effort required to do so (1200 episodes take at least 24 hours).

Figures 8 and 9 evaluate success rate and the number of steps taken to locate the source, respectively. Both policies, the policy with history and the policy without history, described in Section III-C, are compared. The history-based model meets the 70% success rate after about 1200 episodes, while the model without history reaches at most 60% success rate, after which it overtrains at about 1100 episodes.

Both models show a decreasing trend in the step count (Figure 9), e.g., the agent learns to find the source in fewer steps and thus time (each step is 0.3 seconds). The model without history information reaches a slightly lower number of steps. Likewise, both models show an initial increase in success rate, but they level-off at different success rates.

Judging from the training data alone, the model that uses history information performs substantially better in success rate and only slightly lacks in the number of steps.

C. Inference in Simulation

Training data provide limited information as the model is continuously changing. So, we evaluate both of the policies after training (as shown in Table III). We compare them 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. These metrics are irrelevant when the agent fails to find the source.

The **Float32 History** model is an FP32 model, which contains the current and past three states, features the highest success rate in the list. Its success rate surpasses the baseline, albeit not in the number of steps or the distance traveled.

The **No History Float32** model is also an FP32 model, which only uses the current state. It shows a dramatic decrease in the success rate and the number of steps compared to the model with history. We suspect this behavior is caused by the lack of history and therefore its inability to determine a gradient in source strength.

The **Uint8 History** model is an 8-bit quantized model with history. It outperforms the *No History Float32* in all metrics. However, we see a significant drop in the success rate, distance traveled, and the number of steps from the *Float32 History*. Quantized models are known to drop their "accuracy" with post-training quantization [21]. During simulation, we observe a delayed obstacle avoidance behavior in the quantized model, leading to the lower success rate.

However, the number of steps and traveled distance have improved significantly. As we show later, the drop in success rate is recovered in practice due to the simulation-to-reality gap. The real-life nano drone is much smaller than the simulated drone, and our testing environment is smaller than the simulation environment. So during simulation, the (larger) agent learns to be cautious, which in the real world helps the smaller drone succeed more often as it reacts well ahead of any obstacles. This strategy, to teach the agent to be more conservative in order to account for quantization, might be useful for various (source seeking) applications.

D. Flight Tests

For flight tests, we use the history-based quantized model in a room that is `5 x 6` meters in size (see Figures 10-13). We use a `50 Watt` light source attached to the roof, radiating a 120 ° beam onto the ground, as the light source. A flight-test is successful when the agent gets within `0.7 m` of the light source—the `1 m` threshold used in simulation was lowered to `0.7 m` as the room size was scaled down.

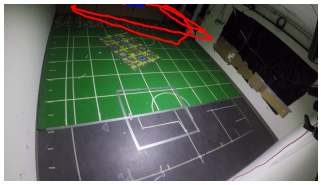

Fig. 10. Zero obstacles trajectory, stuck in dark zone.

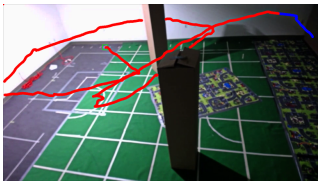

Fig. 11. Two obstacles trajectory, finds source and starts spiraling.

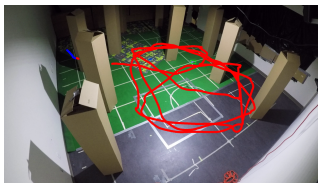

Fig. 12. Ten obstacles trajectory, finds source and starts spiraling.

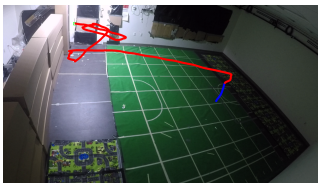

Fig. 13. Source position in corner, shows direct path to source.

We conduct a total of 105 flight tests in a variety of different scenarios, involving highly cluttered environments. Across all flight tests, we get an average success rate of *80%*. To better understand this success rate, we decompose the tests into four categories: 1) obstacle variation; 2) consistency; 3) drone position and 4) source position. Figure 14 summarizes mission time and success rate for all the experiments. Figures 10-13 show four distinct trajectories, where a blue trajectory highlights take-off and a green box the last detected drone position during that run.

Obstacle Variation In simulation, the agent is exposed to obstacles (typically between 0 or 1 per episode). We set out to investigate the behavior in much more cluttered environments during flight tests. We considered zero, two or ten obstacles, all with random initial drone positions (as shown by the "Obstacle Variation" bar cluster in Figure 14).

Surprisingly, we see a higher success rate and lower mission time in more cluttered environments. The superior performance in increasingly cluttered environments can be explained by the increased level of stochasticity in a more cluttered environment. With no obstacles, the agent occasionally gets stuck in a circle in the dark part of the room (Figure 10), taking longer to reach the source and hence increasing the probability of a collision. Placing obstacles breaks this pattern and thereby leads to superior performance. Our hypothesis is that this behavior may be caused by the inherently low variation in the light readings far from the source (Figure 5), implying the lack of sufficient source information for actively locating the source.

Repeat We see significant variations in mission metrics with varying obstacle settings, but a study on consistency/repeatability is of great interest. We initialize the drone at the same position during 10 consecutive runs and keep the environment constant ("Repeat" in Figure 14). The nano drone found the source in all 10 runs, but mission time varied between 7 and 74 seconds. The average mission time was 32 seconds, with a standard deviation of 31 seconds.

From this, we conclude that even while keeping all parameters constant, mission performance is susceptible to stochastic behavior. Noise in light readings and unpredictable aerodynamic behavior (e.g., drift, turbulence) likely cause variation in mission metrics. A new battery was used for each run, but performance between these batteries varies.

Drone Position In another experiment ("Drone Pos" in Figure 14), we study the influence of different initial drone position on mission metrics. We initialize the drone `1.6 m` ("Close") and `3.5 m` ("Far") away from the source in a series of 20 experiments with varying obstacle positions. The far initial location introduces a lower success rate and higher mission time compared to a close initial location. A larger initial distance means a larger expected distance traveled, and hence greater mission time and lower success rate (i.e., higher probability of collision).

Source Position Up to now, all experiments have been executed with the source in the same position and the drone in a random position, as shown in Figures 10–12. But it is also essential to verify behavior when the source position is varying. Therefore, we conduct a range of flight tests with the light source in the corner of the room, as shown in Figure 13.

The results are shown under "Source Pos" in Figure 14. The initial drone position is random, and no obstacles are present. The success rate for "Middle," which implies the light source is approximately in the middle of the room, is 60%. The success rate for "Corner," which means the light source is in the corner of the room, is 70%. While the success rates are similar, mission time, however, varies significantly. The drone takes 158 seconds versus 32 seconds for "Middle" versus "Corner" light sources, respectively.

With the source in the middle of the room ("Middle"), we see a spiraling trajectory. The drone moves closer to

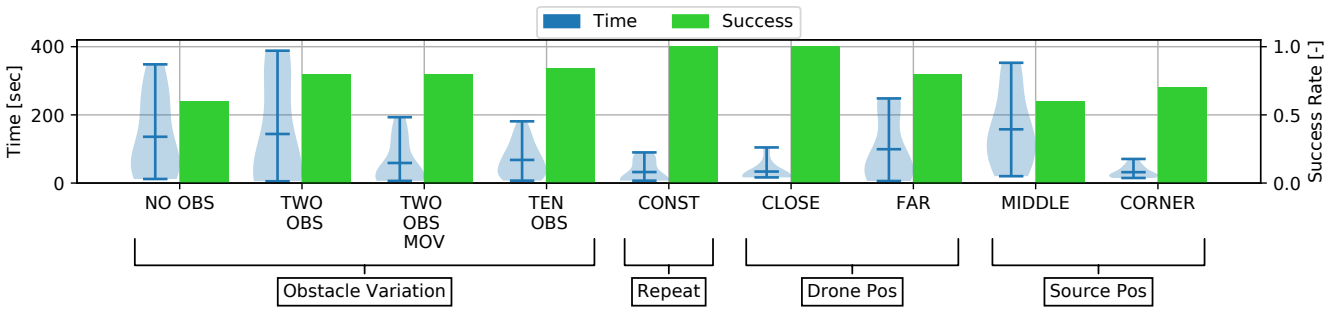

Fig. 14. Success rate and mission time over 105 flight tests. We use a "violin plot" to visualize mission time distribution and probability density.

the source as it spirals inward over time. In contrast, when the light source is at the corner of the room (as shown in Figure 13), the drone first takes a much more direct trajectory to the source at first and then it spirals around the source.

V. Conclusion

We demonstrate that deep reinforcement learning can be used to enable source seeking applications on microcontroller-based aerial robots without the need for specialized AI hardware. Even though specialized hardware can accelerate AI-applications, manufacturing and deploying them is costly and time-consuming. Instead, we have shown that by carefully analyzing the system's constraints and doing optimizations, we can enable deep-learning applications in size-, weight- and power-constrained robots. We believe our methodology, which involves careful application modeling, algorithm design, and system optimization, is sufficiently general that it can be used to perform not only light-seeking but also other forms of source seeking such as gas- and radiation-seeking in real-time on extremely limited hardware. We anticipate future work to build upon our work, extending it to those other application areas, as well as deal with multiple sources to identify the source that is most relevant.

ACKNOWLEDGMENTS

We would like to thank Pete Warden and Aakanksha Chowdhery (from Google), in addition to Hassan Khawaja, Sharad Chitlangia, Jim MacArthur and Robert Wood (from Harvard) for their help in enabling this project. Pete and Aakanksha helped us with early access to TensorFlow Lite for microcontrollers. Jim aided with the custom light sensor board, Sharad helped with debugging the TF-Lite code, Hassan helped with the experimental setup and Robert Wood enabled flight tests.

References

[1] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini, "Ultra low power deep-learning-powered autonomous nano drones," *CoRR*, vol. abs/1805.01831, 2018. [Online]. Available: http://arxiv.org/abs/1805.01831

[2] 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

[3] 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

[4] 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

[5] 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

[6] V. Braitenberg, *Vehicles: Experiments in Synthetic Psychology*, ser. Bradford Books. MIT Press, 1986. [Online]. Available: https://books.google.com/books?id=7KkUAT_q_sQC

[7] 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

[8] J. Burgus, V. Hernndez, A. J. Lilienthal, and S. Marco, "Smelling nano aerial vehicle for gas source localization and mapping," *Sensors*, vol. 19, no. 3, 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/3/478

[9] R. A. Cortez, H. G. Tanner, and R. Lumia, "Distributed robotic radiation mapping," in *Experimental Robotics*, O. Khatib, V. Kumar, and G. J. Pappas, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 147–156.

[10] K. Qian, A. Song, J. Bao, and H. Zhang, "Small teleoperated robot for nuclear radiation and chemical leak detection," *International Journal of Advanced Robotic Systems*, vol. 9, no. 3, p. 70, 2012. [Online]. Available: https://doi.org/10.5772/50720

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

[12] E. Ramrez-Llanos and S. Martnez, "Stochastic source seeking for mobile robots in obstacle environments via the spsa method," *IEEE Transactions on Automatic Control*, vol. 64, no. 4, pp. 1732–1739, April 2019.

[13] 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.

[14] 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

[15] 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

[16] 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

[17] 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

[18] 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

[19] 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

[20] 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

[21] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. [Online]. Available: http://arxiv.org/abs/1510.00149

[22] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," 2011.

[23] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International Conference on Machine Learning*, 2015, pp. 1737–1746.

[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*, vol. 3, no. 4, pp. 2799–2806, Oct 2018.

[25] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *CoRR*, vol. abs/1806.08342, 2018. [Online]. Available: http://arxiv.org/abs/1806.08342

[26] 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.