Robust Flight Navigation with Liquid Neural Networks

by

Patrick Kao

B.S. Computer Science and Engineering
Massachusetts Institute of Technology, 2022

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2022

Author .....
Department of Electrical Engineering and Computer Science
May 6, 2022

Certified by .....
Daniela L. Rus
Director
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Robust Flight Navigation with Liquid Neural Networks

by

Patrick Kao

Submitted to the Department of Electrical Engineering and Computer Science
on May 6, 2022, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Autonomous robots can learn to perform visual navigation tasks from offline human demonstrations, and generalize well to online and unseen scenarios within the same environment they have been trained on. It is fundamentally challenging for these intelligent agents to take a step further and robustly generalize to new environments with drastic scenery changes they have never encountered before. Here, we present a method to create robust flight navigation agents that successfully perform vision-based fly-to-target tasks beyond their training environment under drastic distribution shifts. To this end, we design an imitation learning framework utilizing liquid neural networks, a brain-inspired class of continuous-time neural models that are causal and adapt to changing conditions. We observe that liquid agents learn to distill the task they are given from visual inputs, and drop irrelevant features. This way, they transfer their learned navigation skills to new environments. When compared to other advanced deep agents, we confirm this level of robustness in decision-making is exclusive to liquid networks, both in their differential equation and closed-form representation.

Thesis Supervisor: Daniela L. Rus
Title: Director

Acknowledgments

I would like to thank my direct supervisors Ramin Hasani and Aaron Ray along with faculty supervisor Daniela Rus for introducing me to this project, checking up on my progress, and providing me guidance and direction in my research. I would also like to thank collaborators Makram Chahine, Ryan Shubert, Mathias Lechner, and Alexander Amini for helping with online testing, learning procedure design, and other details that made these endeavors possible. I would also like to thank Joseph Feld, Ita Futran, Kartikesh Mishra, Patrick Whartenby, Charles Vorbach, Tsun-Hsuan Wang, and Wei Xiao for platform setup and testing out NCP and CfC models for confirmation of results across other domains. Additionally, I would like to acknowledge Sildomar T. Monteiro of Aurora Flight Sciences for providing constructive feedback and validating our experimental setting throughout this project. I would like to thank IBM for generously providing access to the Satori computing cluster and MIT/MIT Lincoln Labs for providing access to the Supercloud computing cluster. Both clusters were very helpful for training models and hyperparameter optimization.

Contents

1 Introduction **17**

2 Related Work **25**

2.1 Imitation Learning 25

2.2 Applications in Visual Navigation 27

2.3 Testing Liquid Networks 28

3 Model Background **29**

3.1 Fundamentals of Continuous-Time Recurrent Neural Networks . . . 29

3.2 LTC Equations 31

3.3 NCP Wiring 32

3.4 Closed-form Solutions 33

3.5 Causality of Liquid Networks 34

4 Method **35**

5 Experiments and Results **51**

6 Conclusions **65**

6.1 Discussion 65

6.1.1 Choice of models matter for out-of-distribution generalization. 65

6.1.2 Why can liquid networks generalize to OOD settings? 67

6.1.3 Robustness in perception and control. 68

6.2 Lessons Learned 69

6.3 Future Work 69

A Resources **71**

List of Figures

1-1 **Liquid Neural Networks. A.** schematic demonstration of a fully connected liquid time-constant (LTC) layer [28]. The dynamics of a single neuron i is given, where $\mathbf{x}_i(t)$ represent the state of the neuron i, and τ represents the neuronal time-constant. Synapses are given by a nonlinear function, where $f(.)$ represents a sigmoidal nonlinearity, A_ij is a reversal potential parameter for each synapse from neuron j to i. **B.** Schematic representation of a closed-form liquid network (CfC) together with its state equation. This continuous-time representation consists of two sigmoidal gating mechanisms, σ and $1-\sigma$ with their activity being regulated by the nonlinear neural layer f. f acts as the *liquid time-constant* for CfCs. These gates control two other nonlinear layers g and h to create the state of the system. **C.** A schematic representation of a neural circuit policy (NCP) [41]. The network architecture is inspired by the neural circuits of the nematode C. elegans. It is constructed by 4 sparsely connected LTC layers called: sensory neurons, interneurons, command neurons and interneurons. **D.** Sparse CfC. A sparse neural circuit with the same architectural motifs as NCPs this time with CfC neural dynamics. 20

3-1 **NCP vs. Conventional RNN:** Depiction of NCP (left), a continuous-time neural network, compared to a standard discrete-time network (right). The hidden state of a CTRNN is continuous, while a conventional RNN's state is only updated at discrete periodic intervals. . 31

4-1 **End to end learning setup. A.** Policies were trained to solve the following task: Using only images taken from an onboard camera, navigate the drone from its starting location to a target 10m away, keeping the object in frame throughout. **B.** One goal of the work was to explore the generalization capability of various neural architectures by adapting the task to previously unseen environments. **C.** Another goal was to understand the causal mechanisms underlying different networks' behaviors while completing the task i.e. by visualizing the network input saliency map. **D.** The training process starts by hand-collecting human expert trajectories, recording camera observations and human expert actions. **E.** To further convey the task, synthetic sequences are then generated by taking images collected during human flights and repeatedly cropping them, creating the appearance of a video sequence in which the drone flies up to the target and centers it in frame. **F.** The networks are trained offline on the collected expert sequences using a supervised behavior cloning MSE loss. **G.** We then deploy trained networks on the drone in online testing and observe task performance under heavy distributional shift, varying lighting, starting location, wind, background setting, and more. 36

4-2 **Sample frames: A.** Third-person view of the quadrotor and one of the targets in the woods where data is collected. **B.** Third-person view of the quadrotor during testing on an urban campus patio, with multiple adversary objects dispersed around the target camping chair. **C.** Training data frame samples from the quadrotor onboard camera containing various targets (camping chair, storage box, and RC car from top to bottom) and taken during different seasons (summer, fall, and winter from left to right). **D.** Test data frame samples from the quadrotor onboard camera against each of the four test backgrounds: Training Woods (top left), Alternative Woods (top right), Urban Lawn (bottom left), and Urban Patio (bottom right). . . 37

4-3 **Losses: A.** Training loss plots for all model types (lower is better). Observe that LSTM has the best training loss, but poor test-time performance, implying training loss is a poor indicator of task performance.. **B.** Validation loss plots for all models. Observe that the general trend for validation loss is relatively horizontal. We empirically observe that the overfit models with the lowest train loss perform best on the task while checkpoints with the lowest validation loss fail. 43

4-4 **System block diagram:** A custom ROS node processes gimbal camera images and runs inference on them in a separate thread. The resulting velocity commands are translated into motor thrusts by an onboard flight controller. 46

5-1 **Stress tests:** These experiments run both an original image sequence collected by the drone and a perturbed version through a network, integrate the output velocity commands, and record the final displacement between sequences (lower is better). Bars show the standard deviation across 10 runs. Trendlines for NCP, LSTM, and Sparse-CFC are shown with shaded brown, orange, and gray lines respectively. Observe that liquid networks are generally more robust to perturbation than their discrete counterparts; NCP outperforms LSTM on noise, brightness, and contrast perturbations, while Sparse-CfC outperforms LSTM in all cases. (n=10) **A.** Distances when Gaussian noise with std deviation x is added to image frames. **B.** Distances when brightness is shifted by x times the maximum pixel value. **C.** Distances when contrast in the image is multiplied by x. Note that the first data point, 0.5 is actually a *reduction* in contrast, and moves all pixel values *towards* the mean, while the other 3 data points *increase* contrast and move pixel values *away* from the mean. **D.** Distances when saturation (S in HSV colorspace) is multiplied by x. Note the first data point, 0.5, *reduces* the saturation of colors in the image while the other 3 data points *increase* saturation 59

5-2 **Attention maps. A.** A frame illustration of the attention profile of different networks computed by the VisualBackProp saliency map computation algorithm [7]. Top row shows the input image frame. Bottom row shows the saliency map. The brighter regions indicates a greater concentration of the convolutional layer's attention. Saliency scores range between 0 (black) and 1 (white). **B.** A flight case study where an LSTM agent loses its target and gets confused by an adversary. Left to right shows time progress. This is while liquid NCPs manage to maintain their focus on the target and complete the task. **C.** A similar scenario as in (b), in a different out-of-distribution testing environment. 60

5-3 **Training Environment Comparison:** Saliency maps from networks before and after augmentation run on one of the hardest sequences from the training set (black means least contribution to network decision and yellow means highest contribution to network decision). Observe that before augmentation, LSTM, CT-RNN, and GRUODE place little saliency on the chair, instead focusing on edges with lighting changes. The NCP is able to attend to the chair the earliest out of all models, starting at frame 2, but it too places focus on spurious image edges. After augmentation, LSTM, NCP, CFC, and Sparse-CFC are consistently able to attend to the chair after frame 2, and have significantly less noise. The NCP saliency map is still one of the cleanest after augmentation, containing minimal artifacts in frames 1-4 and containing almost none afterwards. 61

5-4 **Lab Environment Comparison:** Saliency maps from networks before and after augmentation run on a sequence taken from a cluttered indoor lab environment (black means least contribution to network decision and yellow means highest contribution to network decision). The variety of objects littered throughout the scene creates an image very dissimilar to the train environment. Note that without augmentation, only TCN and NCP focus attention on the chair, and all of the other networks actually focus their attention on the yellow square toolbox instead. When trained with augmentation, all of the model architectures are able to attend primarily to the chair. However, all networks except for NCP and CFC also attend to a thin line above the chair's position. 62

5-5 **Distractor Chair Environment Comparison:** Saliency maps from networks before and after augmentation run on a sequence from online testing captured on the patio (black means least contribution to network decision and yellow means highest contribution to network decision). The difficult lighting and presence of distractor chairs create a very challenging scenario. Before augmentation, observe that the NCP is the only network to focus attention on the target chair in frames 1-3. Even afterwards, the NCPs saliency map is the most focused on the chair, and contains very little attention on the bright regions caused by reflection. TCN, LSTM, CFC, and Sparse-CfC also pick up on the chair in frames 4-7, but also assign a lot of focus to spurious image regions. After augmentation, the saliency maps of all netowrks improve, and all of them except for TCN focus on the chair in frames 4-7 and contain much less extraneous focus than before. Note that all networks except for NCP focus on a sunny line in the top left corner. 63

List of Tables

4.1 **Hyperparameters:** Best hyperparameters for each neural architecture found via TPE sampling. Fixed parameters were not found through optimization, and were set ahead of time. 48

4.2 **Number of parameters:** For each model, the number of trainable parameters in the recurrent part of network is listed (not including the CNN backbone, which is listed separately at the end of the table). Observe that the liquid architectures that performed the best, NCP and Sparse-CfC, achieve superior task performance and robustness to perturbation than the LSTM despite having over 10x fewer trainable parameters. 49

4.3 **CNN Architecture:** Shared architecture of the CNN backbone that processed visual inputs for the different recurrent neural architectures. 49

4.4 **Training hyperparameters:** Miscellaneous parameters that were shared across all model types. 49

5.1 **Online test results:** Closed-loop evaluation of the fly-to-target task for the trained policies in four testing environments. Quantitative results are success rates. Higher is better. (n=20) 54

5.2 **Range test results:** Range test success rates for trained LSTM and NCP policies. (n=5) 54

Chapter 1

Introduction

Intelligence in natural brains fundamentally differs from today's deep learning systems. The differences are rooted in the ways these systems learn to make sense of their environment and manipulate it to achieve their goals. Natural learning systems interact with their environments to understand their world. We define understanding as the ability to *capture causality and contexts* while learning *abstract* concepts to reason, plan and control [43]. This rich representation learning capability allows them to extrapolate and perform inference and credit assignment even in novel scenarios even *out-of-distribution* (OOD). This is beyond what today's deep learning systems can achieve, as statistical learning theory by convention holds only for independent and identically distributed (i.i.d.) settings [70].

Studying natural brains effectively narrows the search space of possible algorithms for acquiring intelligent behavior. For instance, neural circuits in brains are significantly more robust to perturbations and distribution shifts while also being more flexible in tackling uncertain events compared to deep learning systems. This is because they deploy both *unconscious* (to facilitate changes in distributions, faster) and *conscious* (to distill causal structure of data and manipulate learned concepts and models) processes [43] for decision-making. In contrast, today's deep learning systems are incapable of capturing causality and use extracted concepts and features explicitly, despite being able to learn fast from implicit types of observations.

We can readily take advantage of foundational properties of natural learning systems to transform our current deep representation learning frameworks, especially for real-world deployment of artificial learning systems as intelligent *agents*.

In this study, we explore how biological priors on neural models and network architectures can lead to more flexible, robust and understandable decision-making for autonomous robots. In particular, we investigate how to construct agents capable of generalizing and achieving zero-shot transfer to new environments.

To study this, we focus on a spectrum of end-to-end drone navigation tasks from pixel inputs. For example, consider a scenario where the objective is to navigate a drone agent to a static target placed in a forest environment. We aim to train neural network agents on a few runs of data collected by a human pilot in an offline supervised learning setting, and observe if the agents transfer well under drastic changes of scenery and conditions. Performing vanilla imitation learning on raw expert demonstrations is sample-efficient, but generally results in a poor closed-loop (active) testing performance, due to policy stationarity and compounding errors [62]. There has been numerous research on improving generalization performance of few-shot, one-shot and zero-shot imitation learning agents by adopting augmentation strategies [57, 74], human interventions [33, 24], goal-conditioning [16, 23, 45, 19], reward conditioning [66, 50, 13], task-embedding [32], and meta-learning [20].

These advances help design a better gradient descent-based learning scheme without much consideration of the structure of the underlying policy architecture. In particular, there is evidence that brain-inspired neural dynamics significantly improve the robustness of decision-making process in autonomous agents, leading to better transferability and generalization in new settings under the same training distribution [41, 42, 27, 72]. We aim to take advantage of these modeling pipelines and empirically demonstrate that if the true causal structure of a given task is captured by a neural model from expert data, then they can perform robustly even out-of-distribution (OOD). In a set of fly-to-target experiments with different time-

horizons, we show a certain class of brain-inspired neural models manage to generalize well to many OOD settings, well beyond state-of-the-art models.

A bottom-up approach to building brain-inspired neural models is to look into how neurons interact with each other through synapses in small biological brains [42, 27]. We can identify three principled mechanisms for information propagation that are abstracted away in the current building blocks of neural network-based controllers: 1) neural dynamics are typically continuous processes described by differential equations [37], 2) synaptic release is much more than scalar weights; it involves a nonlinear transmission of neurotransmitters, the probability of activation of receptors, and the concentration of available neurotransmitters, among other nonlinearities [36], and 3) the propagation of information between neurons is induced by feedback and memory apparatuses.

These biological priors lead to the design of neural and synapse building blocks at a level of abstraction that is scalable while not being overly simplistic to lose enriching neural information processing attributes. The class of liquid neural networks has been proposed to accommodate the aforementioned criteria [28].

Liquid neural networks (Fig. 1-1) are constructed by the interaction of leaky-integrator neural models [39] with the steady-state dynamics of a conductance-based nonlinear synapse model [37]. The model is a differentiable dynamical system with an input-dependent varying (i.e., liquid) time characteristic. As a result, they are called liquid time-constant networks (LTCs). Their outputs are computed by numerical differential equation (DE) solvers when described by ordinary DEs, and by continuous functions when described in closed-form [26] (Fig. 1-1B). These brain-inspired models are instances of continuous-time (CT) neural networks [14, 28] that can be trained via gradient descent in modern automatic differentiation frameworks.

Liquid networks exhibit stable and bounded behavior, yield superior expressivity within the family of CT neural models [14, 28], and give rise to improved performance on a wide range of time series prediction tasks compared to advanced recurrent neural network models [26]. In particular, a sparse network configura-

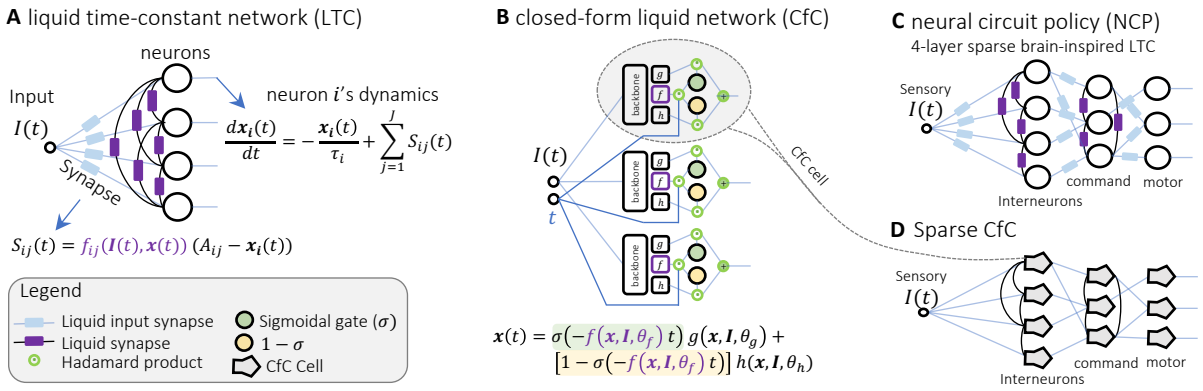

Figure 1-1: **Liquid Neural Networks. A.** schematic demonstration of a fully connected liquid time-constant (LTC) layer [28]. The dynamics of a single neuron i is given, where $\mathbf{x}_i(t)$ represent the state of the neuron i, and τ represents the neuronal time-constant. Synapses are given by a nonlinear function, where $f(.)$ represents a sigmoidal nonlinearity, A_ij is a reversal potential parameter for each synapse from neuron j to i. **B.** Schematic representation of a closed-form liquid network (CfC) together with its state equation. This continuous-time representation consists of two sigmoidal gating mechanisms, σ and $1-\sigma$ with their activity being regulated by the nonlinear neural layer f. f acts as the *liquid time-constant* for CfCs. These gates control two other nonlinear layers g and h to create the state of the system. **C.** A schematic representation of a neural circuit policy (NCP) [41]. The network architecture is inspired by the neural circuits of the nematode C. elegans. It is constructed by 4 sparsely connected LTC layers called: sensory neurons, interneurons, command neurons and interneurons. **D.** Sparse CfC. A sparse neural circuit with the same architectural motifs as NCPs this time with CfC neural dynamics.

tion composed of less than two dozen LTC neurons supplied with convolutional heads showed great promise in learning, end-to-end, to map high-dimensional visual input stream of pixels to robust control decisions [41]. These liquid network instances are called *Neural circuit policies* (NCPs), as their 4-layered network structure is inspired by the neural circuits of the nervous system of the nematode *C. elegans* [61], as illustrated in Fig. 1-1C and 1-1D.

The key to their robust performance under distribution shifts is their ability to dynamically capture the true causal structure of their given task [72]. This can be shown analytically to the fact that LTCs are dynamic causal models [22, 72], a framework through which models can account for internal and external interventions with independent mechanisms. Causality focuses the attention of LTCs onto the task rather than the context of the task and for this reason, in this paper we hypothesize and show that tasks learned in one environment can be transferred to different environments for LTC networks where other models fail. For instance, consider a drone navigation scenario, where a neural network agent is trained to fly the drone to a static object in a visually challenging environment. Liquid networks are expected to learn the fly-to-target task, by extracting this causal relationship from data and ignoring irrelevant concepts to generalize well.

In this work, we seek to determine the extent of the generalization benefits of liquid neural networks when applied to a navigation task. We empirically assess this robustness property of liquid networks in a large series of in and out-of distribution end-to-end navigation tasks in challenging environments. To this end, we build and test liquid neural control agents for autonomous drone navigation tasks (fly-to-target). We first hand-collected a high-variance expert dataset consisting of training examples in which humans solved the task. We then supplemented this dataset with closed-loop augmentation, repeatedly cropping an image of the target to create a sequence in which the drone appears to center the target in frame and move closer to the target. We then combined the two datasets and trained agents using imitation learning to mimic the actions in both datasets, relying on the human-collected dataset to teach the drone geospatial frame transformations

and relying on the synthetic dataset to teach the drone task understanding.

We then explore the generalization capabilities of different neural architectures by running them online in closed-loop in new environments with a drastic change of scenery, weather conditions, and other natural adversaries. After observing real-world agent behavior, we attempt to gain insight into the causal structures of different neural architectures by performing offline analysis.

We compared the results of liquid network agents against conventional RNNs and other continuous-time baselines. The superior expressive power of liquid networks and inherent causal structure helped them perform better than their discrete and CT counterparts in both the training environment and also in unseen environments. The intrinsic causality that comes with the liquid architecture enables zero-shot domain transfer with high task performance even in drastically different environments in the training environment, a feat that is much harder for other networks. One indicator why liquid networks exhibit superior generalization can be found in the saliency maps of the different CNN image backbones preceding the recurrent architectures. In liquid networks, the CNN feature is almost solely affected by the target region while the other networks focus on spurious features like bright spots and edges. Because the liquid networks are causal structures, their attention map focuses on the target and not on the surrounding environment. As a result of the surrounding environment being less important to liquid networks, when the task is moved to another environment, they are able to generalize better by focusing more attention on the target.

Specifically, we conduct online testing in four environments, with each environment getting successively more challenging and further from the training distribution. We first test network performance in the training environment. Afterwards, the drone is moved 45m away from the target, over 4x further than in any training data, and the networks attempt to locate the chair in the training environment from extreme range. We then perform zero-shot domain transfer experiments to evaluate network understanding, testing the networks on MIT campus. The drone must first navigate on a piece of lawn near CSAIL set against an urban backdrop.

For the final, hardest test, the drone is placed on a patio with heavy glare and no greenery or objects found in the training environment. To further complicate the task, the target chair is placed among many distractor chairs, some of which are the same color as the target.

After recording online testing results, we perform offline tests to understand causality and the underlying reasoning behind agent behavior. Ideally, robust networks would produce the same control outputs if environmental factors such as lighting and background were changed, but the target remained in the same position. To measure robustness, we simulate changing environmental conditions by perturbing input image sequences and measuring the network output perturbations. We also observe qualitative differences in the VisualBackProp saliency maps for different networks. We first compare saliency maps among the different neural architectures that underwent live testing. Then, we gauge inherent task affinity by observing the effect of sequence augmentation on network saliency maps in a variety of environments.

In this thesis, Chapter 2 addresses related work, including prior advancements in imitation learning for developing control policies for aerial vehicles along with the development of liquid neural networks, including empiral results and causality studies. Chapter 3 provides background information regarding liquid networks, including mechanisms and semantics. Chapter 4 describes preparation and methodology required to conduct the experiments, while Chapter 5 provides detailed descriptions of the online and offline experiments conducted and presents their results. Chapter 6 provides discussion regarding the work, and presents ideas for possible extensions for the project. Finally, Appendix A provides information about how to obtain all the materials used in creating this paper, including our publicly available datasets and code.

Chapter 2

Related Work

2.1 Imitation Learning

Imitation learning is an established procedure for training policies in autonomous agents [18] in which an agent is trained in a supervised manner. Imitation learning's lack of environment interaction makes it very dissimilar to reinforcement learning (RL), in which an agent repeatedly interacts with the environment without any known correct action to maximize a reward signal. Imitation learning teaches an agent to perform a task from only demonstrations of the task, with no external feedback. We choose to use pure imitation learning in this paper because it is extremely sample-efficient and requires no expert supervision during training. However, as a result, imitation learning techniques perform poorly OOD in closed-loop testing [62]. This occurs because of both issues associated with policies operating in closed-loop like policy stationarity as well as more fundemental properties of neural networks. For example, neural networks are prone to learning non-robust features that don't carry over between tasks [31], making them perform differently in train and test environments.

One way to improve generalization is to utilize hybrid techniques that introduce expert demonstrations in the midst of training or perform further interactions with the environment after receiving the expert dataset [57, 29, 71, 49]. Another way to guide networks is to automate the data collection process in some

manner to allow for larger training dataset size [25, 45]. Imitation learning is in part challenging because of the difficulty of communicating the intent of the task from demonstrations in scenarios, especially when testing scenarios look different from the training data. While intent can be communicated by performing 3D registration of training examples into test samples [64] or an explicit intent prediction network [18], these systematic process changes add train/test overhead. Another approach to communicating the task is to explicitly compare behaviors [15] or perform goal-conditioned RL in which the goal state is explicitly given to the policy [35, 34]. Another way to improve the sample efficiency of pure RL is to employ model-predictive control to reduce the complexity the trained policy must learn. A dynamics model can be learned to enable models to explicitly predict future input states [48, 21], enabling planning and reasoning to seek the best possible outcome by imagining different actions and states. By executing techniques such as receding horizon control [46] and cross-entropy [17], models can plan long-term trajectories without having to exhaustively explore the space of possible futures.

In contrast to the methods listed above, we sought to enable generalization without having to rely on any RL or on-policy evaluations during training. By targeting the policy networks themselves as opposed to the training procedure, we can train robust agents from just demonstrations without any additional training interventions overhead that requires collecting additional data or human input. Relying on the liquid architecture as the main factor in enabling generalization also allows predicting actions using one forward-pass of a neural network, not requiring a model to be searched through or the goal state to be known ahead of time. We propose a technique to improve network robustness that requires little additional information or compute at train and test time, enabling effortless task performance improvement and generalization.

2.2 Applications in Visual Navigation

Robotic visual navigation is an established problem with a variety of solutions. Classical approaches to the task relied on chaining together solutions for independent perception challenges like localization and planning. Force fields [9] and occupancy grids [10, 11] both involve creating maps, which can then be used to localize a robot. Alternatively, optical-flow [60] based techniques infer position using frame-to-frame deltas without explicit map construction [69]. These localizations can then be fed to planners [40] to generate control actions.

More recently, the success of deep RL [65] has promoted more end-to-end solutions to navigation tasks. Many-works have focused on improvements to RL techniques to solve different facets of visual navigation tasks, such as hierarchical RL [4, 38] to modularize behaviors in case of sparse rewards or multi-task RL [59, 47] to enable domain transfer.

Imitation learning has also been explored an approach for solving robotic visual navigation tasks similar to the task we investigate in this work. Many papers explore the idea of zero-shot domain transfer, with [51] focusing on utilizing demonstrations from different vehicles, while [52] involves the agent randomly interacting with the environment before seeing a sequence of images depicting the desired behavior. In a similar vein, [73] aims to solve tasks with only visual sequences, not providing any teleoperation or controls. [18] takes a meta-learning approach, explicitly defines a set of similar tasks for training and presents one demo of a new task at test-time.

Our work aims to provide end-to-end zero-shot domain transfer in a slightly different manner. While we do provide exact demonstrations containing both the inputs and outputs we want to emulate, we aim to transfer the same task to different environments without explicitly training on multiple environments or specifying separate tasks in a meta-learning style.

2.3 Testing Liquid Networks

Existing work has investigated the ability of sparse liquid networks to navigate autonomous simulated aerial [72] and real ground [41] vehicles. These works show that liquid networks are capable of much more robust navigation than their advanced deep learning counterparts, in a large series of behavioral-cloning experiments within their training distribution. In this work, we aim to extend the experiments in [72] by moving them to the real world and to OOD scenarios. To our knowledge, we are the first paper to test liquid networks on real-world aerial platforms. In our work, we overcome sim2real [68, 2] complications and perform extensive generalization and robustness experiments difficult to replicate in simulation.

Chapter 3

Model Background

This chapter provides background information on the mechanisms and mathematics behind liquid networks, including LTCs, NCPs, and CFCs.

3.1 Fundamentals of Continuous-Time Recurrent Neural Networks

Continuous-time (CT) RNNs are a class of recurrent models in which the state update is not dictated by a set of discrete updates as in conventional RNNs like LSTMs or GRUs, but is instead governed by an ordinary differential equation (ODE). In CT RNNs, the *derivative* of the state is expressed as a function of the input and previous state as opposed to the *value* being set explicitly.

Let x represent RNN state, I represent network input, and y represent network output. Conventional network dynamics are shown in equation 3.1 and contrasted against CT RNNs, which have state updates in the form of equation 3.2. Note the output of both types of networks is computed in a similar way, but the way that inputs are computed changes.

$$\begin{aligned} x(t) &= f(x(t-1), I(t)) \\ y(t) &= g(x(t), I(t)) \end{aligned} \tag{3.1}$$

$$\begin{aligned} \frac{dx}{dt} &= f(x(t), I(t)) \\ y(t) &= g(x(t), I(t)) \end{aligned} \tag{3.2}$$

Note that the output of a CTRNN cannot be computed with a single explicit function evaluation like a conventional RNN, and forward passes must be instead calculated using numerical integration, an implicit process that solves a system of differential equations given initial conditions by discretizing the continuous-time process into a series of many small timesteps, as in equation 3.3.

$$\frac{dx(t)}{dt} \approx \frac{x(t+\delta t) - x(t)}{\delta t} \approx f(x(t), t, \theta) \implies x(t+\delta t) = x(t) + \delta f(x(t), t, \theta) \tag{3.3}$$

CTRNN's backward pass can be done through the ODE solver [42] by simply viewing the solver as a single composite function, and chaining derivatives from each function. This process is applied recursively for each timestep to yield a process caused backpropagation through time (BPTT). Alternatively, a technique known as the *adjoint method* [55] solves an augmented ODE backwards in time to get the gradients. BPTT is generally more memory-intensive than the adjoint method, which has a fixed memory cost per-layer. However, the adjoint method introduces more numerical error.

A comparison of a CTRNN (NCP), and a conventional discrete-time RNN can be found in Fig. 3-1

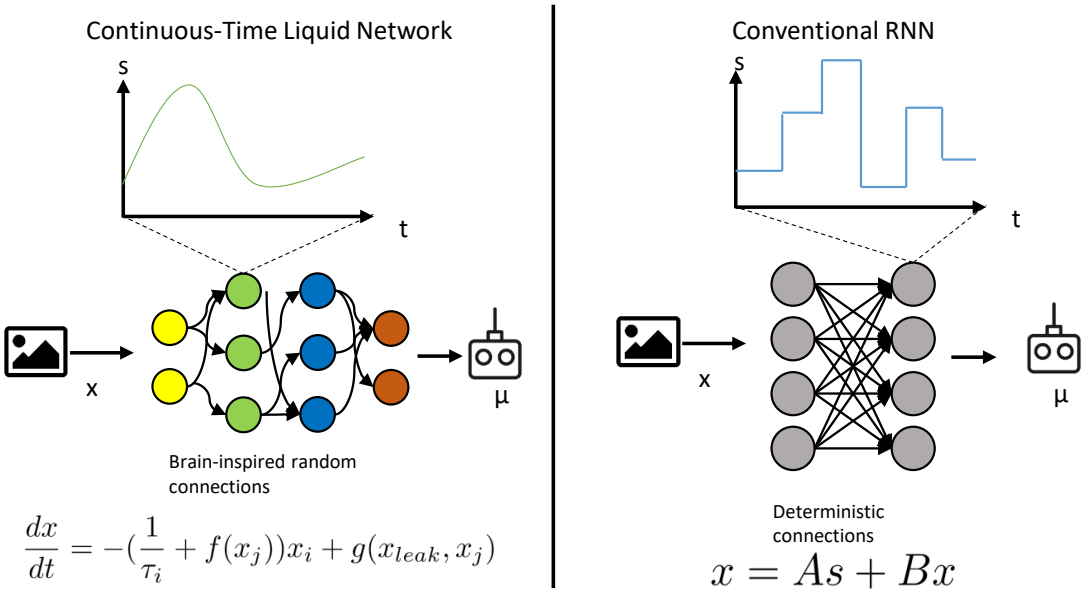

Figure 3-1: **NCP vs. Conventional RNN:** Depiction of NCP (left), a continuous-time neural network, compared to a standard discrete-time network (right). The hidden state of a CTRNN is continuous, while a conventional RNN's state is only updated at discrete periodic intervals.

3.2 LTC Equations

A Liquid Time-Constant Network (LTC) is a particular type of CTRNN that can adjust its time constant, a constant that affects the speed of network updates by scaling the magnitude of the derivative. The dynamics of LTCs are given by equation 3.4 (θ and A represent parameters of the system), adapted from [28].

$$\frac{dx(t)}{dt} = -[\frac{1}{\tau} + f(x(t), I(t), t, \theta)]x(t) + f(x(t), I(t), t, \theta)A \tag{3.4}$$

Note that the multiplier of x determines the time constant of the system governed by the ODE. If the time constant were fixed, the system would become a linear system. However, in the LTC, because the nominal time constant $1/\tau$ has the nonlinear liquid $f(x(t), I(t), t, \theta)$ added to it, the effective time constant of system, the multiplier of $x(t)$ can change as a function of the state and the input.

The fact that the input I appears in the nonlinearity f multiplying x couples the

input and system dynamics, and the fact that the state x itself appears in the nonlinearity f multiplying x couples the state and system dynamics. This nonlinearity f is implemented as a neural network, and it acts as an input and state-dependent varying time constant. (Note that by rearranging terms, the actual time constant of the system can be given by equation 3.5)

$$\frac{\tau}{1 + \tau f(x(t), I(t), t, \theta)} \tag{3.5}$$

A variable time constant could be useful when driving a car, for example, as there are multiple modes when driving that require different input sensitivity. For example, it would be better for the time constant to be higher while turning than when going straight. A high time constant while turning could allow the network to make faster updates and forget old information, while a low time constant when driving straight could prevent it from making overly rapid motions.

LTCs exhibit provably stable and bounded behavior, have better expressivity, or ability to compute any function, than other models, and yield improved task performance on long-term dependency sequence modeling and control tasks [28].

3.3 NCP Wiring

While the original LTC paper [28] used fully-connected layers of LTCs, the authors found success with an alternative 4-layer arrangement of LTC neurons [41] inspired by the brain of of the nematode *C. elegans* [61]. Neurons in the sensory, inter-neuron, command, and motor layers are connected with randomly-selected synapses with randomly determined synaptic polarities allowing data flow between separate layers. Additionally, recurrent connections are randomly inserted between neurons in the command layer, including self-connections. The NCP wiring is depicted in figure 1-1C.

The dynamics of an NCP neuron's state are given by equation 3.6, where x_j represents the state of an input synapse, w_j is the input weight on the synapse

connecting neuron i to j, C_i is the membrane capacitance, and E_{ij} is the randomly-selected polarity of the synapse.

$$\frac{dx_i(t)}{dt} = -[\frac{1}{\tau_i} + \frac{w_{ij}}{C_i}\sigma_i(x_j(t))]x_i(t) + (k + \frac{w_{ij}}{C_i}\sigma_i(x_j(t)E_{ij}) \tag{3.6}$$

Notice that the NCP has the same variable time constant that depends on the input that it receives from other neurons, and therefore has the state-dynamics coupling found in the LTC.

NCPs are found to perform exceptionally well on end-to-end lane-following task even when exposed to noise, and have CNN saliency maps that attend to more relevant task features than other networks [41].

3.4 Closed-form Solutions

Closed-form Continuous-depth Networks (CfC) aim to make liquid networks more performant by solving the network's dynamical system in closed form, eliminating the need for ODE solvers. They are able to run inference for a single time step in time proportional to the number of units, with no dependence on the previous number of time steps or the order of the ODE solver, without compromising expressivity compared to LTCs and NCPs.

By solving the differential equation dictated by the equation of an LTC, and approximating the solution of the integral $\int_0^t f(I(s))ds$ by assuming the inputs to the system I are piecewise constant, CfCs can solve a closed form expression that has provable tightness bounds to the actual closed form solutions.

A closed-form solution to the LTC equations allows a 150x accuracy-per-compute improvement [26] that enables larger scale models and faster inference and training times, without a substantial loss of performance.

3.5 Causality of Liquid Networks

It can be shown that LTCs belong to a class of networks known as Dynamic Causal Models (DCM). DCMs are models that can be expressed in the form given in equation 3.7, where Matrix A represents an internal coupling of the system, Matrix B controls the effect of inputs on the connections between network nodes, and Matrix C represents the effect of external inputs on the system.

$$\frac{dx}{dt} = (A + I(t)B)x(t) + CI(t) \tag{3.7}$$

DCMs can capture both internal and external interventions, or causes, on the system. It is possible to rewrite LTCs into a DCM framework if f is a Lipschitz-continuous nonlinearity. When LTCs are transformed into a DCM, the external intervention coefficients C are given in equation 3.9 and the internal intervention coefficients B are given in equation 3.8. Because LTCs are DCMs, internal and external interventions can help LTCs learn the causal mapping underlying a task [72].

$$W(1 - f^2) \odot A \tag{3.8}$$

$$W(f^2 - 1) \odot [2W_\tau f \odot (A - x) + 1] \tag{3.9}$$

Because of these causal properties, liquid networks that derive from LTCs, like the NCP, can capture the causal structure of visual navigation tasks in simulation. In the experiments in [72], the saliency maps of the NCP are shown to be the only ones out of the networks tested that put focus on the target object. This causal reasoning allows the NCP to perform better than its counterparts [72] across of a variety of tasks in different simulated environments.

Chapter 4

Method

In this section, we detail our experimental setup and preparations required for experiments.

Training Procedure

The following section details preparation of the data and hyperparameters used for training the onboard models.

Data Collection

The machine learning framework we adopt is that of imitation learning, a choice motivated by its sample efficiency for real-world deployment [56]. We wish to demonstrate empirically the ability of deep models to capture the causal structure of the fly-to-target tasks, from limited amounts of data collected from the natural environment. We also aim to assess the transferability of learning regarding the ability of neural networks to perform in closed-loop on real world applications. Both these objectives are best attended to via imitation learning as opposed to expensive techniques such as reinforcement learning (RL) [67].

The training data was collected in a wooded environment as illustrated in Fig. 4-1A and Fig. 4-2A. It consists of roughly 100 expert demonstration rounds performed over multiple seasons. The year-round acquisition introduces diversity

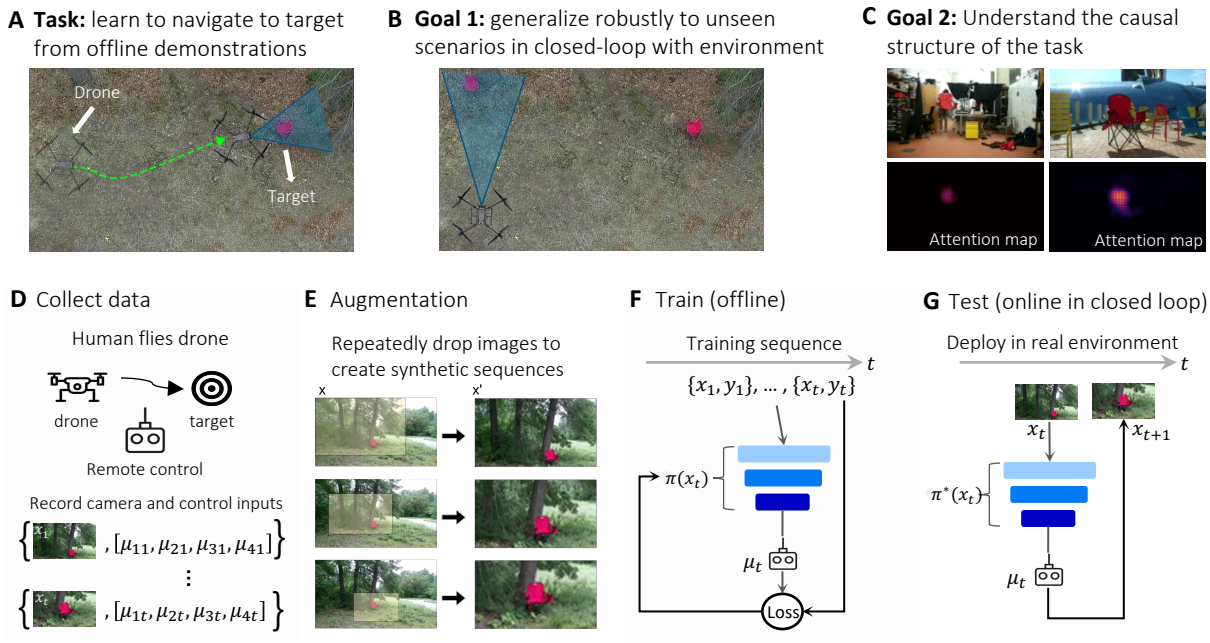

Figure 4-1: **End to end learning setup. A.** Policies were trained to solve the following task: Using only images taken from an onboard camera, navigate the drone from its starting location to a target 10m away, keeping the object in frame throughout. **B.** One goal of the work was to explore the generalization capability of various neural architectures by adapting the task to previously unseen environments. **C.** Another goal was to understand the causal mechanisms underlying different networks' behaviors while completing the task i.e. by visualizing the network input saliency map. **D.** The training process starts by hand-collecting human expert trajectories, recording camera observations and human expert actions. **E.** To further convey the task, synthetic sequences are then generated by taking images collected during human flights and repeatedly cropping them, creating the appearance of a video sequence in which the drone flies up to the target and centers it in frame. **F.** The networks are trained offline on the collected expert sequences using a supervised behavior cloning MSE loss. **G.** We then deploy trained networks on the drone in online testing and observe task performance under heavy distributional shift, varying lighting, starting location, wind, background setting, and more.

into the training data, with runs containing backgrounds rich in green leaves in the spring, others containing complex structure and palette of dead leaves in the fall, and finally data containing the striking contrast of dark leafless trees against snow in the winter as depicted in Fig. 4-2. Each run consists in successively flying towards five objects of different sizes and colors; a red bag, a red camping chair, an orange storage box, a black case, and a burgundy RC car (See Fig. 4-2A and 4-2C). At each target of interest, the quadrotor is maintained at a distance of around one

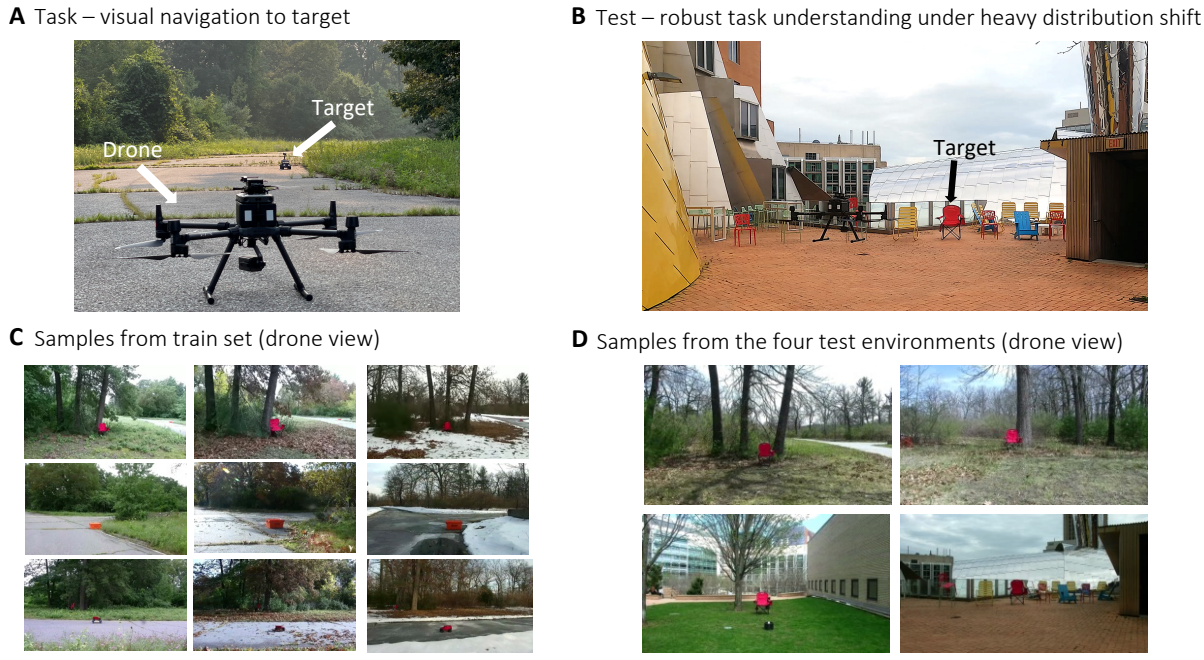

Figure 4-2: **Sample frames: A.** Third-person view of the quadrotor and one of the targets in the woods where data is collected. **B.** Third-person view of the quadrotor during testing on an urban campus patio, with multiple adversary objects dispersed around the target camping chair. **C.** Training data frame samples from the quadrotor onboard camera containing various targets (camping chair, storage box, and RC car from top to bottom) and taken during different seasons (summer, fall, and winter from left to right). **D.** Test data frame samples from the quadrotor onboard camera against each of the four test backgrounds: Training Woods (top left), Alternative Woods (top right), Urban Lawn (bottom left), and Urban Patio (bottom right).

to two meters for a few seconds.

By design, we allow for vagueness in execution and lack of high repeatability in the training runs in order to diversify the observations and actions distributions (Fig. 4-1D). The input data to a recurrent neural network (RNN) π consists of sequences of RGB images collected by the front-facing gimbal-stabilized camera. Each frame x_i is labeled with the instantaneous quadrotor velocities and yaw angular rate in the quadrotor body frame obtained from the onboard Inertial Measurement Unit (IMU), more specifically y_i, a 4 dimensional vector containing the x (pitch), y (roll), and z (throttle) velocities in body-frame in addition to the quadrotor yaw rate. The Imitation Learning process, depicted in Fig. 4-1, trains the net-

work π to output a control command μ_t given an input image x_t that minimizes a loss function on the processed training data. In addition to performing RGB image augmentation on the training data (brightness, contrast and saturation), we generate synthetic image and control label sequences from still training data frames undergoing repeated cropping. This closed-loop augmentation technique allows the networks to learn how to generate controls to recover from unseen corner cases and reduce compounding error [57, 12, 44, 1]. We refer the reader to subsection Data Augmentation for more details.

We train various RNN architectures that are supplied with a compact convolutional head as perception module, on the collected sequences of images using a Behavior-Cloning (BC) Mean Squared Error (MSE) loss, without any additional on-policy corrections or Reinforcement Learning. In this setting, networks are encouraged to mimic the expert demonstrations by outputting the same velocity and yaw rates commanded by the expert pilot during demonstration when given the same input image. Each policy network consists of a stateless Convolutional Neural Network (CNN) head to process raw RGB image data followed by an RNN that takes the image embedding as its input and generates four dimensional control outputs. Although models share the same CNN head architecture, we shed light on the fact that the CNN weights depend on the downstream network through back-propagation during training and, thus, are computed independently for each different architecture.

Seven neural network architectures are evaluated in this work: long short-term memory (LSTM) [30] which is a discretized RNN architecture equipped with gating mechanisms; a variety of continuous-time architectures, including, ordinary differential equation (ODE) RNNs [14] and gated recurrent unit (GRU) ODEs [58], and as well as a range of brain-inspired advanced RNNs depicted in Fig. 1-1, LTCs [28], NCPs [41], CfCs [26], and Sparse-CfC models. Additionally, to be inclusive of all range of possible models for sequential decision-making problems, we also trained Temporal Convolutional Network (TCNs) [3] to assess the robustness of non-recurrent modules in fly-to-target tasks.

Data Preparation

The training runs were originally collected as long sequences in which the drone moved between all five targets. We worried that the sequences searching for the next target and traveling between them could lead to possible ambiguity regarding the drone task. To better encode the task for the networks, for each training run of five targets, we spliced the approach to each of the five targets into five separate training sequences and added these new sliced sequences to the training data.

Data Augmentation

In order to increase the networks' robustness and generalization capabilities, we performed various image augmentations during training. The first set of augmentations we applied were a random brightness shift with brightness offsets selected uniformly at random ranging from 0-0.4x of the max pixel brightness, a random contrast shift with a contrast factor between 0.6-1.4, and a random saturation multiplication between 0.6-1.4. Parameters for the aforementioned image augmentations were fixed within a sequence but randomized between sequences so that two images within the same training sequence would have identical brightness, contrast, and saturation offsets but two images in different sequences would have different offsets. Finally, each image had Gaussian random noise with mean 0 and standard deviation 0.05 added.

Additionally, in order to better convey the task, we performed closed-loop augmentation by generating synthetic image and control sequences and adding them to the training set. To generate these sequences, we took a single image with the target present and repeatedly cropped the image. Over the duration of the synthetic sequence, we moved the center of the crop from the edge of the image to the center of the target, causing the target to appear to move from the edge of the frame to the center. Additionally, we shrunk the size of the cropped window while upscaling all of the cropped images to the same size, causing the target to appear to grow bigger. Together, these image procedures simulated moving the

drone to center the target while moving closer to the target to make it occupy more of the drone's view. We then inserted still frames of the centered target with a commanded velocity of zero for 20-35% of the total sequence length. This pause sought to teach the drone to pause after centering the target. Alg. 1 describes the synthetic sequence generation steps, while Fig. 4-1E visualizes the successive crops calculated by the algorithm and their resulting output frames.

To generate the velocity controls for the synthetic sequence, we generated yaw commands equal to the horizontal distance in pixels between the target and the center of the frame times 0.01, and likewise generated throttle (up-down) commands equal to the vertical target offset times 0.01. We generated pitch (forward-backward) commands in the synthetic data proportional to the size of the cropped window. This technique increased the size of the training set by over 2.5x, using a relatively small sample of images in which the target was present.

The initial position of the target within the frame was chosen randomly such that the x (horizontal) offset of the chair was between 10 and 70 pixels and the y offset was between 5 and 40 pixels. (The size of the input images was 144x256 pixels) Synthetic sequences were later balanced such that the mean control signal across all generated data points was zero for the throttle and yaw channels. The sequence length was also randomized to be between 120 to 250 frames to encourage robustness to external forces or varying update frequencies.

Note that although we used these augmented sequences for training, we did *not* include the closed-loop synthetic sequences in the validation set.

We also tried an alternative augmentation strategy in which sequences would be permuted versions of actual training sequences where the video frames and controls would be offset slightly by an amount that decayed over the course of training. However, we found that this method yielded 0% success rate for all architectures.

Algorithm 1 Synthetic Sequence Generation

Require: Image dataset D, dataset size param k_l, max zoom k_z, control gains k_p, k_t, k_y

Initialize $A := \emptyset$
for $i = 1$ **to** $k_l \cdot |D|$ **do**
  Initialize $A_i = \emptyset$
  Randomly sample sequence augmentation params: $T, o \sim U; b \sim D_x$
  for $t = 1$ **to** T **do**
    Compute sequence completion fraction $f := \dfrac{T - t}{T}$
    Linearly interpolate frame offset $o_t := o \cdot f$and frame zoom $z_t := k_z \cdot f$
    Let $c :=$ crop(image=b, offset=o_t, zoom=z_t)
    Let $v :=$ [pitch= $k_p z_t$, roll= 0, throttle= $k_t o_{ty}$, yaw= $k_y o_{tx}$
    Add sample t to $A_i \leftarrow A_i \cup (c, v)$
  end for
  Update dataset $A \leftarrow A \cup A_i$
end for
return A

Hyperparameter Tuning/Network Architectures

For each network architecture, we use Tree-structured Parzen estimators (TPE) [5, 6] for hyperparameter tuning. TPE finds both universal hyperparameters such as learning rate and lr decay rate, and model-specific hyperparameters such as backbone unit count or connection seed that maximize each architecture's performance. The set of parameters that led to the lowest train+validation loss over the 40 episodes were selected for the final tested model.

TPE, a sequential model-based optimization algorithm, was responsible for sampling new hyperparameters to try across different episodes. After 3 trials, episodes that had a higher median train+validation loss on the 10th epoch were pruned early and did not run to completion. Hyperparameter optimization was implemented using the Optuna library. Note that not all parameters were found via hyperparameter tuning; some parameters were hand-selected to shrink the dimensionality of the search space.

The best hyperparameters found for each model architecture and used during testing are listed below in Table 4.1.

For each chosen hyperparameter configuration, the number of trainable param-

eters in the corresponding model for every neural architecture are listed below in Table 4.2.

Each network tested was prefixed by a simple CNN backbone for processing incoming images. The 128-dimensional CNN features were then fed to the recurrent unit that predicted control outputs. The shared CNN architecture is pictured in Table 4.3.

Fine-tuning

In order to simultaneously learn the geospatial constructs present in the long, uncut sequences *and* the task-focused controls present in the cut and synthetic sequences, we fine-tuned a model trained on the long sequences with sequences featuring only one target.

The starting checkpoint was trained on the original uncut sequences, all sliced sequences, and synthetic data corresponding to all targets. We then regenerated synthetic data with new randomized offsets for one target and fine-tuned on sliced sequences containing the one target and new synthetic augmented sequences that featured that target. Training parameters are listed in Table 4.4.

Checkpoint Selection

After hyperparameter optimization was completed, each model was trained 5 times with the best set of hyperparameters. Among these 5 training runs the models with the best overall training and validation loss were selected for testing on the drone. Alg. 2 summarizes the high-level learning procedure.

Training and validation loss curves for the runs with the overall lowest training loss are shown in Fig. 4-3. During training, we observed that the validation loss did not decrease substantially after the first 10% of training. This lack of improvement could be attributed to overfitting on the training set or the disparity between the train and validation sets. Because the synthetic sequences are used for train loss but not validation loss, policies that achieve low loss on the consistently-generated

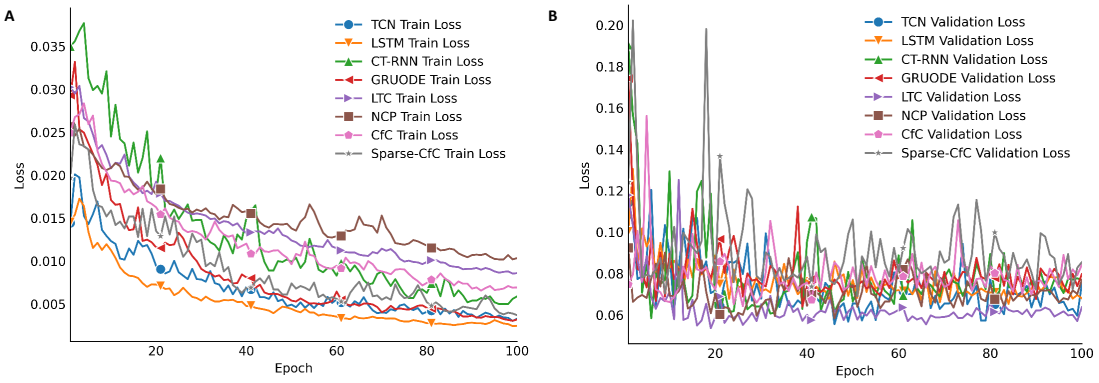

Figure 4-3: **Losses: A.** Training loss plots for all model types (lower is better). Observe that LSTM has the best training loss, but poor test-time performance, implying training loss is a poor indicator of task performance.. **B.** Validation loss plots for all models. Observe that the general trend for validation loss is relatively horizontal. We empirically observe that the overfit models with the lowest train loss perform best on the task while checkpoints with the lowest validation loss fail.

synthetic sequences may have high loss on the high-variability human-collected validation set.

During testing, it was discovered that the checkpoints with the lowest validation loss often performed poorly across a variety of neural architectures even in the train environment. We find that that good performance on the validation set does not correlate with good real-world performance and that more training improves online test performance despite the appearance of overfitting. This can likely be attributed to the variance and variability in task-solving strategy in the hand-collected data. This variance, along with the exclusion of the synthetic sequences, makes some strategies that perform well in practice and on the synthetic sequences achieve poor validation performance. As a result, all results were collected using models that had the lowest training loss across all 5 training runs.

Platform Setup

We evaluated the proposed RNN architectures on a commercially available quadcopter with custom software running onboard. The system features a forward-facing gimbal-mounted camera and a GPU to run onboard inference. We use ROS

Algorithm 2 : Learning Procedure: Generate networks for online testing

Require: Expert training dataset $D = [(x_1, [y_{11}, y_{12}, y_{13}, y_{14}]), \ldots (x_t, [y_{t1}, y_{t2}, y_{t3}, y_{t4}])]$

call Algorithm 1 on D to generate synthetic dataset: A
Aggregate datasets $D \leftarrow D \cup A$
call Algorithm 3 on D to train policy π
Extract sequences containing specifc target $\hat{D} :=$ subsample(D, target=t)
call Algorithm 1 on $\hat{D}$ to generate new synthetic dataset $\hat{A}$
Aggregate datasets $\hat{D} \leftarrow \hat{D} \leftarrow \hat{A}$
call Algorithm 3 on $\hat{D}$ 5 times
return policy trained on $\hat{D}$ with best overall train loss: $\hat{\pi}$

handle communication between the RNN inference code and the lower-level drone flight controller.

Drone Hardware

A DJI M300 RTK quadcopter was used for data collection and evaluation of policy performance. The M300 is an enterprise-grade drone that can be flown by manual control in its default configuration. It also interfaces with the DJI Manifold 2 companion computer, enabling programmatic control of the drone. The companion computer includes an NVIDIA Jetson TX2 which has a CPU and GPU. The DJI Onboard SDK [1] and its associated ROS wrapper [2] provide an interface for giving the drone's low-level flight controller desired linear and angular velocity setpoints. The flight controller is a black box not modifiable by the end user, but it controls the four rotor speeds to track the velocities specified by the TX2 computer.

A Zenmuse Z30 camera and gimbal are mounted on the underside of the drone, pointed forward. The gimbal compensates for the drone's current orientation, such that roll or pitch of the drone do not result in roll or pitch of the camera image. The gimbal follows the drone's current yaw such that the camera is always pointed forward. Images from the camera are available to the companion computer via the SDK.

[1] `https://developer.dji.com/onboard-sdk/`
[2] https://github.com/dji-sdk/Onboard-SDK-ROS

Algorithm 3 Training Algorithm

Require: Dataset $D = (X, Y)$, sequence length l, stride s, batch size b
  Initialize sliced dataset $\hat{D} \coloneqq \emptyset$
  for $i = 1$ **to** $|D|$ **do**
    Let sequence start $m \coloneqq 0$
    while $m + l \leq |D_i^x|$ **do**
      Slice sequence into subsequence $d \coloneqq D_i[m : m + l : s]$
      Update sequence start $m \coloneqq m + l$
      Append sequence $\hat{D} \leftarrow \hat{D} \cup d$
    end while
  end for
  Split $\hat{D}$ into train set $\hat{D}^t$ and validation set $\hat{D}^v$
  Randomly initialize weights w
  Note best loss $l_{\text{best}} \coloneqq \infty$
  for $e = 1 \ldots \text{epoch}_m$ **do**
    for batch (x, y) of size b randomly drawn from $\hat{D}^t$ **do**
      Compute loss $L \coloneqq \text{MSE}(f(x, w), y)$
      Perform weight update $w \coloneqq w - \alpha_{\text{ADAM}} \dfrac{\partial L}{\partial w}$
    end for
    Compute validation loss l_v on $\hat{D}^v$
    if $l_e \leq l_{\text{best}}$ **then**
      $l_{\text{best}} \coloneqq l_e$
      $w_{\text{best}} \coloneqq w$
    end if
  end for
  return w_{best}

System Design

We implemented our software control system in ROS. We wrote a ROS node that receives images from the onboard camera, runs inference with the RNN controller to compute the new desired control based on the current image, and sends the generated control outputs to the low-level flight controller using the onboard ROS SDK. The node has 2 threads; one thread is responsible for processing incoming images and the other thread is responsible for running inference on the images and sending the resulting control signals to the flight controller. The TX2's GPU runs the RNN inference with GPU-accelerated Tensorflow 2.3.x. A depiction of the system design can be found in Fig. 4-4.

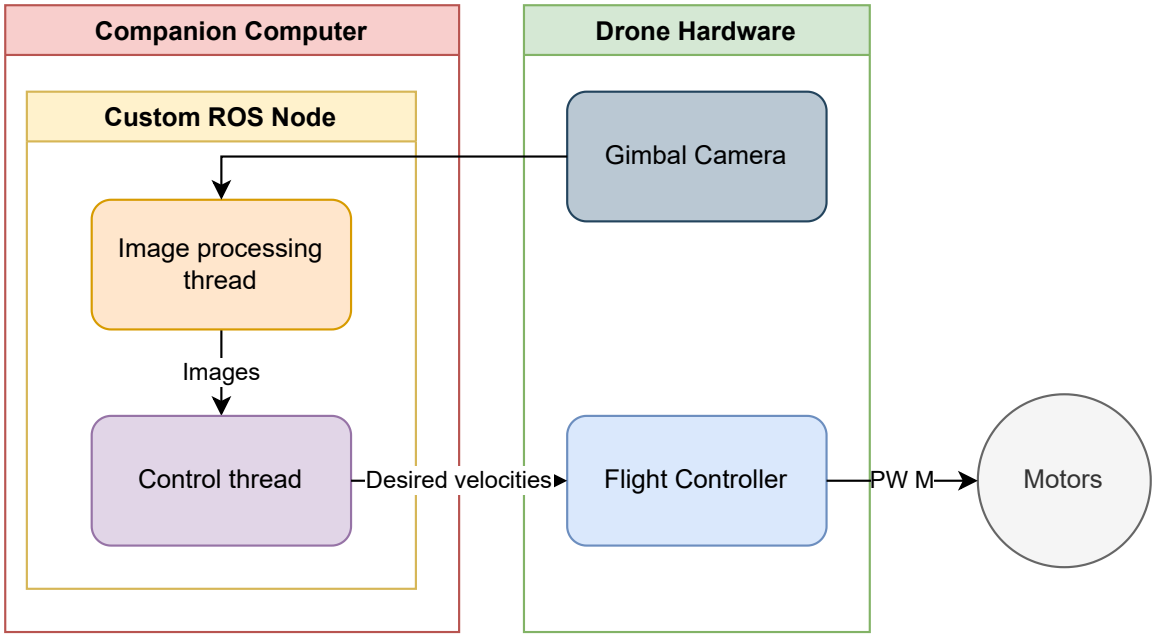

Figure 4-4: **System block diagram:** A custom ROS node processes gimbal camera images and runs inference on them in a separate thread. The resulting velocity commands are translated into motor thrusts by an onboard flight controller.

Analysis Methodology

Visual Backprop

A primary goal of this paper is to analyze task-understanding of the policy networks. We visually analyze the task-understanding of a policy network by examining how the policy's CNN encodes and extracts features from an input. Each of the policies share the same CNN head architecture, so differences in CNN weights between policies are representative of what the downstream architecture learns to extract from the environment.

Using the Visual BackProp algorithm, we use observations of the environment to compute attention mappings from the CNN head of the model architectures. Given an input image, pass it through each layer of the model's CNN head. The algorithm, shown in Alg. 4, is as follows: As an input is passed through the CNN, each of the feature maps is averaged across the channels and saved. Starting from the final layer, the averaged feature map is deconvolved to the size of the previous layer's averaged feature map. The deconvolved feature map is multiplied by the

previous layer's averaged feature map. This product is then passed to the previous layer. The deconvolution and multiplication is repeated iteratively, updating at each step until it reaches the first layer. The result is a feature map displaying locations in the source image to which the model attends the most attention. These maps are the saliency maps [8].

Algorithm 4 : Visual BackProp: Compute the saliency map of an input image for a policy's CNN

Require: Channel-Averaged Feature Maps $\overline{h_1}, \overline{h_2}, \ldots, \overline{h_N}$
  $s := \overline{h_N}$
  for $i = N - 1$ **to** 1 **do**
    $s := \overline{h_i} \odot$ deconvolution-to-size-of$(s, \overline{h_i})$
  end for
  return s

Table 4.1: **Hyperparameters:** Best hyperparameters for each neural architecture found via TPE sampling. Fixed parameters were not found through optimization, and were set ahead of time.

Param Name	Param Value
Connection seed	22224
Learning rate	$2.91x10^{-4}$
LR decay rate	0.97
Fixed (not tuned) parameters	
Inter neurons	18
Command neurons	12
Motor neurons	4
Sensory fanout	6
Recurrent command synapses	4
Motor fanin	6

(a) NCP Params

Param Name	Param Value
Forget bias	3.01
Backbone units	147
Backbone layers	2
Weight decay	$4.23x10^{-8}$
RNN size	193
Learning rate	$1.83x10^{-4}$
LR decay	0.91
Fixed (not tuned) parameters	
Backbone activation	silu
Backbone dropout	0.1

(b) CfC Params

Param Name	Param Value
RNN size	196
Learning rate	$4.79x10^{-4}$
LR decay	0.96

(c) Sparse-CfC Params: note connection seed and wiring were set to be same as NCP for fairness of evaluation

Param Name	Param Value
RNN size	252
Learning rate	$2.92x10^{-4}$
LR decay	0.85

(d) CTRNN Params

Param Name	Param Value
RNN size	198
Learning rate	$9.25x10^{-4}$
LR decay	0.93

(e) GRUODE Params

Param Name	Param Value
RNN size	69
Learning rate	$5.06x10^{-4}$
LR decay	0.85

(f) LTC Params

Param Name	Param Value
RNN size	193
Dropout	0.15
Learning rate	$2.90x10^{-4}$
LR decay	0.98

(g) LSTM Params

Param Name	Param Value
Number of filters	236
Kernel dilations	2
Kernel size	[1, 2, 4, 8, 16]
Dropout	0.15
Learning rate	$2.90x10^{-4}$
LR decay	0.98

(h) TCN Params

Table 4.2: **Number of parameters:** For each model, the number of trainable parameters in the recurrent part of network is listed (not including the CNN backbone, which is listed separately at the end of the table). Observe that the liquid architectures that performed the best, NCP and Sparse-CfC, achieve superior task performance and robustness to perturbation than the LSTM despite having over 10x fewer trainable parameters.

Model Type	Number of trainable parameters
TCN	157.8k
LSTM	248.6k
CT-RNN	96.3k
GRUODE	194.2k
LTC	55.0k
NCP	**22.4k**
CfC	183.3k
Sparse-CfC	**12.3k**
Shared CNN Backbone	103.7k

Table 4.3: **CNN Architecture:** Shared architecture of the CNN backbone that processed visual inputs for the different recurrent neural architectures.

Layer Type	Input Dim.	Filters	Kernel Size	Stride	Activation
2D Conv.	144x256x3	24	5x5	2	relu
2D Conv.	70x126x24	36	5x5	2	relu
2D Conv.	33x61x36	48	5x5	2	relu
2D Conv.	15x29x48	64	3x3	1	relu
2D Conv.	13x27x64	16	3x3	2	relu
Flatten + Fully Conn.	1248	N/A	N/A	N/A	linear

Table 4.4: **Training hyperparameters:** Miscellaneous parameters that were shared across all model types.

Param Name	Param Value
Initial train epochs	100
Fine tune epochs	100
Validation split	0.05
Optimizer	ADAM
Data shift	16
Data stride	1
Sequence length	64
Batch size	128

Chapter 5

Experiments and Results

We demonstrate the robust task-understanding capacity of brain-inspired liquid neural networks on a series of quadrotor closed-loop control experiments leading to their out-of-distribution generalization. The fly-to-target task consists of autonomously identifying a target of interest and performing the flight controls driving the quadrotor towards it using the onboard stabilized camera's sequences of RGB images as sole input (Fig. 4-1). We initially start the quadrotor approximately 10m away from the target and require that the policy guides the drone to within 2m of the target with the target centered in the camera frame. The objective is to learn to complete this task entirely from an offline dataset of expert demonstrations. The learned policies are then tested online in closed-loop within the training distribution as well as in drastically distinct settings. This experimental protocol allows for the principled assessment of performance and generalization capabilities of liquid networks compared to modern deep models [14, 58, 3].

Closed-loop (online) Testing

The fly-to-target task performance is evaluated in closed-loop with the environment, by running the neural networks onboard the drone. In this setting, the quadrotor is placed at a fixed distance from a target of interest seen in the training data. In our evaluation experiments, we use the red camping chair as the target for

collision avoidance purposes; due to its relatively large size it can occupy an important portion of the camera frame without the drone having to get dangerously close. The target is present in the initial field of view, and its position in the initial input frame is randomized but balanced over all runs (each network is tested on the same equal number of right, center and left starting positions at slightly different altitudes). For more information about the platform setup, see section Platform Setup.

A single testing protocol is repeated for a number of testing environments. We randomly position the quadrotor at a distance of about 10 meters from the target with the latter in the field of view of the onboard camera. We launch the closed-loop policy and observe whether the network can successfully guide the drone to the target. The test is repeated 20 times for each network. A success is accounted for when the network is able to both stabilize the drone in a radius of 2 meters and maintain the target in the center of the frame for 10 seconds. Failure cases are identified when the network's generated commands leading to an exit of the target from the range of view without possibility of recovery. We also include cases where the drone fails to reach the target in under 30 seconds to account for rare runs where the network generates commands indefinitely maintaining the drone in a stable position with the target in sight but with no intent of flying towards it.

We test the policies in four environments we call *Training Woods*, *Alternative Woods*, *Urban Lawn*, and *Urban Patio* (Fig. 4-2B and 4-2D). The first corresponds to the target in the woods respecting roughly thesame position and orientation distribution of that of the training set. We thus evaluate the performance of the networks on image sequences from the scenery used during training with different lighting conditions and wind profiles. Alternatively, we move the chair to a different spot with a different background in a neighboring part of the woods. This experiment evaluates the networks' generalization performance on input data with scenery distribution shifts, but maintains proximity to sequences seen during training. To truly evaluate robust task-understanding, we subsequently setup the zero-shot domain transfer experiments in a drastically dissimilar environment: a piece of lawn

on CSAIL MIT's campus. Although containing green grass and a couple of trees, the landscape is largely different from the woods as shown in Fig. 4-2A-D. Indeed, frames now contain buildings, windows, large reflective metallic structures, and the artificial contrast from the geometric shades they induce. We also mention that lighting conditions at different times of day as well varying wind levels add additional perturbations.

We finally push the networks' generalization capabilities to their limits in a completely out-of-distribution environment of a brick patio. In addition to the background including a number of man-made structures of different shapes, colors and reflectivity. We add an extra layer of complexity to this experiment by positioning a number of other chairs in the frame of different types colors (including red) and sizes. This ultimate test including real world adversaries requires robustness in the face of extremely heavy distribution shifts in addition to proper task understanding to recognize and navigate towards the correct target. Table 5.1 shows the success rates of the different network architectures across the four environments. It is evident that variants of liquid networks outperform other models in terms of task completion. Moreover, we observe that generally recurrent networks work better when tested within the same training distribution such as the Training Woods and Alternative Woods environment. Meanwhile, in out-of-distribution scenarios, their performance drops significantly. However, in such OOD settings, liquid networks significantly stand out with a sparse liquid CfC network having a success rate of 95% in Urban Lawn and 65% in Urban Patio (which contains more natural adversaries).

Beyond the extensive testing conducted at the fixed distance of 10 meters from the target (which is roughly the starting position in the training data), we also perform a range test. For reasons pertaining to logistics and unfortunate weather, we only managed to perform the test for two network architectures, liquid NCPs and LSTMs as the best alternative candidate amongst other RNNs. The experimental scenery is that of the *Training Woods* setup presented above, but with drone's starting positions increasing in distance to the target. We run 5 experiments from each

Table 5.1: **Online test results:** Closed-loop evaluation of the fly-to-target task for the trained policies in four testing environments. Quantitative results are success rates. Higher is better. (n=20)

Algorithm	**Environment**			
	Training Woods	Alt. Woods	Urban Lawn	Urban Patio
LSTM	75%	**90%**	70%	25%
GRU-ODE	25%	5%	35%	15%
ODE-RNN	65%	80%	15%	15%
TCN	15%	0%	0%	0%
CfC (ours)	95%	85%	65%	5%
Sparse-CfC (ours)	50%	**90%**	**95%**	**65%**
NCP (ours)	**100%**	**90%**	55%	**55%**

Table 5.2: **Range test results:** Range test success rates for trained LSTM and NCP policies. (n=5)

Algorithm	**Initial Distance (m)**			
	10	20	30	45
LSTM	80%	80%	20%	0%
NCP (ours)	**100%**	**100%**	**60%**	**100%**

of the following quadrotor-target distances: 10, 20, 30, and 45 meters. The results for each of the networks are summarized in Table 5.2.

The results of this experiment shows strong evidence that liquid networks possess the ability to learn a robust representation of the task they are given and can generalize well to OOD scenarios where other models fail. This observation is aligned with recent works [72] that showed liquid networks are dynamic causal models [22] and can learn robust representations for their perception modules to perform robust decision-making. In particular, it is worth noting that NCP networks managed to fly the drone autonomously from 45m distance to their targets form raw visual data with a success rate of 100%. This is in contrast to an LSTM network that loses the target in every single attempt, leading to a 0% success rate.

Stress Tests

In addition to the flight experiments with the networks running online, we performed a series of offline stress tests as another measure of robustness of different neural architectures to distribution shifts, by perturbing the input frames to the network and observing changes in network outputs. Ideally, networks should be able to continue to achieve high task performance in spite of varied environmental conditions. To simulate this domain randomization, we can perturb images from collected sequences, changing brightness, contrast, saturation, and noise to mimic lighting differences, color variations, and sensor noise found in real-world testing. We can then measure if networks substantially change their control outputs, indicating that they fail to generalize to new environments, or generate consistent outputs across different perturbations, displaying resilience to distribution shifts.

To quantify difference in network outputs, we applied a perturbation to the frames of a recorded sequence, fed the original and perturbed frames to the network, and recorded the original and perturbed output velocities generated by the network. We then applied Forward Euler numerical integration to integrate the velocities into position trajectories. We then measured the final distance between the last point on the two trajectories and recorded this as the perturbed distance and repeated this process for different perturbation types and severities of perturbation.

Fig. 5-1 shows the results of stress testing for noise, brightness, contrast, and saturation perturbations.

We average the trajectory deviations for 10 runs recorded by the networks running in closed-loop. 5 runs were taken from the training environment, and 5 were taken from the urban test task. The perturbed runs were collected across a variety of different network types, including both liquid networks and conventional networks.

We only fed 40% of the frames in each run before measuring the final distance, as the longer the sequence is, the more the trajectories diverge. Because the in-

put images fed to the model are fixed, we cannot model the change in perspective caused by the diverging viewpoints, and any estimates of future position become more inaccurate. To limit the effect of the offline frame generation while still maintaining enough frames to yield statistically significant results, we chose to only keep the first 40% of each sequence.

In all perturbation analysis cases, sparse CfCs demonstrate superior robustness compared to its counterparts. They deviate the least from their nominal trajectories. This observation is consistent with our active testing results where in out of distribution scenarios, sparse CfCs stand out in task completion rate. liquid NCPs also show great resiliency to noise, brightness and contrast perturbations, while they their performance is hindered by change in input pixels' saturation level. We note that in this offline setting, LSTMs are also among the most robust networks after liquid networks, however, their OOD generalization is poor compared to liquid networks.

Attention Profile of Networks

To assess the task-understanding capabilities of all networks, we computed the attention maps of the networks by applying a feature saliency computation method called the VisualBackProp [7] to the CNN backbone that precedes each recurrent network. A more detailed description of the VisualBackProp procedure can be found in subsection Visual Backprop. VisualBackProp associates importance scores to input features during decision-making. The method has shown promise in real-world robotics applications where visual inputs are processed by convolutional filters first. The attention maps corresponding to the convolutional layers of each network in a test case scenario is shown in Fig. 5-2. We observe that the saliency maps of liquid networks (both NCPs and CfCs) are much more sensitive to the target from the start of the flight compared to that of other methods.

We show more comprehensive saliency maps in different scenarios in Supplementary Materials Figs. S1, S2 and S3. We observed that the quality of the attention

maps of the convolutional layers significantly improves for all networks when data augmentation is used during training.

The attention maps are greatly advantageous as they allow us to inspect and interpret reasons for a failed or successful decision-making process by a network under distribution shifts. For instance, we illustrate two out-of-distribution test cases in Figs. 5-2B and 5-2C, in which LSTM agents lose their target by the environmental adversaries while in very similar cases liquid networks manage to keep their focus on the target and complete their tasks. We conclude that learning more robust perception module filters (convolutional filters) can certainly enhance the generalization of the autonomous flight agents. This is while the robustness of the transformation from convolutional heads to the recurrent module (the decision-making compartment) highly depends on the choice of model. Our observations suggests that liquid networks improves representation learning at both these fronts compared to other modern architectures.

Augmentation Effect on Saliency Maps

In both online testing and offline causality analysis, we observed that modifications to the training procedure, especially closed-loop synthetic sequence augment greatly impacted performance. In figures 5-3, 5-4, and 5-5, we highlight the impact of augmentation on the network saliency maps on the CNN backbone that precedes the recurrent network. We analyze image sequences collected in train, indoor lab, and patio testing environments. Each figure contrasts saliency maps for the same sequence taken in a given environment for a network trained with the synthetic augmented sequences and for one network trained without them. Without augmentation, very few neural architectures are capable of capturing the reasoning behind the task, and most networks select regions that aren't the target to motivate their decisions. In all environments, liquid networks like NCP and CFC generate more relevant saliency maps than baselines like LSTM and other CT architectures. When flying these networks in the real world, no network that was

trained without augmentation was capable of solving the learning task when run online, even the liquid networks with better saliency maps.

However, once we introduce augmented sequences in training, the saliency maps of all networks shift their attention towards the target. Despite the saliency maps for most networks appearing to focus on the chair, Table 5.1 shows that not all networks are actually able to navigate to the chair in closed-loop.

The fact that some networks are able to capture the driving factor behind the task before augmentation while others are not implies that some neural architectures inherently have greater affinity for understanding the causal nature of the task, even when the directions are unclear. Some brain-inspired liquid networks have a strong enough grasp of causality that they are able to encode task information into their control outputs and their own recurrent layers and also propagate this information up into the CNN backbone that precedes them, which is what the saliency maps visualize. However, we show that task understanding can be encoded into any neural architecture with enough clear, simple examples of the task, like the synthetic augmentations.

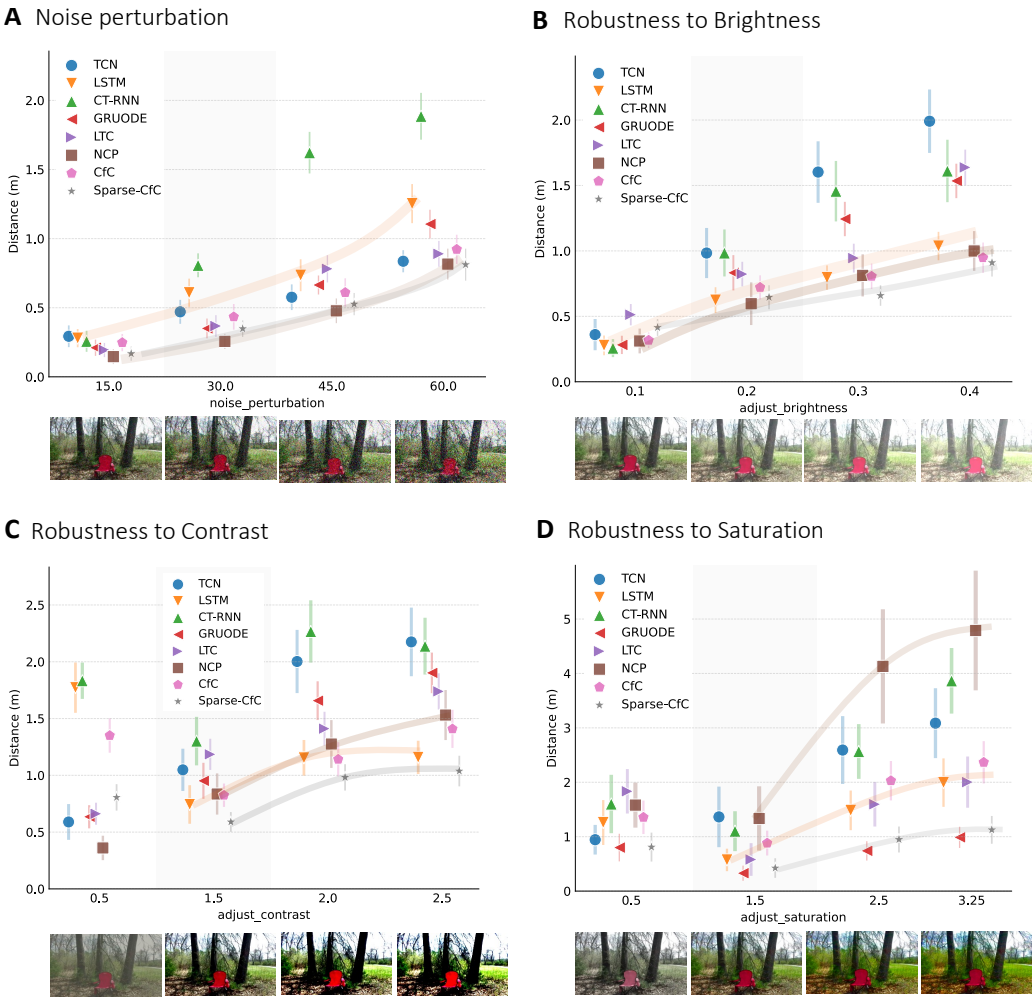

Figure 5-1: **Stress tests:** These experiments run both an original image sequence collected by the drone and a perturbed version through a network, integrate the output velocity commands, and record the final displacement between sequences (lower is better). Bars show the standard deviation across 10 runs. Trendlines for NCP, LSTM, and Sparse-CFC are shown with shaded brown, orange, and gray lines respectively. Observe that liquid networks are generally more robust to perturbation than their discrete counterparts; NCP outperforms LSTM on noise, brightness, and contrast perturbations, while Sparse-CfC outperforms LSTM in all cases. (n=10) **A.** Distances when Gaussian noise with std deviation x is added to image frames. **B.** Distances when brightness is shifted by x times the maximum pixel value. **C.** Distances when contrast in the image is multiplied by x. Note that the first data point, 0.5 is actually a *reduction* in contrast, and moves all pixel values *towards* the mean, while the other 3 data points *increase* contrast and move pixel values *away* from the mean. **D.** Distances when saturation (S in HSV colorspace) is multiplied by x. Note the first data point, 0.5, *reduces* the saturation of colors in the image while the other 3 data points *increase* saturation

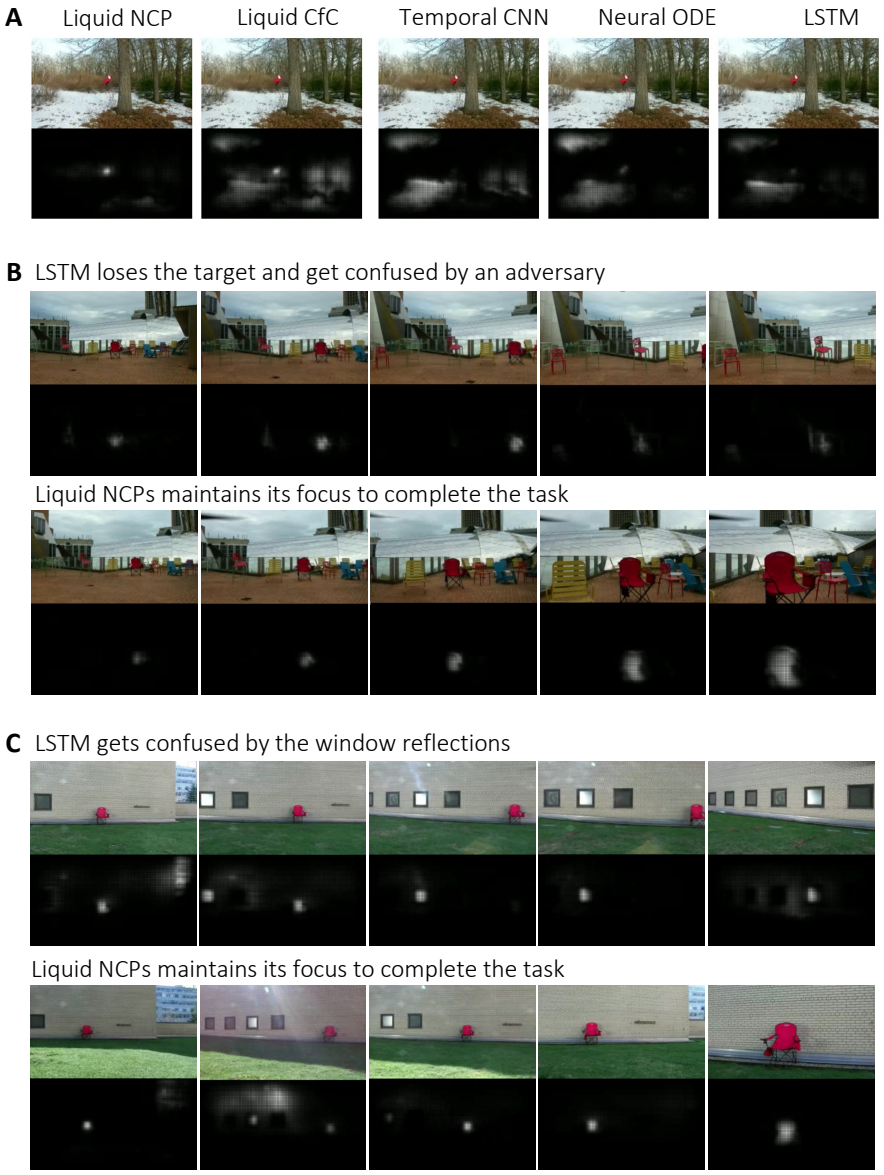

Figure 5-2: **Attention maps. A.** A frame illustration of the attention profile of different networks computed by the VisualBackProp saliency map computation algorithm [7]. Top row shows the input image frame. Bottom row shows the saliency map. The brighter regions indicates a greater concentration of the convolutional layer's attention. Saliency scores range between 0 (black) and 1 (white). **B.** A flight case study where an LSTM agent loses its target and gets confused by an adversary. Left to right shows time progress. This is while liquid NCPs manage to maintain their focus on the target and complete the task. **C.** A similar scenario as in (b), in a different out-of-distribution testing environment.

Figure 5-3: **Training Environment Comparison:** Saliency maps from networks before and after augmentation run on one of the hardest sequences from the training set (black means least contribution to network decision and yellow means highest contribution to network decision). Observe that before augmentation, LSTM, CT-RNN, and GRUODE place little saliency on the chair, instead focusing on edges with lighting changes. The NCP is able to attend to the chair the earliest out of all models, starting at frame 2, but it too places focus on spurious image edges. After augmentation, LSTM, NCP, CFC, and Sparse-CFC are consistently able to attend to the chair after frame 2, and have significantly less noise. The NCP saliency map is still one of the cleanest after augmentation, containing minimal artifacts in frames 1-4 and containing almost none afterwards.

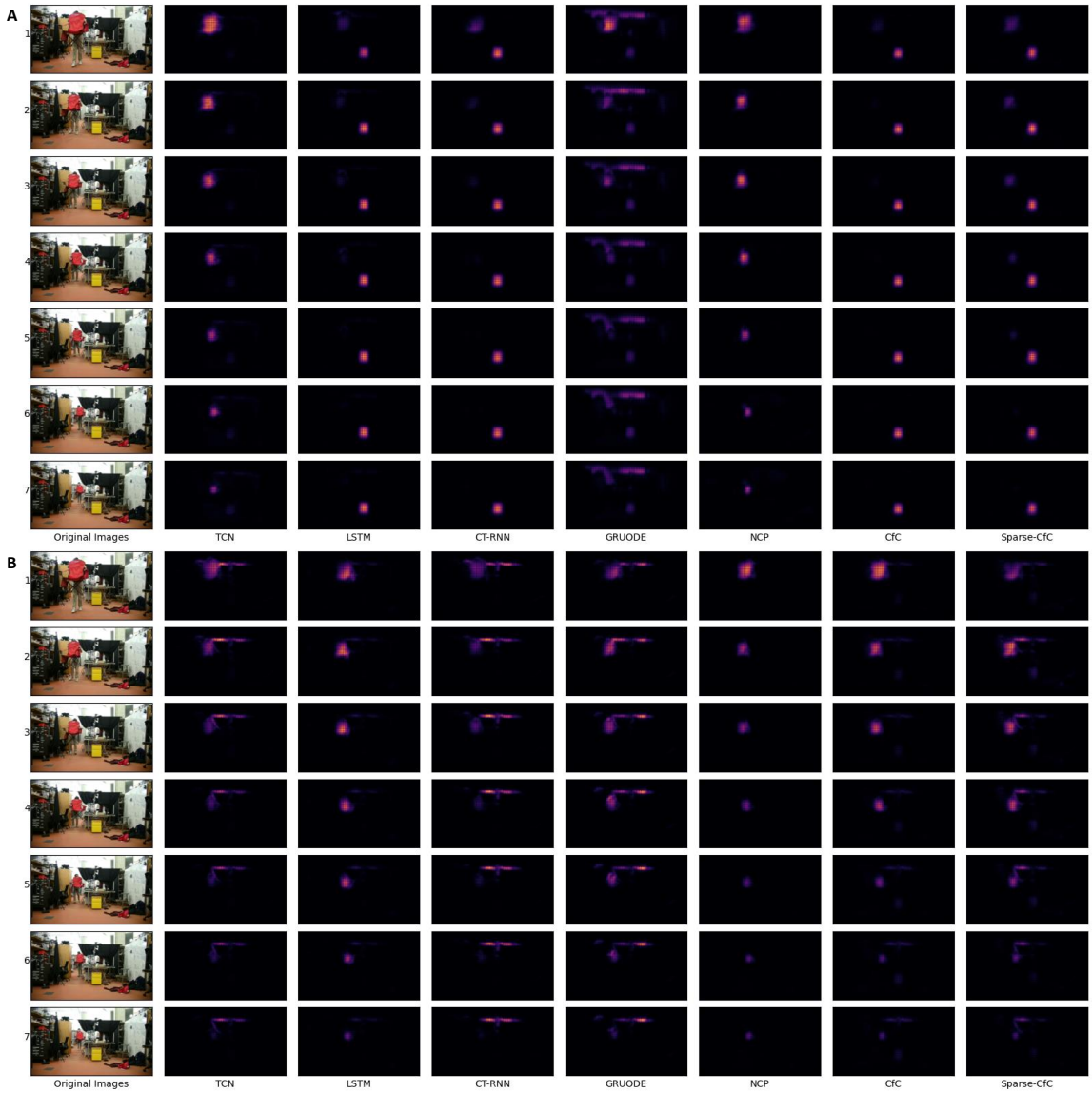

Figure 5-4: **Lab Environment Comparison:** Saliency maps from networks before and after augmentation run on a sequence taken from a cluttered indoor lab environment (black means least contribution to network decision and yellow means highest contribution to network decision). The variety of objects littered throughout the scene creates an image very dissimilar to the train environment. Note that without augmentation, only TCN and NCP focus attention on the chair, and all of the other networks actually focus their attention on the yellow square toolbox instead. When trained with augmentation, all of the model architectures are able to attend primarily to the chair. However, all networks except for NCP and CFC also attend to a thin line above the chair's position.

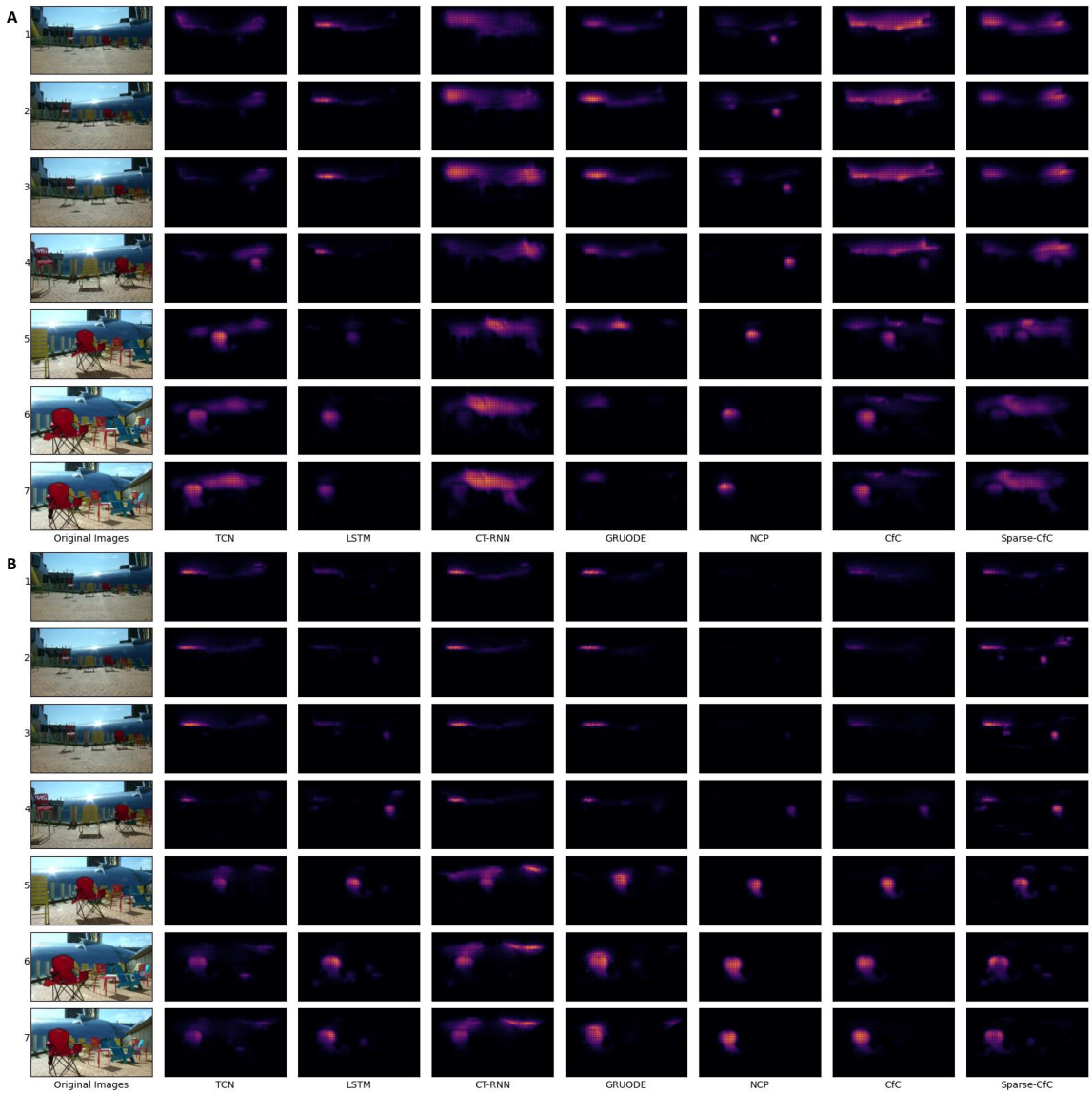

Figure 5-5: **Distractor Chair Environment Comparison:** Saliency maps from networks before and after augmentation run on a sequence from online testing captured on the patio (black means least contribution to network decision and yellow means highest contribution to network decision). The difficult lighting and presence of distractor chairs create a very challenging scenario. Before augmentation, observe that the NCP is the only network to focus attention on the target chair in frames 1-3. Even afterwards, the NCPs saliency map is the most focused on the chair, and contains very little attention on the bright regions caused by reflection. TCN, LSTM, CFC, and Sparse-CfC also pick up on the chair in frames 4-7, but also assign a lot of focus to spurious image regions. After augmentation, the saliency maps of all netowrks improve, and all of them except for TCN focus on the chair in frames 4-7 and contain much less extraneous focus than before. Note that all networks except for NCP focus on a sunny line in the top left corner.

Chapter 6

Conclusions

6.1 Discussion

6.1.1 Choice of models matter for out-of-distribution generalization.

Our experiments show large inequalities between different recurrent neural network architectures when it comes to performing the fly-to-target control tasks in closed-loop, and all the more so when required to generalize in unseen and adversarial environments. Indeed, Table 5.1 suggests that only five out of the seven networks tested are capable of reaching the target in the *Training Woods* environment on half of the attempts or more. Both TCN (15%) and GRU ODE (25%) fail to achieve this threshold even on data from the training background distribution.

These models also perform poorly in the other testing scenarios, with TCN unable to achieve a single success in any of the other three environments and GRU ODE performing worse than on the *Training Woods* scenario with the exception of *Urban Lawn*, where it succeeds about a third of the time (35%). These two architectures exhibit both poor closed-loop performance and generalization and are thus deemed incapable of understanding and performing the control task assigned. Among the models performing reasonably well in both woods scenarios is the ODE-RNN with a 65% success rate when the target is in the same position as

in the training data and up to 80% with the target placed in a slightly different position. Although this network seems to have acquired the capability to achieve the task in closed-loop, it generalizes poorly to unseen environments with 15% OOD success rate in both the *Urban Lawn* and *Urban Patio* experiments.

Other models seem to perform consistently in the woods, with success rates of 75% for LSTM, and over 95% for NCP and CfC on the *Training Woods* test. Peculiarly, the Sparse-CfC model performs relatively poorly in what is expected to be the simplest of environments, succeeding on only half of the runs. On the *Alternative Woods* scenario, LSTM and liquid networks succeed in reaching the target at a high success rate of roughly 90%, showcasing the acquired ability to learn and execute closed-loop tasks from expert demonstrations.

When asked to generalize performance on the *Urban Lawn*, the outstanding performer in this environment is Sparse-CfC, which succeeds quasi-systematically in attending to the target (95% success). On the task of flying to the target in the presence of natural adversaries and distractions in the extremely heavy distribution shift setup provided by the *Urban Patio*, the LSTM succeeds only in 25% of the attempts, whereas both the NCP (55%) and the Sparse-CfC (65%) achieve the task over half the time.

These results suggest a crisp advantage for liquid neural networks for closed-loop end-to-end control learning. Indeed, our brain-inspired networks generalize significantly better than popular state-of-the-art recurrent neural networks, a crucial characteristic for implementation in real-world applications.

In addition to exhibiting superior robustness against changes in background and adversarial examples, the range test suggests liquid neural networks enable another type of generalization that does not hold for LSTM. The results of the experiment show that NCP is capable of consistently achieving the fly-to-target task for distances up to 45 meters although in training sequences the distance to the target does not exceed 15 m. This is all the more impressive as we alternately position the drone at an angle so that the target appears in a different corners of the initial frame. For distances exceeding 30 meters, LSTM is incapable of reaching

the target in all but one run out of ten, whereas NCP achieves this on 8 occasions. These empirical results show that our brain-inspired network display broader task understanding as they extrapolate the essence of the task to larger distances not seen in training.

The greater robustness of liquid networks in the face of data distribution shift is also confirmed through offline stress testing. Indeed, we note that our brain-inspired networks are overall less prone to drifting from their nominal trajectories in the presence of artificially introduced image perturbations. Our architectures surpass LSTM on noise, brightness, and contrast perturbations, with the Sparse-CfC model outperforming all models on each of the perturbations introduced.

6.1.2 Why can liquid networks generalize to OOD settings?

We associate the generalization performance of LTCs to two factors: 1) During the end-to-end training, liquid networks impose an inductive bias on the convolutional filters of the head network, that enable a more robust feature representation associated to the task at hand. The saliency maps of the convolutional head networks are a great empirical evidence of this fact.

2) Liquid networks are dynamic causal models at their cell level [72]. The biological priors imposed on liquid networks form a dynamical system that enables mechanisms for regulating external and internal interventions which are the necessary pillars of causal models [63, 53, 54]. This way, during training they manage to interact with the training data in a causal fashion to learn representations for both control and perception. In fly-to-target tasks, we provided extensive evidence for their ability to drop irrelevant features under heavy distribution shifts and execute the task they have picked from training data, i.e., fly to the target regardless of the background profiles and other environmental variables.

6.1.3 Robustness in perception and control.

The generalization of end-to-end imitation agents heavily relies on the data they have seen and the augmentation strategies in place to help encode the task in network weights. We observed that our data augmentation pipeline significantly improved the perception module of all networks and therefore enabled them to obtain a stable saliency maps with attention on the target (See Figs. S1, S2 and S3 in Supplementary Materials). The augmentation however, did not enable a robust execution of the control task upon change of test environments.

For instance, see the scenarios illustrated in Fig. 5-2B and 5-2C, in which the LSTM's perception module detects the target, but still failed to navigate the drone. In these cases the execution of the navigation task is hindered either by a distraction in the environment, or by a discrepancy between the representations learned in the perception module and the recurrent module. This discrepancy might have led to the LSTM agents to fly away from the target or chose an adversarial target instead of the true one which was detected by its perception module in the first place. In liquid networks however, we see the alignment of the representations between the recurrent modules and the perception modules as their execution of the navigation task is significantly more robust to that of other models.

All in all, through extensive end-to-end closed-loop control testing on fly-to-target tasks and thorough offline perturbation analysis, we show a strong performance demarcation signal in favor of liquid neural networks against the state-of-the-art recurrent neural network models. This capacity to capture causality and contexts, and learn reasoned representations for control originates from the incorporation of principled mechanisms for information propagation inspired by neural models of biological brains into our learning algorithms.

6.2 Lessons Learned

The key finding of the work is that through a greater causal grasp of the learning task, liquid networks are able to generalize better in zero-shot transfer to out of distribution environments in closed loop scenarios. We find that saliency maps that are more focused on the target and less on spurious environmental features lead to better generalization and better robustness to inut perturbations, suggesting that networks with better saliency maps have greater causal affinity.

We additionally observe that closed-loop augmentation is capable of encoding task understanding in a wide variety of network architectures and can greatly improve performance.

6.3 Future Work

One interesting direction to explore would be applying the liquid networks explored in this work to a much more challenging task like mid-air refeuling using a fixed wing aircraft. Fixed-wing vehicles have fundamentally more difficult dynamics, and the precision required for one vehicle to fill another vehicle in the air with fuel is extremely tight. Additionally, it would be fun to try to coordinate a distributed multi-agent system composed of liquid network agents, and to explore possible modifications to the learning process necessary to encourage collaboration.

The results in this paper could be supplemented by additional generalization testing. While we establish translation invariance by repositioning the chair in frame in the online testing and scale invariance by performing the range tests, it would be interesting to see if the networks are rotation invariant by flipping the chair upside down or approaching it from behind. It also might be interesting to see what would happen if the network were fine-tuned on 2 targets, and try to see if the network could learn to approach both. The fly-to-target policy could also be manually adapted to go to multiple targets by steering away from a target and

searching for the next one once the target is a certain size in frame.

One could also try to dig into the causality properties of the recurrent part of each network instead of just the CNN backbone, and simultaneously visualize how both the network input feature and hidden state affect the predictions made by the recurrent section of the RNN.

Appendix A

Resources

Code Organization

All of our code is publicly available at the following 2 repositories:

- `https://github.com/GoldenZephyr/rosetta_drone`: This repository contains all code that was run on the drone hardware for online testing described in Section Closed-loop (online) Testing. It includes scripts for collecting training data and commanding drone velocities with network outputs.

- `https://github.com/dolphonie/drone_causality`: This repository contains all offline training, data preprocessing, and network analysis code for the paper. It contains all code necessary to setup models for testing, including the scripts used to perform the sequence slicing described in section Data Preparation, code for running and orchestrating the multiple training runs described in Alg. 2, and code for performing hyperparameter optimization described in section Hyperparameter Tuning/Network Architectures. Also included are analysis scripts like the one used to generate the stress testing results of Fig. 5-1 and Fig. 5-2, along with additional analysis scripts not used for paper figures.

Training Datasets

The original hand-collected training dataset is published here (filename: devens_snowy_fixed, size: 33.2GB). This dataset was used for training the starting checkpoints. The dataset used for fine-tuning the models tested on the drone can be found here (devens_chair, 2.3GB), and has a subset of the full dataset containing only runs with the chair target.

We have also included the exact synthetic datasets we used for our experiments. We have both a full dataset, (synthetic_small4, 14.7GB) used to train the starting checkpoint and a synthetic chair-only dataset (synthetic_chair, 4.3 GB) used to fine-tune the final models for online testing.

Bibliography

[1] Alexander Amini, Tsun-Hsuan Wang, Igor Gilitschenski, Wilko Schwarting, Zhijian Liu, Song Han, Sertac Karaman, and Daniela Rus. Vista 2.0: An open, data-driven simulator for multimodal sensing and policy learning for autonomous vehicles. *arXiv preprint arXiv:2111.12083*, 2021.

[2] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[3] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[4] Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1):41–77, 2003.

[5] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

[6] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123. PMLR, 2013.

[7] Mariusz Bojarski, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Larry J Ackel, Urs Muller, Phil Yeres, and Karol Zieba. Visualbackprop: Efficient visualization of cnns for autonomous driving. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8, 2018.

[8] Mariusz Bojarski, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Larry Jackel, Urs Muller, and Karol Zieba. Visualbackprop: efficient visualization of cnns. *arXiv preprint arXiv:1611.05418*, 2016.

[9] Johann Borenstein and Yoram Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on systems, Man, and Cybernetics*, 19(5):1179–1187, 1989.

[10] Johann Borenstein, Yoram Koren, et al. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE transactions on robotics and automation*, 7(3):278–288, 1991.

[11] Raja Chatila and Jean-Paul Laumond. Position referencing and consistent world modeling for mobile robots. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 138–145. IEEE, 1985.

[12] Jianyu Chen, Bodi Yuan, and Masayoshi Tomizuka. Deep imitation learning for autonomous driving in generic urban scenarios with enhanced safety. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2884–2890. IEEE, 2019.

[13] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34, 2021.

[14] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 6572–6583, 2018.

[15] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

[16] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 4693–4700. IEEE, 2018.

[17] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

[18] Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. *Advances in neural information processing systems*, 30, 2017.

[19] Scott Emmons, Benjamin Eysenbach, Ilya Kostrikov, and Sergey Levine. Rvs: What is essential for offline rl via supervised learning? *arXiv preprint arXiv:2112.10751*, 2021.

[20] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. In *Conference on robot learning*, pages 357–368. PMLR, 2017.

[21] Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards. *arXiv preprint arXiv:1511.07404*, 2015.

[22] Karl J Friston, Lee Harrison, and Will Penny. Dynamic causal modelling. *Neuroimage*, 19(4):1273–1302, 2003.

[23] Dibya Ghosh, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Manon Devin, Benjamin Eysenbach, and Sergey Levine. Learning to reach goals via iterated supervised learning. In *International Conference on Learning Representations*, 2020.

[24] Prasoon Goyal, Raymond J Mooney, and Scott Niekum. Zero-shot task adaptation using natural language. *arXiv preprint arXiv:2106.02972*, 2021.

[25] Ankur Handa, Karl Van Wyk, Wei Yang, Jacky Liang, Yu-Wei Chao, Qian Wan, Stan Birchfield, Nathan Ratliff, and Dieter Fox. Dexpilot: Vision-based teleoperation of dexterous robotic hand-arm system. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9164–9170. IEEE, 2020.

[26] Ramin Hasani, Mathias Lechner, Alexander Amini, Lucas Liebenwein, Aaron Ray, Max Tschaikowski, Gerald Teschl, and Daniela Rus. Closed-form continuous-time neural models. *arXiv preprint arXiv:2106.13898*, 2021.

[27] Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. A natural lottery ticket winner: Reinforcement learning with ordinary neural circuits. In *International Conference on Machine Learning*, pages 4082–4093. PMLR, 2020.

[28] Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. Liquid time-constant networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(9):7657–7666, 2021.

[29] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[30] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[31] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. *Advances in neural information processing systems*, 32, 2019.

[32] Stephen James, Michael Bloesch, and Andrew J Davison. Task-embedded control networks for few-shot imitation learning. In *Conference on robot learning*, pages 783–795. PMLR, 2018.

[33] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*, pages 991–1002. PMLR, 2022.

[34] Leslie Pack Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the tenth international conference on machine learning*, volume 951, pages 167–173, 1993.

[35] Leslie Pack Kaelbling. Learning to achieve goals. In *IJCAI*, volume 2, pages 1094–8. Citeseer, 1993.

[36] Eric R Kandel, James H Schwartz, Thomas M Jessell, Steven Siegelbaum, A James Hudspeth, Sarah Mack, et al. *Principles of neural science*, volume 4. McGraw-hill New York, 2000.

[37] Christof Koch and Idan Segev. *Methods in neuronal modeling: from ions to networks*. MIT press, 1998.

[38] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

[39] Louis Lapique. Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. *Journal of Physiology and Pathololgy*, 9:620–635, 1907.

[40] Jean-Claude Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012.

[41] Mathias Lechner, Ramin Hasani, Alexander Amini, Thomas A Henzinger, Daniela Rus, and Radu Grosu. Neural circuit policies enabling auditable autonomy. *Nature Machine Intelligence*, 2(10):642–652, 2020.

[42] Mathias Lechner, Ramin Hasani, Manuel Zimmer, Thomas A Henzinger, and Radu Grosu. Designing worm-inspired neural networks for interpretable robotic control. In *International Conference on Robotics and Automation (ICRA)*, pages 87–94, 2019.

[43] Yann LeCun and Yoshua Bengio. Reflections from the turing award winners. In *International Conference on Learning Representations (ICLR)*, 2020.

[44] Xiaoshuang Li, Peijun Ye, Junchen Jin, Fenghua Zhu, and Fei-Yue Wang. Data augmented deep behavioral cloning for urban traffic control operations under a parallel learning framework. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[45] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on robot learning*, pages 1113–1132. PMLR, 2020.

[46] Jacob Mattingley, Yang Wang, and Stephen Boyd. Receding horizon control. *IEEE Control Systems Magazine*, 31(3):52–65, 2011.

[47] Piotr Mirowski, Matt Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Andrew Zisserman, Raia Hadsell, et al. Learning to navigate in cities without a map. *Advances in Neural Information Processing Systems*, 31, 2018.

[48] Anusha Nagabandi, Kurt Konolige, Sergey Levine, and Vikash Kumar. Deep dynamics models for learning dexterous manipulation. In *Conference on Robot Learning*, pages 1101–1112. PMLR, 2020.

[49] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 6292–6299. IEEE, 2018.

[50] Gerhard Neumann and Jan Peters. Fitted q-iteration by advantage weighted regression. *Advances in neural information processing systems*, 21, 2008.

[51] Xinlei Pan, Tingnan Zhang, Brian Ichter, Aleksandra Faust, Jie Tan, and Sehoon Ha. Zero-shot imitation learning from demonstrations for legged robot visual navigation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 679–685. IEEE, 2020.

[52] Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A Efros, and Trevor Darrell. Zero-shot visual imitation. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 2050–2053, 2018.

[53] Judea Pearl et al. Causal inference in statistics: An overview. *Statistics surveys*, 3:96–146, 2009.

[54] Will Penny, Zoubin Ghahramani, and Karl Friston. Bilinear dynamical systems. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 360(1457):983–993, 2005.

[55] Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. CRC press, 1987.

[56] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668. JMLR Workshop and Conference Proceedings, 2010.

[57] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[58] Yulia Rubanova, Ricky TQ Chen, and David Duvenaud. Latent odes for irregularly-sampled time series. *arXiv preprint arXiv:1907.03907*, 2019.

[59] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

[60] José Santos-Victor, Giulio Sandini, Francesca Curotto, and Stefano Garibaldi. Divergent stereo for robot navigation: Learning from bees. In *Proceedings of IEEE conference on computer vision and pattern recognition*, pages 434–439. IEEE, 1993.

[61] Gopal P Sarma, Chee Wai Lee, Tom Portegys, Vahid Ghayoomie, Travis Jacobs, Bradly Alicea, Matteo Cantarelli, Michael Currie, Richard C Gerkin, Shane Gingell, et al. Openworm: overview and recent advances in integrative biological simulation of caenorhabditis elegans. *Philosophical Transactions of the Royal Society B*, 373(1758):20170382, 2018.

[62] Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242, 1999.

[63] Bernhard Schölkopf. Causality for machine learning. *arXiv preprint arXiv:1911.10500*, 2019.

[64] John Schulman, Jonathan Ho, Cameron Lee, and Pieter Abbeel. Learning from demonstrations through the use of non-rigid registration. In *Robotics Research*, pages 339–354. Springer, 2016.

[65] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.

[66] Rupesh Kumar Srivastava, Pranav Shyam, Filipe Mutz, Wojciech Jaśkowski, and Jürgen Schmidhuber. Training agents using upside-down reinforcement learning. *arXiv preprint arXiv:1912.02877*, 2019.

[67] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[68] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

[69] Selim Temizer. *Optical flow based local navigation*. PhD thesis, Massachusetts Institute of Technology, 2001.

[70] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1999.

[71] Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[72] Charles Vorbach, Ramin Hasani, Alexander Amini, Mathias Lechner, and Daniela Rus. Causal navigation by continuous-time neural networks. *arXiv preprint arXiv:2106.08314*, 2021.

[73] Tianhe Yu, Pieter Abbeel, Sergey Levine, and Chelsea Finn. One-shot hierarchical imitation learning of compound visuomotor tasks. *arXiv preprint arXiv:1810.11043*, 2018.

[74] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5628–5635. IEEE, 2018.