Interpretable Autonomous Flight Via Compact Visualizable Neural Circuit Policies

Paul Tylkin, Tsun-Hsuan Wang, Kyle Palko, Ross Allen, Ho Chit Siu, Daniel Wrafter, Tim Seyde, Alexander Amini, *Graduate Student Member, IEEE*, and Daniela Rus, *Fellow, IEEE*

***Abstract*—We learn interpretable end-to-end controllers based on Neural Circuit Policies (NCPs) to enable goal reaching and dynamic obstacle avoidance in flight domains. In addition to being able to learn high-quality control, NCP networks are designed with a small number of neurons. This property allows for the learned policies to be interpreted at the neuron level and interrogated, leading to more robust understanding of why the artificial agents make the decisions that they do. We also demonstrate transfer of the learned policy to physical flight hardware by deploying a small NCP (200 KB of memory) capable of real-time inference on a Raspberry Pi Zero controlling a DJI Tello drone. Designing interpretable artificial agents is crucial for building trustworthy AIs, both as fully autonomous systems and also for parallel autonomy, where humans and AIs work on collaboratively solving problems in the same environment.**

***Index Terms*—Reinforcement learning, machine learning for robot control, aerial systems: perception and autonomy.**

I. Introduction

ARTIFICIAL agents can learn to perform challenging tasks in complex domains with models that do not readily allow for understanding *why* they make the decisions that they do. However, there is a crucial reason for preferring agents whose learned behavior is *interpretable:* this allows for humans, or other AIs, to be able to better predict their behavior, and thus makes them inherently more trustworthy, especially in domains with unexpected circumstances.

In this paper, we show that by using Neural Circuit Policies (NCPs) [1]–[3], we can design high-performing agents whose learned policies are interpretable at the neuron level. A schematic diagram of an NCP is shown in Fig. 1. By *interpretable*, we mean being able to visualize the entire network's activity and perform a post-hoc analysis of the neuron activations in response to key environmental changes. Most learning-based control used in robotics today employs deep neural network (DNN) solutions that result in huge models that are difficult to inspect and audit. For control tasks, DNNs usually include tens or hundreds of thousands of neurons; these are black boxes with no simple way for users to audit their inner workings. Our objective with this paper is to explore how to create compact models for two flight control tasks and present a method to audit them visually. The two simulation domains we study are *Canyon Run*, in which a fixed-wing aircraft needs to navigate a narrow canyon of unknown geometry and hit dynamic waypoints, and *Drone Dodgeball*, in which a quadcopter needs to learn to balance remaining close to a waypoint and avoiding oncoming obstacles. As depicted in Fig. 2, we can learn compact NCP models ranging in size from 9 to 24 neurons to achieve the tasks. We then show that the learned policy from our quadcopter simulation environment can be transferred to physical flight hardware, with

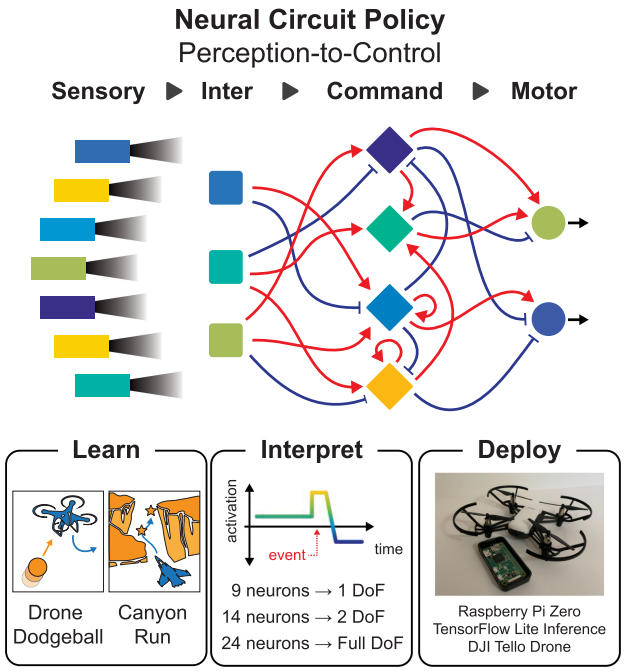

Fig. 1. Neural Circuit Policies allow for end-to-end learning (from perception to control) with a small number of neurons, which leads to policies that can be interpreted at the neuron level and safely deployed on real hardware.

Manuscript received September 9, 2021; accepted December 31, 2021. Date of publication January 27, 2022; date of current version February 8, 2022. This letter was recommended for publication by Associate Editor N. Bezzo and Editor J. Kober upon evaluation of the reviewers' comments. This work was supported by the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator under Cooperative Agreement Number FA8750-19-2-1000. *(Corresponding author: Paul Tylkin.)*

Paul Tylkin, Tsun-Hsuan Wang, Daniel Wrafter, Tim Seyde, Alexander Amini, and Daniela Rus are with the Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Laboratory, Cambridge, MA 02139 USA (e-mail: ptylkin@mit.edu; tsunw@mit.edu; dwrafter@mit.edu; tseyde@mit.edu; amini@mit.edu; rus@csail.mit.edu).

Kyle Palko is with the U.S. Air Force Artificial Intelligence Accelerator, Cambridge, MA 02139 USA (e-mail: kyle.palko.1@us.af.mil).

Ross Allen and Ho Chit Siu are with the Massachusetts Institute of Technology, Lincoln Laboratory, Lexington, MA 02421 USA (e-mail: ross.allen@ll.mit.edu; hoseasiu@mit.edu).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2022.3146555, provided by the authors.

Digital Object Identifier 10.1109/LRA.2022.3146555

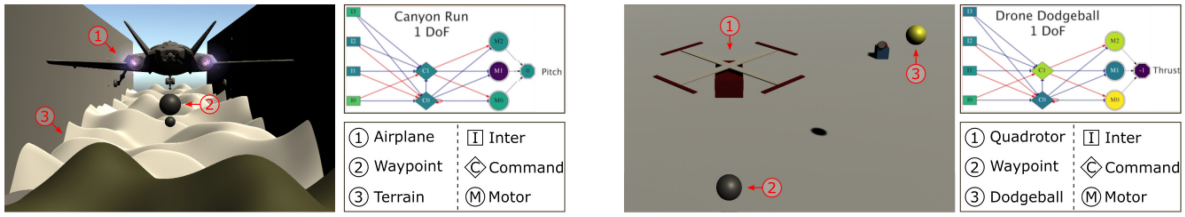

Fig. 2. The two environments, *Canyon Run* and *Drone Dodgeball*, for which we study learning interpretable policies are depicted above. The objective in Canyon Run is to collect waypoints while avoiding collision with the terrain or leaving the canyon. The objective in Drone Dodgeball is to remain as close to the waypoint as possible while avoiding the dynamic obstacles (dodgeballs). The top-right subfigures are NCP controllers that steer the aircraft, which are discussed in detail in Section III-A. The NCPs have three layers of neurons: inter, command, and motor neurons, and output actions.

all inference done on a Raspberry Pi Zero, an inexpensive and highly compute- and power-constrained single-board computer.

Contributions: Our main contributions are as follows:

1) End-to-end learning of interpretable policies in complex flight domains: (1) fixed-wing aircraft navigation through challenging terrain and (2) simultaneous station-keeping and dynamic obstacle avoidance with a quadcopter.
2) Interpretable design achieved by auditing the learned models and extracting the corresponding decision trees.
3) Sim-to-Real transfer of Neural Circuit Policy models onto highly compute- and power-constrained physical flight hardware, running inference in real time on a Raspberry Pi Zero.

II. Related Work

While policies trained with deep reinforcement learning have achieved considerable success in environments including Atari [4], [5] and various board games [6], [7], there has been limited prior work on developing agents whose learned policies are interpretable by design. Most prior work has focused on post-hoc analysis of agents' policies rather than studying the network activations. This has been driven, in particular, by studies of interpretability in image classification [8], [9], which then led to the use of saliency maps for both image classification [10] and reinforcement learning agents [11], [12]. Other approaches include building simpler models (not derived from the neuron activations) for post-hoc analysis of black-box models [13], [14].

In addition to the improved interpretability afforded by the relatively small number of neurons in NCPs, their small size also implies a small footprint in computer memory requiring tens or hundreds of parameters and kilobytes or megabytes of memory. Prior deployment of deep learning in aerial domains has required millions of parameters and gigabytes of memory [15], [16]. The small size of the NCP networks also lends itself to fast inference times on low-powered computers, including real-time inference on a battery-powered Raspberry Pi Zero and 30 Hz inference on a battery-powered Nvidia Jetson Nano.

A complementary direction is developing agents that can learn with fewer interactions with the environment [17]; however prior work on this has not bounded the size of the neural network involved in the decision-making process.

Prior work on Neural Circuit Policies for reinforcement learning [2] has focused on much simpler environments, which can be learned by relatively simple non-NCP networks as well, and also does not address Sim-to-Real transfer of learned policies. In contrast to prior work on NCPs [1], we interpret neuron activations directly to examine whether the learned policy has captured key aspects of the environment.

III. Neural Circuit Policies (NCPs)

A. Foundations of Neural Circuit Policies

A Neural Circuit Policy network [1] is a sparse neural network inspired by biological systems, in particular the *C. elegans* nematode, consisting of four layers of neurons: the sensory neurons, inter-neurons, command neurons, and motor neurons. The *sensory neurons* (also known as *afferent neurons*) take in observations from the environment, typically pre-processed by a feature-extraction component such as a Convolutional Neural Network (CNN) architecture. The *inter-neurons* are connected largely by feedforward synapses to the sensory neurons. The *command neurons*, which comprise the next layer, are connected to the inter-neurons via highly-recurrent synapses; in biological systems, they are responsible for the decision-making. The last layer of the NCP network, connected via feedforward connections to the command neurons, consists of *motor neurons* (also known as *efferent neurons*), which are responsible for actuation. Finally, all of the edges are synapses, either *excitatory* (increasing the potential of the neuron with the in-connection, depicted in red) or *inhibitory* (decreasing the potential of the neuron with the in-connection, depicted in blue).

Each neuron has state dynamics governed by an ODE [3], which allow for substantially more complex dynamics than in a traditional neural network architecture. These neurons are referred to as *Liquid Time Constant* (LTC) neurons, where the varying time constant τ governs their ability to express temporal dynamics in the hidden layers.

The expressive power of LTC networks [3, Theorems 4 and 5] allows for them to model more complex dynamics than in standard feed-forward models. In addition, the state dynamics can be readily approximated and solved by a variant of the Euler method [3, Algorithm 1]. Consequently, training time is similar to that of fully-connected network architectures. Training NCP networks can be done with any policy optimization method; in this work, we use PPO [18].

B. Visualization and Interpretability With NCPs

NCPs are designed to be compact (few neurons) and sparse (few edges) networks, allowing for neuron activations to be directly interpreted when key environmental features are varied.

While NCPs can learn very high-quality policies, as we discuss further in Section VII-A, it is also crucial to note that the learned models can be interpreted by directly looking at the neuron activations. In this work, we conducted a series of interpretability experiments on learned policies by visualizing the activation of each neuron in the four layers while varying inputs, both from sensors and auxiliary information about vehicle state. We also performed this analysis over time, looking at the reaction of neurons to key changes in the environment, such as an oncoming obstacle, and observed, for instance, the network successfully reacting to oncoming obstacles in our quadcopter simulation environment by successfully dodging them and then returning to the waypoint when safe to do so.

One major advantage of NCPs is that by using small, interpretable networks, it is possible to directly examine *what* the policy model has learned and how it would react to various changes in the environment. For instance, we observed similar neural activations occurring for an obstacle approaching from multiple angles, as long as that obstacle was on a trajectory to potentially collide with the vehicle. Therefore, in addition to the established approach of evaluating an agent in terms of the reward it can obtain in its environment, as we do in Section VII-A, examining trained models in this way allows for the models to be interrogated prior to deployment. This is especially valuable when the models are deployed to physical hardware, as described in Section VII-C, and testing the safety and reliability of a policy is much more difficult.

In the following sections, we will walk through the entire pipeline of training an NCP model in two aerial domains, and then deployment of a model from one of these domains to real hardware.

IV. Autonomous Flight Environments

We chose to study the efficacy of Neural Circuit Policies for learning control in aerial domains for two primary reasons: (1) aerial vehicles are expensive and may operate in close proximity to humans (or even carry humans as co-pilots or passengers) making safety through interpretable policies an essential consideration, and (2) the domains presented here are challenging, and given their intrinsic stochasticity, agents within these domains must learn robust policies even when faced with unexpected circumstances. Due to these properties, agents are unable to learn a good policy simply through memorization of a particular sequence of actions.

1) Canyon Run: This task considers a fixed-wing airplane navigating a narrow canyon environment. The agent obtains reward by passing through waypoints while learning to trade off reward-seeking behavior with the risk of terrain collisions (Fig. 2, left). The agent receives a reward of 0.5 for each waypoint that it reaches successfully. The waypoints occur at randomly-generated heights within the canyon, requiring the agent to navigate the terrain to reach the waypoints. In addition to the canyon walls, the canyon floor is comprised of hilly terrain, and waypoints may appear below the hilltops, increasing the difficulty of the scenario. The episode ends if the agent collides with the terrain or flies above the canyon.

2) Drone Dodgeball: This domain models quadcopter station-keeping where the agent needs to learn to balance its mission with the threat posed by dynamic obstacles launched at it by an adversary (Fig. 2, right). The agent receives a reward per time step that increases as the agent's distance from the waypoint decreases.[1] A ball launcher constantly tries to launch balls at the quadcopter, randomly changing horizontal position after each shot. The episode ends if a ball hits the quadcopter (with an additional -1 reward penalty), if the vehicle leaves the 100 m $\times$ 30 m $\times$ 100 m boundaries of the environment, or if its tilt exceeds 70°.

The environments both allow for 1 DoF, 2 DoF, or Full DoF control. For Canyon Run, 1 DoF corresponds to controlling only pitch, 2 DoF to controlling pitch and roll, and Full DoF to controlling pitch, roll, yaw, and thrust. For Drone Dodgeball, 1 DoF corresponds to controlling only thrust (i.e., up/down movement), 2 DoF to controlling thrust and roll, and Full DoF to controlling thrust, roll, pitch, and yaw. In all of these modes of control, we use a multi-discrete approximation to continuous control, so that the models output an action in $\{-1, 0, 1\}$, resulting in a decrease/`no-op`/increase of that axis of control.

Observations are generated by *raycast sensors* and provide a way to sense objects along a vector from the sensor origin. We combine arrays of raycasts arranged in a grid to simulate physical depth cameras without requiring rendering, thus increasing training speed. Raw sensory inputs in both Drone Dodgeball and Canyon Run consist of a *dense sensor* ($21 \times 101 = 2121$ raycasts) with a limited field of view, a *sparse sensor* ($19 \times 33 = 627$ raycasts) with a broader field of view, and a vector of additional information about the vehicle state.[2] The two visual sensors approximate central vision (dense sensor: more detailed, narrower field of view) and peripheral vision (sparse sensor: less detailed, broader field of view) in human visual perception.

These custom environments enable us to vary environmental features and thus test interpretability in a controlled way. The environments were built with the Unity game engine using the ML-Agents package for reinforcement learning [19]. An advantage of using Unity for simulation is that its physics engine allows reasonable approximation of flight dynamics. We consider Canyon Run to represent a much more complex domain than Drone Dodgeball, due to the longer time horizon and sequences of actions required to learn a high-reward policy. Therefore, we believe that the stronger performance of NCPs relative to other network architectures in this domain, as discussed further in Section VII-A, is due to the significantly greater temporal and goal complexity of Canyon Run.

VI. Solution Overview

Our general framework provides the agent with visual input from two depth sensors along with vector-valued information

[1]The reward is $\max(0, 1/(d + 0.5) - 0.1)$ per time step for being a distance d from the waypoint.

[2]For Drone Dodgeball, the vector has information about the location of the waypoint relative to the vehicle. In Canyon Run, the vector has waypoint information and other vehicle information, including altitude, position, and rotation (corresponding to data from common flight instruments).

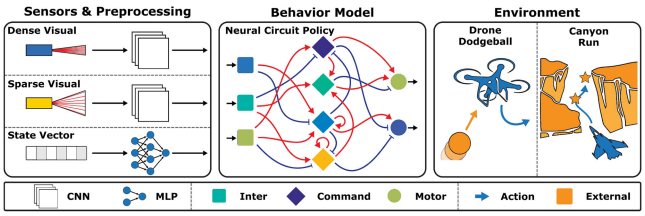

Fig. 3. The agent receives visual input from two depth sensors along with vector-valued state information. The inputs are processed by Convolutional Neural Networks (CNNs) and a Multi-layer Perceptron (MLP), respectively. The resulting latent features are passed to the behavioral model, represented here as a Neural Circuit Policy (NCP), where suitable action outputs are learned for controlling the vehicle in the environment. The NCP provides a bio-inspired architecture that combines specialized neurons arranged in inter-neuron, command, and motor layers with sparse connectivity patterns.

TABLE I
NEURAL NETWORKS FOR EACH ENVIRONMENT

Network	Number of layers	Network configuration
NCP (small)	3: inter, command, motor	{9, 14, 24} neurons
LSTM (small)	3	{9, 14, 24} hidden units
LSTM (large)	1	128 hidden units
MLP	1	1024 neurons

about the current vehicle state. The raw sensor readings are pre-processed using a Convolutional Neural Network (CNN) for each of the visual inputs and a Multi-layer Perceptron (MLP) for the state inputs, as illustrated in Fig. 3. The resulting latent features are then sent to the behavioral model, which represents the agent's policy. Here, we consider the Neural Circuit Policy (NCP) architecture due to its compact representation and high expressive power (Fig. 3, middle). However, we also provide comparisons to alternative feed-forward and recurrent architectures as outlined in Table I. The behavioral model, defining the agent's policy, learns a conditional mapping from latent features to action outputs and therefore provides the control signals for the agent to navigate the environment. The resulting pipeline can be trained end-to-end using any suitable reinforcement learning algorithm. Here, we employ PPO [18] due to its robust learning capabilities, requiring minimal hyperparameter tuning for most environments, and efficient parallelization. Our implementation combines an NCP architecture based on the `keras-ncp` library [1] with PPO training as provided by the RLLib library [20]. Throughout our experiments, we consider a fixed CNN[3] configuration for each depth sensor and evaluate several different NCP architectures.[4]

[3]As a robustness check, we evaluated five different CNN configurations of increasing depth, but found no significant difference in the resulting agents' performance. We therefore chose the smallest CNN configuration that we tested, namely: using the notation [num_channels, kernel_size, stride], [32, 11, 4] for the dense sensor and [16, 5, 4] for the sparse sensor.

[4]We performed a robustness check in which we evaluated different wirings and configurations of NCPs, specifically varying the number of neurons in the NCPs (inter-neurons, command neurons, and motor neurons) from 9 (4 inter-neurons, 2 command neurons, 3 motor neurons) to 40 (16 inter-neurons, 12 command neurons, 12 motor neurons.). The final learned policy, corresponding to the model checkpoint after training for 12 million time steps, is evaluated over 100 episodes in each simulation domain. For interpretability experiments, we tested neuron activations in NCPs in response to key factors in the environment, to determine whether they corresponded to the factors that humans would pay attention to. Additionally, the NCP model for Drone Dodgeball was tested using depth image input from a physical depth camera, and interpretability experiments were repeated with real-world measurements.

TABLE II
TRAINING SETTINGS

Environment Name	Control DoF	Networks Tested
Drone Dodgeball	1, 2, full	NCP(s), LSTM(s,l), MLP
Canyon Run	1, 2, full	NCP(s), LSTM(s,l), MLP

VII. RESULTS

A. Training in Simulation

For each of the two environments, *Canyon Run* and *Drone Dodgeball*, we trained agents over 12 million time steps. The training results are depicted in Fig. 4, bottom. We trained four different neural networks (Table I) for two environments, with three different control modes (degrees of freedom) for each environment (Table II). We also ran a number of additional experiments, comparing the performance across various network depths and larger sizes (up to 10x larger), but found no significant difference in performance.

In the test results, as depicted in Fig. 4, top, we evaluated each model over 100 episodes. We also compared the performance of the learned models to an agent taking random actions, and found that the performance of a random-action agent is extremely low compared to any of the trained models (average reward of 64 for 1 DoF, 12 for 2 DoF, and 3 for Full DoF in Drone Dodgeball, and <1 for each control mode of Canyon Run), indicating that the trained agents learned non-trivial policies.

Overall, we found that in the simpler Drone Dodgeball environment, the performance of NCPs was comparable to the performance of the other neural architectures. In the more complex domain of Canyon Run, we found that NCPs tended to demonstrate relatively higher performance both during training and when testing the fully-trained models.

However, as we discuss next, in addition to being performant, a key advantage of NCPs is that the models can be interpreted, with neural activations mapped to human-interpretable decisions.

B. Network Interpretability

The compact size of NCPs allows for us to study whether the individual neurons have learned to react to salient environmental changes and gain confidence in the decisions that they would make. Fig. 5 shows normalized activations of individual NCP neurons with respect to varying vertical waypoint distances in Canyon Run (left) and normalized activations with respect to an oncoming obstacle in Drone Dodgeball (right). Concretely, these figures demonstrate that, even without needing to evaluate the network over a real trajectory, that the network has learned to respond to salient conditions in the environment and take the appropriate actions.

As depicted in Fig. 5, the next step after observing the neuron activations in response to key environmental changes is to abstract the decisions into *human-readable decision trees*. The low-dimensional policy representation learned by NCPs

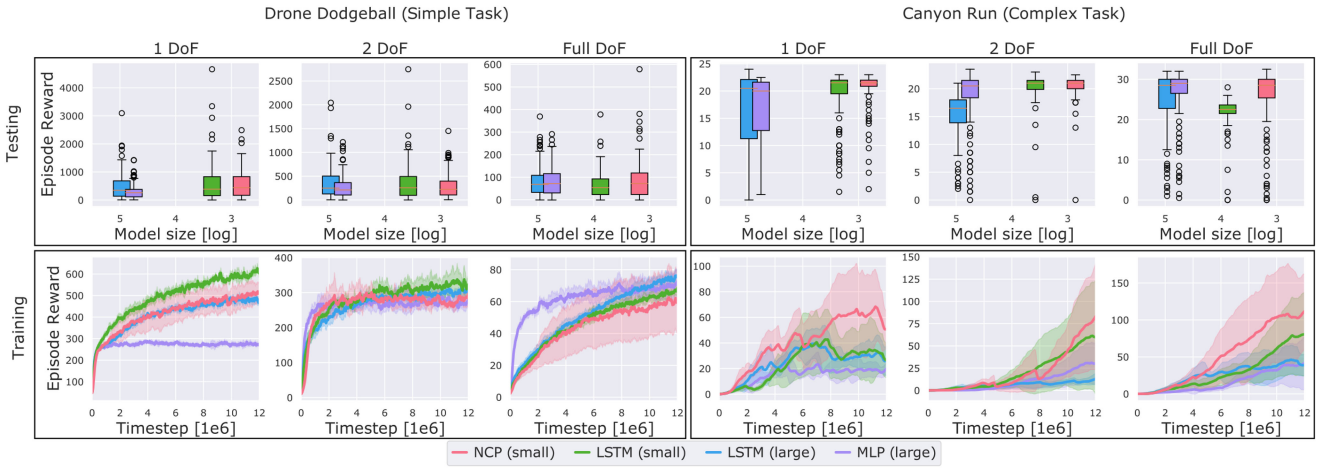

Fig. 4. Training and test results, comparing NCPs, small LSTMs, large LSTMs, and large MLPs. All of the models were trained for 12 million time steps. The average episode length in Canyon Run is much longer than in Drone Dodgeball, leading to smoother rewards during training. The test results were the evaluation of the best training run of each model architecture at 12 million time steps over 100 episodes. Due to the agents' very high performance in Canyon Run, we truncated episodes in that environment at 5,000 time steps during testing, so reward in a given episode could never exceed 25 in 1 DoF and 2 DoF (throttle fixed at 50%), and 36 in Full DoF (agent sets throttle, up to 100%).

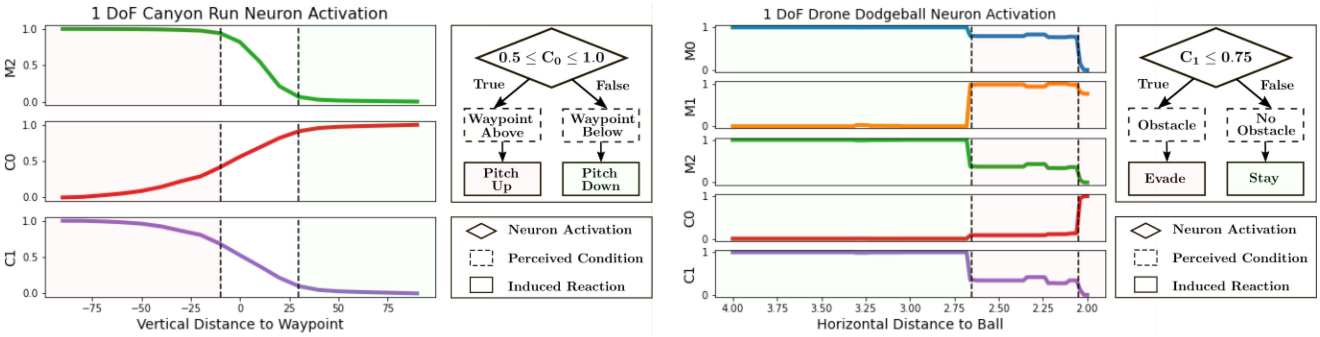

Fig. 5. Neuron activations and example decision trees for 9-neuron NCP models trained in 1 DoF variants of *Canyon Run* (left) and *Drone Dodgeball* (right). Individual neurons have learned to respond to key features in the environments such as the direction to the next waypoint (left) and presence of obstacles (right). Consistent neuron specialization facilitates extraction of behavior patterns into decision trees together with natural language explanations for specific actions conditioned on input stimuli.

facilitates post-hoc analysis of activation patterns conditioned on various inputs. The extracted behavior patterns can then be distilled into a reasoning module, such as a decision tree, whose forward-pass is almost instant and can provide real-time human-readable explanations. This, combined with the small scale of NCPs, allows for the successful deployment of interpretable policies to the environments presented here. We leave developing efficient algorithms for extracting these decision trees in arbitrary environments to future work.

Further expanding interpretability evaluations, Fig. 6 illustrates activations for all 14 neurons in the 2 DoF *Drone Dodgeball* environment. While keeping sensory inputs constant, the waypoint was varied across a 2D grid. The heat maps represent the normalized neuron activations in the network before and after training. Even without actuation, this visualization allows us to inspect the learned model to determine that it is responding appropriately. Notably, we see the emergence of spatial understanding in the trained network: the C1 and C2 command neurons effectively respond to the X and Y coordinates of the waypoint, which then propagates to the motor neurons. The activations of the M0 and M1 motor neurons in the trained network respond to changes in the Y coordinate of the waypoint, and the activations of the M3 and M5 neurons respond to changes in the X coordinate. Similarly, Fig. 7 visualizes neuron activations in response to varying waypoint locations for the Full DoF mode of Drone Dodgeball. For example, we observe a clear specialization of command neuron #0 to waypoints in the half-space with positive x-coordinates (top left), while command neuron #2 shows complementary behavior with very little response to stimuli in that region (top right). This further underlines the consistent specialization of individual neurons in the NCP agents.

C. Deployment on Drone Hardware

We implemented the NCP trained in the Drone Dodgeball simulation on a physical hardware system that consists of a Raspberry Pi Zero that runs inference on the trained model and a DJI Tello drone, providing altimeter observations and RGB video frames used as sensory input to the model (Table III). In order to be able to run inference on the trained model in real time on a Raspberry Pi Zero, we extracted the weights from the

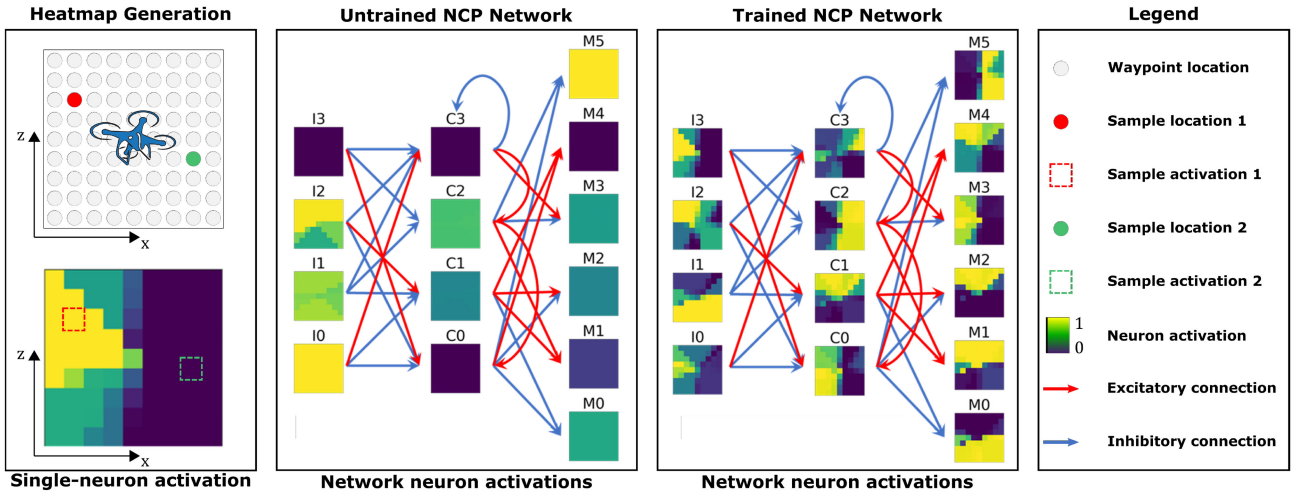

Fig. 6. Heat maps of neuron activations in the 14-neuron NCP model trained in the 2 DoF variant of *Drone Dodgeball*. Heat maps are generated for each neuron by placing waypoints (stimulus) at different locations on a 2D grid and recording the induced activation (left box). We can then visualize the resulting activations for the entire compact NCP model. Starting from nondescript activity patterns in the untrained NCP model (middle), we observe consistent specialization of individual neurons to specific half-spaces in the trained NCP model (right). The consistent smooth variation in learned neural activity indicates that similar stimuli will induce similar agent behavior.

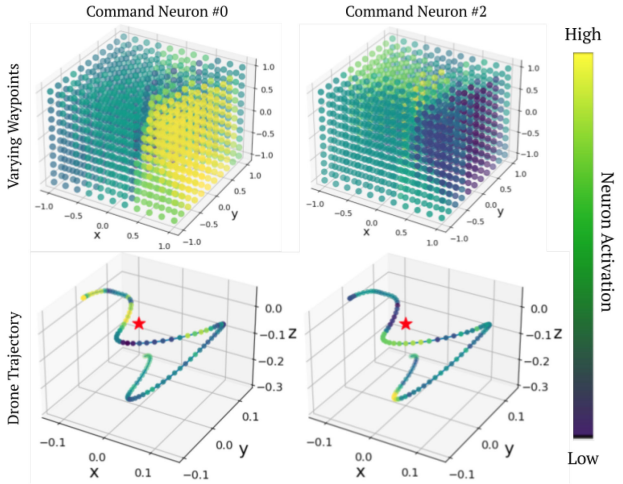

Fig. 7. Neuron activations in the 24-neuron NCP models in Full DoF *Drone Dodgeball*. Top row: example neural responses of two command neurons to varying the waypoint position across a 3D grid. Bottom row: corresponding neuron activations along a trajectory approaching a waypoint located at the origin (red star). Brighter color indicates stronger activation.

TABLE III
DRONE HARDWARE SETTINGS

Drone	Computing hardware	Sensing
DJI Tello	Raspberry Pi Zero	Altimeter, RGB video
Large quadcopter	Jetson Nano	Intel RealSense camera

model trained in simulation and built a TensorFlow Lite model (∼170 KB) and associated observation preprocessor (∼10 KB) for deployment.

We also tested inference using a larger quadcopter model, namely an NXP RRDRONE-FMUK66 flight controller running the PX4 flight stack [21], an Nvidia Jetson Nano board that runs inference on the trained model and the ROS [22] computation graph, and an Intel RealSense d435i stereo camera that generates

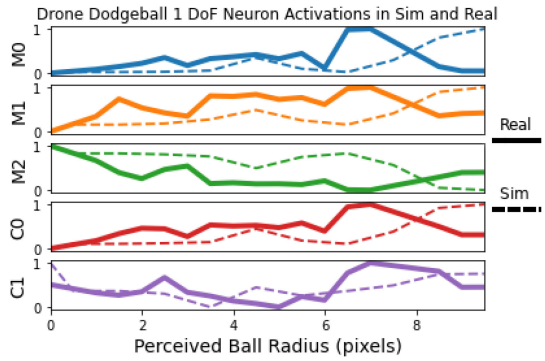

Fig. 8. Normalized neuron activations for simulated (dashed) and real-world (solid) 1 DoF Drone Dodgeball as a function of perceived ball radius in pixel space. The neurons exhibit similar patterns of activation, consistent with the expected dodging behavior in response to an oncoming obstacle.

depth images used as sensory input to the model (Table III). We successfully demonstrated onboard, real-time inference of trained NCPs that produce dodging behavior during flight tests. Early experiments demonstrated inference at 10 frames per second (FPS) with later flight tests achieving >30 FPS due to flight software updates to improve video stream handling. Due to the larger size, lower maneuverability, and fire hazards presented by the large LiPo batteries, all flight tests on the NXP RRDRONE-FMUK66 platform were conducted at low-speed and ground tethered.

Fig. 8 compares the neuron activations in the simulated Drone Dodgeball environment with those generated during real-world hardware experiments. Fig. 9 (center and right panels) shows two frames of depth images taken from the onboard Intel RealSense while a ball is being thrown at the drone. Some important similarities illustrate the successful transfer of the model to the sensor in physical flight hardware. Each neuron displays an increase in activation as the perceived ball radius increases—except the M2 neuron which displays a decrease—in both simulated and

Fig. 9. The larger drone hardware setup for deploying the NCP policies from the Drone Dodgeball simulation environment on physical flight hardware. The quadcopter, which uses an Intel RealSense camera for visual input and a Jetson Nano for processing and inference, is shown on the left. The image is downsampled and processed to make it consistent with the expected model input. The inner rectangle represents the grayscale depth pixels passed to the NCP as input and the colored numbers represent the corresponding NCP action output. The middle image shows the model staying at the waypoint when no obstacle is in view. The right image shows the model lowering its thrust in response to an oncoming obstacle.

Fig. 10. The DJI Tello drone experimental setup for deploying the NCP policies from the Drone Dodgeball simulation environment on physical flight hardware. Inference is done on a battery-powered Raspberry Pi Zero (not shown). The left image shows the experimental setup, in which a ball is moved into the visual field of the drone to test whether it will move out of the way. The middle frame shows a view from the on-board camera. The right frame shows an example observation that the trained NCP inference model receives, which is processed via color masking, resizing, and grayscaling.

real-world data. The primary difference between the observed neuron activations in real-world deployment is that they peak around a ball radius of 7 pixels, whereas simulation-based activation peaks at around 9 pixels. We speculate that this difference comes from how depth scale was converted to grayscale for simulated versus real data (simulated data uses a Unity-defined length scale, while RealSense data is converted to meters). Furthermore, effective field of view of the simulated depth sensor was 90°-by-90°, whereas the RealSense data was cropped from a smaller 86°-by-57° field of view. However, as illustrated in Fig. 9, the drone's policy actions were consistent with the expected dodging behavior.

With the DJI Tello drone experiment (depicted in Fig. 10), we used the altimeter measurement from the drone at takeoff (corresponding to $\sim$1 m) to set the location of the waypoint. Observations from the drone's camera were sent in real time to a battery-powered Raspberry Pi Zero, which processed the image via color masking (corresponding to the obstacle's hue), resizing, and grayscaling, and then passed it to the trained NCP model along with the vector to the waypoint. The drone, controlled by the trained NCP running on the Raspberry Pi Zero, then remained at a steady hover until the obstacle was presented, at which point it began to consistently increase thrust and move upwards, corresponding to a successful dodge maneuver.

VIII. Future Work

As we have demonstrated here, NCPs allow for high-quality policies to be learned in complex environments that are also auditable and can be mapped to interpretable decision-making. Whether NCPs, and interpretable-by-design models in general, lead to better decisions than black-box models remains an open question. Also, applying NCPs to more complex, varied environments represents an interesting direction for future work.

In addition to the standard metric of evaluating the raw performance of deep neural networks as black boxes, the method of evaluation presented here—studying the human-interpretable conditions that a model has learned to take actions in response to—is a way to diagnose the potential failure modes of models, whether due to problems in perception or actuation. We believe that incorporating interpretability into the design of trained models is crucial for developing trustworthy AIs. In particular, developing models capable of mapping from the finite set of scenarios encountered during training/testing to trust in the capabilities of the autonomous system as a whole, including in novel and surprising scenarios, remains a challenging open problem. A related open question is whether a model that has learned human-interpretable features would have better performance in such scenarios than networks that have not learned human-interpretable features.

On the hardware experimentation side, the mass of a Raspberry Pi Zero (<10 grams) allows for it to be mounted on the DJI Tello and powered from the drone's battery, so that inference could be run entirely on board. We tested powering the Raspberry Pi Zero in this way, and performing inference with the drone in a static location, but deploying the model entirely on board is a direction for future work. Furthermore, the small size and low computational demands of NCP inference may also lend itself to deployment on microscale robotics with minimal processing hardware [23].

IX. Conclusion

In this paper, we have showed that Neural Circuit Policies can learn interpretable control in two domains of increasing complexity, and that the neuron activations in the learned models directly correspond to key features in the environment. Unlike other deep neural network architectures, in which there is a tradeoff between interpretability and performance, NCPs allow for learning policies that are both highly performant and interpretable.

We have also demonstrated inference, using the NCP model learned in the Drone Dodgeball environment, deployed on physical flight hardware. The use of interpretable models for training agents in complex domains provides many opportunities as we deploy agents in the real world. We anticipate that agents trained with such models will become increasingly useful for safe AI deployment and future human-AI collaboration.

Acknowledgment

The authors would like to thank the MIT Supercloud and Lincoln Laboratory Supercomputing Center for providing high-performance computing and consultation resources that have contributed to the research results reported within this paper, also would to thank R. Hasani for an introduction and initial discussions about Neural Circuit Policies and their applications to the tasks studied in this paper, and also would like to thank L. Yin for helpful discussions and suggestions. The authors are also grateful to The Boeing Company for support. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

[1] M. Lechner, R. Hasani, A. Amini, T. A. Henzinger, D. Rus, and R. Grosu, "Neural circuit policies enabling auditable autonomy," *Nature Mach. Intell.*, vol. 2, no. 10, pp. 642–652, 2020.

[2] R. Hasani, M. Lechner, A. Amini, D. Rus, and R. Grosu, "A natural lottery ticket winner: Reinforcement learning with ordinary neural circuits," in *Proc. 37th Int. Conf. Mach. Learn.*, 2020, pp. 4082–4093.

[3] R. Hasani, M. Lechner, A. Amini, D. Rus, and R. Grosu, "Liquid time-constant networks," in *Proc. AAAI Conf. Artif. Intell.*, pp. 7657–7666, 2021.

[4] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[5] A. P. Badia *et al.*, "Agent57: Outperforming the atari human benchmark," in *Proc. 37th Int. Conf. Mach. Learn.*, 2020, pp. 507–517.

[6] J. Schrittwieser *et al.*, "Mastering Atari, Go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[7] N. Brown and T. Sandholm, "Superhuman AI for multiplayer poker," *Sci.*, vol. 365, no. 6456, pp. 885–890, 2019.

[8] A. Wan *et al.*, "NBDT: Neural-backed decision tree," in *Proc. Int. Conf. Learn. Representations*, 2021.

[9] B. Kim *et al.*, "Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (TCAV)," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 2668–2677.

[10] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim, "Sanity checks for saliency maps," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 9525–9536.

[11] S. Greydanus, A. Koul, J. Dodge, and A. Fern, "Visualizing and understanding atari agents," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 1792–1801.

[12] N. Puri *et al.*, "Explain your move: Understanding agent actions using specific and relevant feature attribution," in *Proc. Int. Conf. Learn. Representations*, 2020.

[13] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 4768–4777.

[14] H. Lakkaraju, E. Kamar, R. Caruana, and J. Leskovec, "Faithful and customizable explanations of black box models," in *Proc. AAAI/ACM Conf. AI, Ethics, Soc.*, 2019, pp. 131–138.

[15] F. Sadeghi and S. Levine, "CAD2RL: Real single-image flight without a single real image," *Robot.: Sci. Syst.*, 2017.

[16] N. J. Sanket *et al.*, "EVDodgeNet: Deep dynamic obstacle dodging with event cameras," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 10651–10657.

[17] A. Pritzel *et al.*, "Neural episodic control," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 2827–2836.

[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, [Online]. Available: *arXiv:cs.LG/1707.06347*.

[19] A. Juliani *et al.*, "Unity: A general platform for intelligent agents," 2020, [Online]. Available: *arXiv:cs.LG/1809.02627*.

[20] E. Liang *et al.*, "RLlib: Abstractions for distributed reinforcement learning," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 3053–3062.

[21] L. Meier, D. Honegger, and M. Pollefeys, "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2015, pp. 6235–6240.

[22] M. Quigley *et al.*, "ROS: An open-source robot operating system," in *Proc. ICRA Workshop Open Source Softw.*, 2009.

[23] K. Y. Ma, P. Chirarattananon, S. B. Fuller, and R. J. Wood, "Controlled flight of a biologically inspired, insect-scale robot," *Sci.*, vol. 340, no. 6132, pp. 603–607, 2013.