Deep Nonlinear Adaptive Control for Unmanned Aerial Systems Operating under Dynamic Uncertainties

Zachary Lamb[*] and Ricardo Sanfelice[†]
Hybrid Systems Lab at UC Santa Cruz, Santa Cruz, CA, 95060, USA.

Zachary I. Bell[‡]
Air Force Research Laboratory, Eglin AFB, FL 32542, USA.

Matthew Longmire[§] and Jared Paquet[¶] and Prashant Ganesh[‖]
Autonomous Vehicles Lab at University of Florida REEF, Shalimar, FL 32579, USA.

Recent literature in the field of machine learning (ML) control has shown promising theoretical results for a Deep Neural Network (DNN) based Nonlinear Adaptive Controller (DNAC) capable of achieving trajectory tracking for nonlinear systems. Expanding on this work, this paper applies DNAC to the Attitude Control System (ACS) of a quadrotor and shows improvement to attitude control performance under disturbed flying conditions where the model uncertainty is high. Moreover, these results are noteworthy for ML control because they were achieved with no prior training data and an arbitrary system dynamics initialization; simply put, the controller presented in this paper was designed with no knowledge about the underlying system, yet yields excellent trajectory tracking performance whilst rejecting significant undesirable model disturbances learned through a DNN. The combination of ML techniques to learn a system's dynamics and the Lyapunov analysis required to provide stability guarantees leads to an effective adaptive controller with applications to safety-critical systems that may undergo uncertain model changes, as is the case for most aerial systems. Experimental findings are analyzed in the final section of this paper, and DNAC is shown to outperform the trajectory tracking capabilities of PID, MRAC, and the recently developed Deep Model Reference Adaptive Control (DMRAC) schemes.

I. INTRODUCTION

Linear controllers have drawbacks in controlling rigid body attitudes for Unmanned Aerial Systems (UAS's). For one, they are only effective around some small flight envelope near the equilibrium point, and two, they often require extensive lookup tables to control a flight over a wider regime [1] [2] [3]. Many nonlinear and adaptive control strategies exist to address these issues [4][5]; however, fewer solutions exist which combine these two strategies into a single controller. Furthermore, some nonlinear control solutions require prior accurate identification of various system parameters such as drag coefficients [6]. These are often difficult to estimate and still do not address the issue of a changing model where these parameters are also uncertain. Seeing as how rigid bodies in many UAS applications are required to perform nonlinear maneuvers while undergoing stochastic model disturbances such as wind gusts, fuel slosh, or turbulence, there is a real need to build adaptive strategies into nonlinear controllers themselves.

The objective of an adaptive controller is to improve tracking performance even when the dynamics of a system are uncertain or changing. By assuming a model and estimating some set of parameters, control laws can be configured to change alongside the model. Model Reference Adaptive Control (MRAC) is one such adaptive scheme that works in a similar manner; by designing a reference model which one would want the real system to behave like, MRAC is able to optimize a set of control weights in order to produce system output which matches that of the reference model.

[*]Zachary Lamb is currently with Lockheed Martin, Space. Formerly with the Air Force Research Lab at Eglin AFB, FL 32542, USA and the Hybrid Systems Lab at UC Santa Cruz, Santa Cruz, CA 95060, USA.

[†]R. G. Sanfelice is with the Hybrid Systems Lab at UC Santa Cruz, Santa Cruz, CA 95060, USA.

[‡]Zachary I. Bell is with the Munitions Directorate, Air Force Research Laboratory, Eglin AFB, FL 32542 USA. Email: zachary.bell.10@us.af.mil.

[§]Matthew Longmire is with the Autonomous Vehicles Lab, University of Florida REEF, Shalimar, FL 32579. Email: m.longmire@ufl.edu

[¶]Jared Paquet is with the Autonomous Vehicles Lab, University of Florida REEF, Shalimar, FL 32579. Email: jared.paquet.edu

[‖]Prashant Ganesh is currently with EpiSys Science Inc. Formerly with the Autonomous Vehicles Lab, University of Florida REEF, Shalimar FL 32579, USA.

This works by minimizing the loss between the reference model output and the true systems output. This technique is commonly applied in the field of aerospace. Yet, one key drawback is that MRAC relies on a linear model assumption with few parameters.

Recently, a nonlinear adaptive Time Delay Estimation based approach was employed to handle learning state-dependant uncertainties for a quadrotor [7]. This method reduced the necessity for a priori knowledge of the system and enabled the learning of more natural nonlinear dynamics through a set of parameters. This method has shown promising results, and its implementation requires only a small amount of intuition on the bounds of model parameters which are specific to each dynamic system. However, applying modern neural-network based learning strategies would reduce the required model information even further, facilitating simpler generalization between systems, while also potentially allowing for more complex estimation of the uncertainties in the dynamics. To this effect, neural networks have been used in recent times to replace traditional model-based learning strategies for their empirical improvement over other system identification methods [8].

Neural networks which include one or more hidden layers are defined as Deep Neural Networks (DNN), and are often regarded as Universal Function Approximators (UFA) for their ability to approximate any function. However, while neural networks are powerful, many of their controller based applications lack performance guarantees. Proximal Policy Optimization (PPO) is one such example of a Reinforcement Learning (RL) algorithm that lacks these properties. PPO typically requires many thousands or even millions of training cycles to become effective at the task being trained on. During this training phase the target system will behave undesirably, thus requiring the training be done in simulation. Moreover, once a policy of this kind has been sufficiently trained in a simulation environment, questions of the environment's fidelity and whether or not performance will translate to hardware still remain. Because of this lack of a notion of stability for these types of ML strategies, low-level autopilot control is often avoided. Instead, many RL control techniques tend to be implemented at a high level for path planning where their outputs can be constrained to maintain reasonable trajectory commands [9] [10]. To effectively build an ML based adaptive autopilot, one must maintain a notion of stability, as well as a fast time-to-learn; i.e. a policy that takes millions of learning cycles to become effective will not be useful in controlling a quadrotor if an instantaneous change to the model occurs, such as when a propeller is damaged.

Building on these overarching issues of a slow time-to-learn and lack of stability guarantees in some machine learning control policies, [11] designed a DNN based Model Reference Adaptive Controller (MRAC) which was coined "Deep MRAC" (DMRAC). By marrying the ability of a DNN to learn models with the the ability of MRAC to provide bounded stability guarantees, a stable adaptive controller was designed to properly allow DNN based control to thrive on a safety-critical system at low-levels of the feedback loop.

However, as was a limitation with traditional MRAC, DMRAC forces linear model tracking behavior. These linear model based controllers are effective at tracking trajectories near the point of linearization, yet for UAS's required to perform nonlinear attitude maneuvers, a nonlinear control strategy is preferred. In an attempt to improve on DMRAC, [12] designed DNAC, a generalized nonlinear adaptive controller which uses a DNN to estimate the model uncertainty. This design addresses issues associated with linear controllers, and shows promising trajectory tracking results for nonlinear systems in simulation. In DNAC, a Lyapunov analysis is combined with a DNN to provide performance guarantee's which an RL based controller may not have.

Building off of the trajectory tracking results achieved by DNAC in simulation for a Van der Pol oscillator, this paper features an easily configurable control architecture for UAS's which significantly improves tracking performance and can adapt to changes in the model, as tested on hardware. The contributions of this paper are summarized as follows:

1) DNAC's setup is generalized with no reliance on prior knowledge of the underlying system's dynamics, allowing it to be applied with little to no changes to any system.
2) The unique learning structure of the DNN allows for fast real time updates to the model, resulting in a controller capable of tracking desired trajectories under rapidly evolving dynamic uncertainties.
3) Most applications of ML based controllers lack boundedness guarantees. This approach benefits from the important learning capabilities of a DNN, while maintaining stability guarantees necessary to control a safety critical system.
4) This work not only successfully applies [12] to hardware, but does so using a relatively small network size. This suggests DNAC may be useful as an application of ML to computationally-constrained systems.

This paper begins with necessary background in Section II, providing adaptive update and control law algorithms from [12] as well as describing aspects of the DNN design. Section III dives into the quadrotor control platform, including details about the onboard hardware and software. Finally, Section IV presents a description of DNAC's configuration and compares hardware results for PID, MRAC, DMRAC, and DNAC attitude controllers. These controllers are run in experiments where the quadrotor attempts to track a desired trajectory whilst undergoing heavy model disturbances which include a crosswind, wall effect, added mass, and sloshing effect.

II. BACKGROUND

A. System Dynamics

In this paper, nonlinear systems with dynamics affine in control input are considered given by

$$\dot{x} \quad = \quad f(x) + g(x)u, \tag{1}$$

where $x \in \mathbb{R}^n$ is the state of the system with $n \in \mathbb{Z}$ components, $g : \mathbb{R}^n \longrightarrow \mathbb{R}^{n\times n}$ is the control effectiveness function, $u \in \mathbb{R}^n$ is the control input, and $f : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ represents the drift dynamics which are usually uncertain. Often times the underlying dynamics do not reflect the model exactly; however, in practice, the control affine structure is sufficient to approximate the model. Additionally, the true system model is typically unknown and assumptions are made enabling the utilization of adaptive control methods. The recent result in [12] proposes a method to estimate f using a DNN and demonstrates how to estimate g using a novel control structure. Expanding this concept to systems where the control effectiveness is constant and full rank, it follows that a constant g can be utilized as a user-defined, diagonal gain matrix. The DNN compensates for the uncertainty of the system enabling the DNN control structure to add stable adaptability.

Motivated by [12], the model uncertainty f is modeled via a DNN as

$$f(x) \quad = \quad W^\top \sigma(\Phi(x)) + \epsilon(x), \tag{2}$$

where, by the UFA property [13], $W \in \mathbb{R}^{L\times n}$ is the unknown ideal output layer weights, $L \in \mathbb{Z}$ denotes the user-defined number of neurons in the output layer, $\sigma : \mathbb{R}^p \longrightarrow \mathbb{R}^L$ is the output layer activation function, $\Phi : \mathbb{R}^n \longrightarrow \mathbb{R}^p$ is the ideal unknown DNN features function, $p \in \mathbb{Z}$ denotes the user-defined number of inner layer features, and $\epsilon : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ is the bounded unknown function reconstruction error. The ideal unknown DNN features function is expressed as a function composition of the inner layer weights and activation functions, namely, $\Phi(x) = (\phi_k \circ \phi_{k-1} \circ ... \circ \phi_1)(x)$, where $\phi_l = \sigma_l(W_l^\top \phi_{l-1} + b_l),\ l \in \{1, 2, ..., q\}$, $q \in \mathbb{Z}$ denotes the number of inner layers of the DNN, $W_l \in \mathbb{R}^{L_{l-1}\times L_l}$ represents the l^{th} inner layer weights with L_l neurons, and ϕ_l denotes the l^{th} inner layer activation function. This yields the following approximation of the model uncertainty:

$$\widehat{f}(t, x) \quad \triangleq \quad \widehat{W}^\top(t)\sigma(\widehat{\Phi}(x)), \tag{3}$$

where $\widehat{W}(t)$ and $\widehat{\Phi}(x)$ are the current estimates of W and $\Phi(x)$, respectively. Updates to $\widehat{W}(t)$ and $\widehat{\Phi}(x)$ in (3) is subsequently broken into two stages. The output weights $W(t)$ are estimated online in real-time using a Lyapunov-based adaptive update law. The inner layer weights, $\Phi(x)$, are estimated using batches of data collected online, for which a user-defined loss function is optimized using the extended stochastic gradient decent algorithm, Adam [14]. The combination of online and batch updates allows for fast, real-time adaptation of the outer-layer weights of the DNN, and slower-timescale learning for inner-layer weights to learn the latent features of the nonlinear dynamics.

While the DNN compensates for the model uncertainty term $f(x)$ from (1), the user-defined control effectiveness matrix is selected as

$$\widehat{g}(x) \quad \triangleq \quad \mathrm{diag}(\widehat{g}_1, \widehat{g}_2, ...\widehat{g}_n), \tag{4}$$

where $\widehat{g}_i,\ i \in \{1, ..., n\}$ is the ith element of the control effectiveness gain matrix. Since this matrix is inversely proportional to the input in the subsequently designed control law, $\widehat{g}$ can be designed as another tunable gain matrix to adjust control effort.

Remark 1 *Experimental results in Section IV will demonstrate that utilizing the assumptions made for $\widehat{f}(x)$ and $\widehat{g}(x)$ is sufficient to derive a controller which adeptly learns to stabilize a system with unknown and or changing dynamics. However, if desired, a more accurate $\widehat{g}(x)u$ term can be configured to adaptively update using system ID techniques following Assumption 1 in [12].*

B. Control Design

The control objective is to minimize the norm of a time-varying trajectory tracking error for a given reference $t \mapsto x_d(t)$. For a state trajectory $t \mapsto x(t)$, the tracking error is defined as

$$e(t) = x_d(t) - x(t), \tag{5}$$

where $e : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$. To achieve this goal, the control design determines a robust and adaptive control law that utilizes the DNN estimates of the model uncertainty and guarantees trajectory tracking for (1), namely, $e \to 0$ as $t \to \infty$. The Lyapunov analysis for the subsequent control and update law follow [12] closely and are therefore left out of this paper; however, the resulting control law $u(e, x, t)$ and outer-layer weight update law $\dot{\widehat{W}}(e, x, t)$ were derived to drive the tracking error to zero.

$$u(e, x, t) \triangleq \widehat{g}^{-1}(-Ke - K_s \text{sgn}(e) + \dot{x}_d(t) - \widehat{f}(x, t)), \tag{6}$$

$$\dot{\widehat{W}}(e, x, t) \triangleq \Gamma_W \sigma(\widehat{\Phi}(x(t)))e^\top(t), \tag{7}$$

where K, K_s, and Γ_W are user-defined tuning parameters. It is also useful here to acknowledge the conditions for which these update laws yield asymptotic stability,

$$\epsilon(x)e(t) < Ke(t)^2 + K_s \text{sgn}(e(t))e(t) \tag{8}$$

This constraint is equivalent to the alternative form presented in Theorem 1 of [12], and is a strong result for the application of DNN's to safety critical systems. Recall that $\epsilon(x)$ relates to the *terminal* loss of the function approximation, not the current loss - In essence, this result states that so long as a DNN of sufficient learning capability has been merely setup, and the gains K, K_s are large enough, then update laws (6), (7) yield tracking of the trajectory error $e(t)$.

Remark 2 *Generally, a large Γ_W is associated with the ability to quickly learn the system dynamics, and a large K is associated with fast trajectory tracking convergence. In practice, if Γ_W is too small, then convergence of the model uncertainty term may be slow. As a result, a large K will amplify undesirable signals present in $\widehat{f}$. For this reason, one should first tune Γ_W for state estimation convergence, then tune K for tracking performance. K_s is a sliding mode control and should remain positive but close to 0. User-defined training parameters for updating the inner-layer weights of the DNN, such as the number of epochs, activation functions, and number of hidden layers, are selected to enable the DNN to learn the nonlinear and slower timescale components of the dynamics.*

C. DNN Inner-Layer Design and Update

The inner-layer weights are updated in batches after a replay buffer has reached the set memory size. To perform the j^{th} batch update to the inner-layers, data samples are stored in the j^{th} replay buffer, $\mathcal{B}_j \triangleq \{B_k\}_{k=1}^{M}$, where $B_k = [\dot{x}_k, x_k, g(x_k)u_k]$, and $\dot{x}_k, x_k, u_k$ represent $\dot{x}(t_k), x(t_k), u(t_k)$, $t_k \in [t_{j-1}, t_j]$, $k \in [1, 2, ..., M]$, M is the user-defined memory size, and t_{j-1} is the time the last update to the inner-layer features completed. Once the j^{th} buffer has stored M samples, an update is performed for the inner-layer weights using the j^{th} batch of data in $\mathcal{B}_j$ to compute the Smooth L1 loss

$$\mathcal{L}_j = \frac{1}{M}\sum_{k=1}^{M} \text{L}_k, \tag{9}$$

where

$$\text{L}_k = \begin{cases} 0.5\frac{(\dot{\widehat{x}}_k - \dot{x}_k)^2}{\beta}, & \text{if } |\dot{\widehat{x}}_k - \dot{x}_k| < \beta, \\ |\dot{\widehat{x}}_k - \dot{x}_k| - 0.5\beta, & \text{otherwise,} \end{cases} \tag{10}$$

$$\dot{\widehat{x}}_k = \widehat{f}(x_k) + \widehat{g}(x_k)u_k \tag{11}$$

Here, $\dot{\widehat{x}}_k$ is a DNN-based estimate of $\dot{x}_k$, which along with the smoothing parameter β, is used in calculating the loss via (9) and (10). After computing the loss, an optimization step is taken to update the inner-layer weights via Adam.

III. QUADROTOR CONTROL PLATFORM

The problem setup in Section II is generalized for controllable states of nonlinear systems to track time-varying trajectories, for example, the attitude and angular velocities of a quadrotor. To demonstrate the performance of this control design, quadrotor flight experiments are performed for which a variety of uncertain model disturbances are present. An overview of the quadrotor control setup and position-level feedback loop will be described in Subsection III.A. The Robot Operating System (ROS) is used for message passing and facilitates this entire feedback structure.

The experimental setup uses ROSflight [15] - an autopilot firmware that is used to control the quadrotor. This specific firmware is useful because it has support for multiple Flight Control Units (FCU's) as well as a software-in-the-loop (SIL) simulation environment known as Gazebo. ROSflight provides a standard PID attitude control scheme which is naive to many of the linear model-based control issues mentioned in Section I. The ROSflight attitude PID scheme is improved upon via a parallel DNAC implementation which does not so much as replace the attitude level PID, but rather augments it. This architecture is preferred as it provides a baseline controller to facilitate stable control of the quadrotor in nominal environments, while allowing DNAC to assist when uncertain disturbances inject themselves into the system. This augmentation architecture has been used in [16] for a similar MRAC implementation. Moreover, this augmentation scheme is easily reproducible at the position and velocity levels, allowing for multiple DNAC's to be run concurrently. Section III.B dives further into the aforementioned augmented ACS.

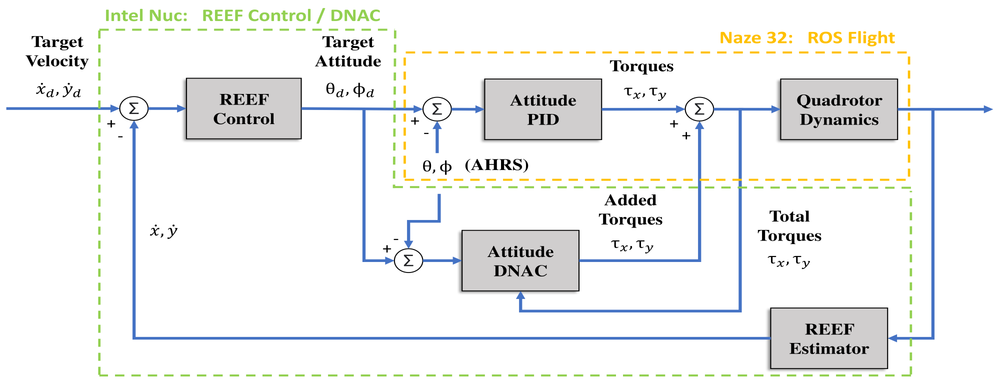

Figure 1 **DNAC Augmented Attitude Control System - DNAC acts in parallel to the ROSflight attitude PID using ROSflight estimated attitudes and target attitudes produced by the REEF Control PID. REEF Control produces these target attitudes using a PID loop based on REEF Esimator velocities and target velocities. The control system is flexible to different target velocity generators, and in theses experiments, the target velocities are generated by a PID loop on position tracking error (not shown).**

A. Baseline System Architecture

The flight software in this work is run onboard a custom built quadrotor capable of carrying several sensors and powerful computational devices. The ROSFlight autopilot runs on an STM32 based Naze32 board and handles PID based attitude control of the drone; subsequent controllers are required to control position and velocity. The REEF Estimator/Controller package [17] is a simplified velocity estimator and controller used to estimate the body-level velocity of the quadrotor by using data from a combination of sensors – namely, an RGB-D camera, motion capture system, sonar and an IMU. These velocity estimates are fed into the REEF Control module which is implemented using a PID controller to output attitude and thrust commands. All of this processing and computation occurs on an Intel NUC companion computer which is also mounted onboard the quadrotor and serially linked to the FCU. The serial delay in communication is negligible in this setup.

Thrust and attitude commands are then fed into the ROSFlight attitude level PID controller, where they are compared against the onboard Attitude Heading Reference System (AHRS) estimates which are generated via a complementary filter [18]. Finally, the attitude level PID produces torque commands which are converted to motor PWM signals through a motor mixing algorithm. The reference signal for the REEF Controller comes from a position trajectory generator node. It is through this process that the REEF Estimator and Controller aid in commanding the drone to any desired position. A block diagram of this entire control system (with DNAC augmentation) is presented in Figure 1.

B. Augmented DNAC Attitude Control System

The control configuration described in Subsection III.A is typical for a quadrotor's flight control design, however, this configuration results in an inherent inability for the drone to adapt to changes in the model (e.g. a sudden gust of wind or a propeller is damaged). There are also limitations to the range of attitude trajectories the drone can reliably track because of linear model assumptions that were made when the controller was designed. To improve the adaptability of the controller, DNAC can be used in parallel with the stable ROSflight attitude PID controller running onboard the FCU. Training neural networks on microcontrollers is difficult, thus DNAC is run onboard the companion computer. From there, DNAC operates on the same attitude commands and AHRS estimates as the ROSflight PID controller. It is in this way that the two separate controllers ensure they are operating on the same tracking error terms. Next, DNAC computes *added torque* commands via control law (6), which are then sent to the FCU and added to the output of the ROSflight PID block. Finally, this is fed back into DNAC as a *total torque* command which is necessary for learning the correct dynamics. Figure 1 provides a visual of this implementation.

The reasoning behind augmenting the current ROSflight autopilot and not replacing it is two-fold: 1) It is typically safer to restrict the amount of contribution an ML based controller can have on a system in case of malfunction. If DNAC becomes dysfunctional for any reason, the augmentation scheme can be severed and the stable PID attitude controller takes over 100% of the flight control. With this augmentation scheme, it is up to the user how much they wish for DNAC to contribute to the ACS. 2) This augmentation design adds to the simplicity of implementation for other UAS's where stable flight controllers already exist and the user only wishes to see if performance may be improved without removing a large chunk of the baseline performing software.

Without loss of generality, the experiments run in the next section will focus on the roll and pitch of a quadrotor to demonstrate DNAC's ability to adaptively trim out a quadrotor under heavy wind and mass disturbances. Yaw is left out of the augmentation as it does not yield significant benefit because the applied disturbances tend to effect roll and pitch more-so.

IV. RESULTS

This section covers two experiments, where the purpose is to fly the quadrotor using various attitude control schemes under highly nonlinear dynamic uncertainties, and compare the results for each controller. Subsection IV.A begins with DNAC's control configuration; this includes gain values, neural network architecture, and loss function choice. PID, MRAC, and DMRAC control configurations are discussed less extensively. Subsections IV.B and IV.C then cover experimental results and analyze the performance of each attitude controller. A standard ROSflight PID scheme is the baseline controller to which each additional controller augments.

DNAC runs in parallel to this PID as previously described in Figure 1. Additionally, MRAC and DMRAC are implemented in a similar fashion, where the outputs of their respective control laws augment that of the original ROSflight PID. These four control schemes (PID, PID+MRAC, PID+DMRAC, PID+DNAC) are flown in a highly uncertain environment where disturbances include a crosswind, wall effect, hanging mass, and sloshing effect from a partially filled water bottle. An image of the quadrotor operating under some of these conditions is provided in Figure 2.

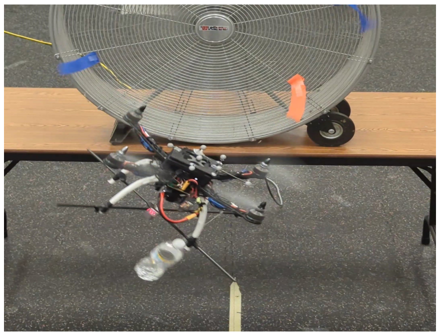

Figure 2 **Quadrotor demonstrating the effect of the sloshing hanging mass and wind disturbances whilst tracking a circular trajectory. The quadrotor is shown 1 m from the fan.**

The attitude RMSE is compared for each controller, and position plots are presented as a visual aid. Note that the original ROSflight PID is capable of flying each of the trajectories in the experiments below given no disturbances are present. Moreover, each augmentation control scheme for all four controllers have been tested to work while operating under no model disturbances as well.

A. Controller Configuration

The original ROSflight PID scheme has been previously covered in [15]. MRAC and DMRAC control schemes follow closely to the configurations found in [19]. The DNAC control configuration is as follows: Beginning with a definition of the state and input, define $x = [\phi, \theta]^\top$, and $u = [\tau_\phi, \tau_\theta]^\top$. From the system given in (1), it follows that the input to the DNN estimate of the model uncertainty, $\widehat{f}(t, x)$, is the current roll and pitch of the drone. The setup of control law (6) begins with the assumption that both torque commands will actuate the vehicle with the same amount of control effectiveness. As such, a simple starting point for the control effectiveness matrix is $\widehat{g}(x(t)) = I_2$. However, in practice $\widehat{g}(x(t))$ can be exploited as another tuning parameter since $\widehat{g}$ is inversley proportional to the outputted control effort. This relation allows for using $\widehat{g}$ as a method to reduce the overall effectiveness of u. In the following experiments, $\widehat{g} = 100 \cdot I_2$. While this approximation is likely inaccurate, the DNN will compensate for it in the model uncertainty term, $\widehat{f}(t, x)$. This model uncertainty is approximated via update law (3), which uses a DNN to adjust the inner layer weights in batches and a nonlinear adaptive law (7) to adjust the outer layer weights in real-time. The real-time learning gain is set as $\Gamma_W = 10$. This term effects the outer layer weight estimates similar to the step size in gradient descent learning strategies. Finally, the proportional control gain and sliding mode gain are set as $K = 10$ and $K_s = 0.001$ respectively. In practice, K_s is kept small.

The slower timescale learning is accomplished by utilizing PyTorch with a network structure $\widehat{\Phi} : \mathbb{R}^2 \to \mathbb{R}^8$ which is decomposed for each layer as $\widehat{\Phi}(x) = (\widehat{\phi}_3 \circ \widehat{\phi}_2 \circ \widehat{\phi}_1)(x)$ where $\widehat{\phi}_1 : \mathbb{R}^2 \to \mathbb{R}^3, \widehat{\phi}_2 : \mathbb{R}^3 \to \mathbb{R}^4, \widehat{\phi}_3 : \mathbb{R}^4 \to \mathbb{R}^8$ and $\widehat{\phi}_l(x) = \sigma_l(\widehat{W}_l^\top x + \widehat{b}_l)$. The associated activation functions after each linear layer are σ_1 = tanh, σ_2 = log-sigmoid, and σ_3 = tanh. The network is finally transformed by the outer layer weights, $\widehat{W}$, to map the output back into the $\mathbb{R}^2$ space as $\widehat{W}^\top \sigma(\widehat{\Phi}(x))$.

The replay memory of size $M = 100$ is split evenly into smaller batches of size $S_b = 20$, and the 5 segments are shuffled randomly. The state x (roll and pitch) is extracted from these mini batch segments, and are passed through the network $N_e = 5$ (number of epochs) times each during training cycles. This training results in an $\widehat{f}(x_k)$ term which is used alongside batch data $\widehat{g}(x_k)u_k$ to estimate $\hat{\dot{x}}_k$ (the roll and pitch rates of the drone) via (11). Finally, an optimization step is taken to minimize the smooth L1 loss between the network's approximation of $\hat{\dot{x}}_k$ and the measured $\dot{x}_k$, which is present in the replay buffer. A summary of these controller parameters is provided in Table 1.

Parameter	Value	Description
K	10	Control Gain
K_s	0.001	Control Gain
$\widehat{g}$	$100 \cdot I_2$	Model Assumption
Γ_W	10	Online Learning Gain
M	100	Batch Memory Size
S_b	20	Training Batch Size
N_e	5	Number of Epochs

TABLE 1 **DNAC Controller parameters related to control law (6), adaptive update law (7), and DNN architecture.**

Remark 3 *In the experiments that follow, it is worth mentioning that DNAC outperformed each of the opposing controllers even with a randomly initialized DNN; this implies that while each of the network weights were not ideal for approximating the model, DNAC maintained stability of the system. This point is in correspondence with the claim from Section I that DNAC is capable of achieving tracking performance with no prior understanding of the underlying system's dynamics, and is an important result for proving the efficacy of DNAC as a useful flight controller on hardware. Moreover, not only was the network randomly initialized, the network size itself is small – this is an important factor which opens up the opportunity for DNAC to be implemented on systems with computational constraints.*

Remark 4 *Because a measurement (or estimate) of the state derivative with respect to time is required for the training of the inner-layer weights, controlling velocity or acceleration may prove more difficult, as it requires an accurate estimate of acceleration or jerk respectively.*

B. Experiment 1

Using the quadrotor control setup outlined in Section III, the quadrotor is flown in a 1 m radius circular trajectory with a water bottle filled to 125 mL hanging from a string attached at the base of the motor (located 0.25 m from the center of mass) in the positive pitch – positive roll quadrant. A large fan (36 in diameter) is then placed 2 m from the center of the circle trajectory, and creates an 18 mph crosswind. Recall that the quadrotor is tracking velocity commands generated via cascading PID loops. These velocity PID controllers generate attitude references which the drone's augmented ACS attempts to track. As such, positional tracking is not a direct measurement of each controller's performance; however, it is still an indirect indicator to evaluate performance.

The circularity of the quadrotor's trajectory (Figure 3) is a visual evaluation of each control scheme's (PID, MRAC, DMRAC, DNAC) robustness to disturbances. Under the conditions, each augmentation controller improves upon baseline PID performance (as quantified in Table 2); this is evident by the removal of the cusps in the baseline PID trajectory which is most obvious when comparing PID and MRAC performances. The deep controller schemes go further and remove the PID's oscillatory behavior with varying levels of success. The DNAC scheme not only rejects oscillatory behavior better than the DMRAC scheme, but the DNAC scheme also reduces the position tracking error norm to 16.6 cm which is a 32% decrease from DMRAC, a 37.8% decrease from MRAC, and a 44.7% decrease from PID. Furthermore, DNAC reduces the velocity tracking error norm to 14.9 cm/s which is a 49.3% decrease from DMRAC, 53.4% decrease from MRAC, and a 63% decrease from PID.

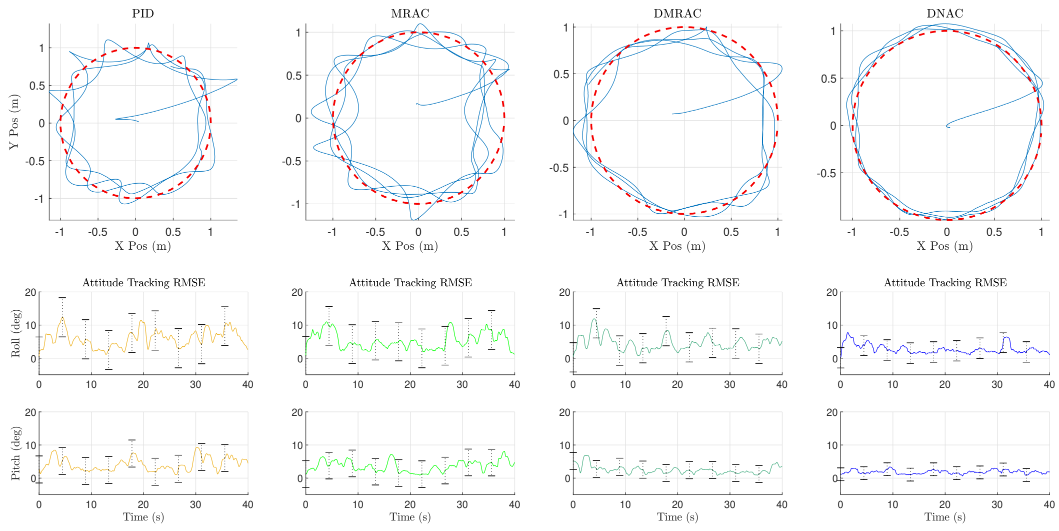

Figure 3 **Top row: Quadrotor desired trajectory (red dashes) and true trajectory (blue) results for the hardware experiment. Model disturbance rejection is visualized by the circularity of the quadrotor's true trajectory. Bottom row: Quadrotor moving RMS attitude tracking error (solid line) and standard deviation (vertical bars). From left to right: (i) baseline PID performance (ii) baseline PID with MRAC augmentation (iii) baseline PID with DMRAC augmentation (iv) baseline PID with DNAC augmentation.**

Controller	$\|\theta, \phi\|_2(°)$	$\|x, y\|_2(\mathrm{cm})$	$\|\dot{x}, \dot{y}\|_2(\mathrm{cm/s})$
PID	6.88	30.0	40.2
MRAC	6.13	26.7	32.0
DMRAC	4.93	24.4	29.4
DNAC	3.06	16.6	14.9

TABLE 2 **Average tracking error L_2 norms for flight data taken while the drone tracks a circular trajectory in Experiment 1.**

While the visual improvements from each position plot are insightful, a more relevant performance metric is to measure the error between the quadrotor's attitude trajectory and the reference attitude trajectory. The attitude error norm results for each controller show that the DNAC implementation improves upon DMRAC performance by an error decrease of 37.9%, MRAC performance by an error decrease of 50.8%, and baseline PID performance by an error decrease of 55.5%. Furthermore, DNAC yields a lower standard deviation on attitude tracking error than the other controllers. The standard deviation of each controller (indicated by black bars in Figure 3) is summarized in Table 3.

Controller	$\sigma_\phi(^\circ)$	$\sigma_\theta(^\circ)$
PID	5.92	4.09
MRAC	5.85	4.02
DMRAC	4.42	2.59
DNAC	3.08	1.93

TABLE 3 **Standard deviation of the attitude tracking error for each controller configuration in Experiment 1.**

This flight test presents DNAC's ability to handle disturbance rejection in an unexplored situation. It's able to not only learn the model quickly enough to maintain tracking from the beginning of flight, but also track the trajectory much better than any of its competitors.

C. Experiment II

While the first experiment exists to place emphasis on DNAC's strong disturbance rejection capabilities, the next experiment will highlight the fast learning ability of the control structure. In the second experiment, a new desired trajectory is designed and another model disturbance is introduced. The quadrotor now flies in a four-petal rose pattern described by the polar coordinates $r = a\sin(2\theta)$, where $a = 2.8$ and $\theta \in [0, 2\pi]$. The same model disturbances from Experiment 1 are present, as well as an additional wall effect disturbance; that is, two L-shaped wall's surround the two left-most rose-petals in the trajectory in order to induce turbulent airflow when the quadrotor approaches, while the fan largely effects the two right petals with an 18 mph source stream of wind. The water bottle inputs stochastic torques into the system at all times. This diversity of model disturbances in various flight regimes offers insight into DNAC's ability to handle learning sudden and brief disturbances. These conditions are unlike experiment 1, where the 18mph wind gusts were relatively constant throughout the entire flight path, and the flight path itself was much smaller. Moreover, because the flight regime in experiment 2 is larger, the quadrotor will be commanded to deviate further from the linearized hover state. This leads to the hypothesis that DNAC, a nonlinear controller, should outperform DMRAC, a linear controller, in a more obvious manner.

Experiment 1 and [19] both conclude DMRAC outperforms PID and MRAC in a quadrotor autopilot application. For this reason, and for brevity, these control schemes are left out of the experiment 2 analysis. Similar to Experiment 1, DNAC is initialized with random weights and has not been further tuned. This is a key point in the analysis of DNAC's ability to adapt to a variety of situations with little to no software/algorithm change. The position and attitude tracking RMSE results of the four-petal rose experiment are provided in Figure 4.

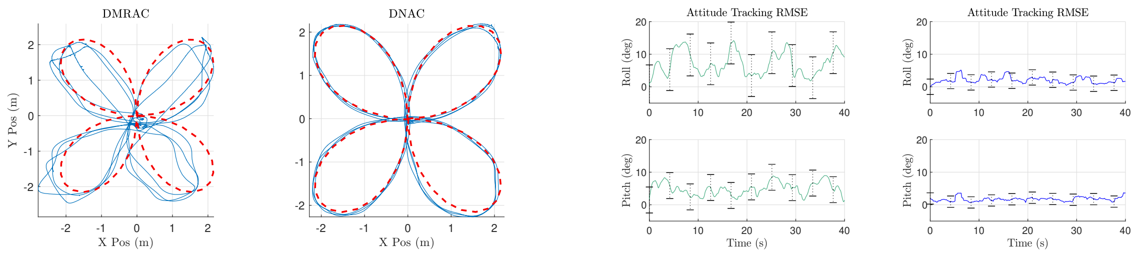

Figure 4 **Far left: DMRAC position tracking. Middle left: DNAC position tracking. Middle right: DMRAC attitude tracking RMSE and standard deviation. Far right: DNAC attitude tracking RMSE and standard deviation. In the position tracking plots, the wind disturbance field originates in the positive-X negative-Y quadrant blowing upward, while the two L-shaped walls are located in the corners of the trajectory in the negative-X quadrant.**

With this new flight path, DMRAC has a difficult time handling the variety of sudden disturbances, and performs worse than when the disturbances remained more constant over time. This is not the case for DNAC as it learns to reject each of the disturbances quickly while moving across them. From Table 4, it is observed that DNAC tracks desired attitude with greater accuracy than both DRMAC from experiment 2, as well as DNAC from experiment 1; specifically, DNAC tracks the four-petal rose pattern with an attitude error L2 norm of 2.10°. This is a 6.21° improvement over DMRAC. Moreover, the standard deviation of the attitude error (provided in Table 5) for DNAC shows substantial improvement over DMRAC. With respect to experiment 1, DNAC exhibits a 0.96° decrease in attitude tracking RSME as well as a decrease in standard deviation. These performance boosts may in part be due to the additional system modes being excited along the larger and more complex flight path.

Controller	$\|\theta, \phi\|_2(^\circ)$	$\|x, y\|_2(\mathrm{cm})$	$\|\dot{x}, \dot{y}\|_2(\mathrm{cm/s})$
DMRAC	8.31	70.0	49.2
DNAC	2.10	47.0	19.1

TABLE 4 **Average tracking error L_2 norms for flight data taken while the drone tracks a four-petal rose trajectory in Experiment 2.**

Controller	$\sigma_\phi(^\circ)$	$\sigma_\theta(^\circ)$
DMRAC	6.41	3.98
DNAC	2.35	1.73

TABLE 5 **Standard deviation of the attitude tracking error for each controller configuration in Experiment 2.**

Remark 5 *The disturbances present in both experiments are significant, and force standard PID autopilots off the reference path by a relatively constant bias which it cannot compensate for. Counter to this behavior, trajectory tracking performance for DNAC seems to be improving over time. As the inner layer features are trained on a slower timescale, tracking bias is removed on each cycle of the flight path. This is clear from the position tracking plots where one see's (forward in time) a more tightly followed reference trajectory.*

V. CONCLUSION

Linear model-based rigid body controllers demonstrate drawbacks for aerospace systems which can be remedied through nonlinear adaptive control techniques. Background was provided for recent advancements in the literature which introduce Deep Neural Network based adaptive control techniques with stability guarantees – namely, DMRAC & DNAC. The control laws governing DNAC were then presented, and an augmented attitude control system was devised for a quadrotor. Flight experiments were run in which a variety of attitude controller performances were compared while operating in a highly uncertain flight environment (see Figure 2). These experiments demonstrate that DNAC outperforms other controllers with an ability to easily learn and reject undesirable model disturbances without a reliance on prior training data or prior identification of some set of system parameters. This paper 1) demonstrated the design of an ML based controller capable of leveraging the powerful model estimation ability of a DNN whilst maintaining stability guarantee's necessary to control a safety-critical system with no reliance on prior training data, and 2) offered an easily configurable and tunable implementation of a deep nonlinear adaptive controller which can run in parallel to existing control structures and improve performance even at the lowest levels of the feedback loop. DNAC achieves this performance while only requiring basic knowledge about the system and simple tuning processes, showing the potential of the DNAC architecture to improve performance in other safety critical systems operating in highly uncertain environments.

References

[1] Chu, C.-K., Yu, G.-R., Jonckheere, E., and Youssef, H., "Gain scheduling for fly-by-throttle flight control using neural networks," *Proceedings of 35th IEEE Conference on Decision and Control*, Vol. 2, 1996, pp. 1557–1562 vol.2. https://doi.org/10.1109/CDC.1996.572744.

[2] Mix, D., Koenig, J., Linda, K., Cifdaloz, O., Wells, V., and Rodriguez, A., "Towards gain-scheduled H/sup /spl infin// control design for a tilt-wing aircraft," *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*, Vol. 2, 2004, pp. 1222–1227 Vol.2. https://doi.org/10.1109/CDC.2004.1430208.

[3] Abbas-Turki, M., Duc, G., Clement, B., and Theodoulis, S., "Robust gain scheduled control of a space launcher by introducing LQG/LTR ideas in the NCF robust stabilisation problem," *2007 46th IEEE Conference on Decision and Control*, 2007, pp. 2393–2398. https://doi.org/10.1109/CDC.2007.4434356.

[4] Nan, F., Sun, S., Foehn, P., and Scaramuzza, D., "Nonlinear MPC for Quadrotor Fault-Tolerant Control," *IEEE Robotics and Automation Letters*, Vol. 7, No. 2, 2022, pp. 5047–5054. https://doi.org/10.1109/LRA.2022.3154033.

[5] Faessler, M., Franchi, A., and Scaramuzza, D., "Differential Flatness of Quadrotor Dynamics Subject to Rotor Drag for Accurate Tracking of High-Speed Trajectories," *IEEE Robotics and Automation Letters*, Vol. 3, No. 2, 2018, pp. 620–626. https://doi.org/10.1109/LRA.2017.2776353.

[6] Jia, J., Guo, K., Yu, X., Zhao, W., and Guo, L., "Accurate High-Maneuvering Trajectory Tracking for Quadrotors: A Drag Utilization Method," *IEEE Robotics and Automation Letters*, Vol. 7, No. 3, 2022, pp. 6966–6973. https://doi.org/10.1109/LRA.2022.3176449.

[7] Dantu, S., Yadav, R. D., Roy, S., Lee, J., and Baldi, S., "Adaptive Artificial Time Delay Control for Quadrotors under State-dependent Unknown Dynamics," *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2022, pp. 1092–1097. https://doi.org/10.1109/ROBIO55434.2022.10011673.

[8] Goodfellow, I., Bengio, Y., and Courville, A., *Deep learning*, MIT press, 2016.

[9] Song, Y., Steinweg, M., Kaufmann, E., and Scaramuzza, D., "Autonomous Drone Racing with Deep Reinforcement Learning," *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 1205–1212. https://doi.org/10.1109/IROS51168.2021.9636053.

[10] Hwangbo, J., Sa, I., Siegwart, R., and Hutter, M., "Control of a Quadrotor With Reinforcement Learning," *IEEE Robotics and Automation Letters*, Vol. 2, No. 4, 2017, pp. 2096–2103. https://doi.org/10.1109/LRA.2017.2720851.

[11] Joshi, G., and Chowdhary, G., "Deep Model Reference Adaptive Control," *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 4601–4608. https://doi.org/10.1109/CDC40024.2019.9029173.

[12] Sun, R., Greene, M. L., Le, D. M., Bell, Z. I., Chowdhary, G., and Dixon, W. E., "Lyapunov-Based Real-Time and Iterative Adjustment of Deep Neural Networks," *IEEE Control Systems Letters*, Vol. 6, 2022, pp. 193–198. https://doi.org/10.1109/LCSYS.2021.3055454.

[13] Stone, M. H., "The Generalized Weierstrass Approximation Theorem," *Mathematics Magazine*, Vol. 21, No. 4, 1948, pp. 167–184. URL http://www.jstor.org/stable/3029750.

[14] Kingma, D. P., and Ba, J., "Adam: A Method for Stochastic Optimization," *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, edited by Y. Bengio and Y. LeCun, 2015. URL http://arxiv.org/abs/1412.6980.

[15] Jackson, J., Ellingson, G., and McLain, T., "ROSflight: A lightweight, inexpensive MAV research and development tool," *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2016, pp. 758–762. https://doi.org/10.1109/ICUAS.2016.7502584.

[16] Anderson, M., Ramos, J. H., Dickinson, B., Brink, K., and Ganesh, P., "The REEF Autonomous Vehicles Laboratory: A collaboration hub expediting flight capabilities for the US Department of defense and academic research," *IEEE Control Systems Magazine*, Vol. 43, No. 4, 2023, pp. 63–83.

[17] Ramos, J. H., Ganesh, P., Warke, W., Volle, K., and Brink, K., "REEF Estimator: A Simplified Open Source Estimator and Controller for Multirotors," *2019 IEEE National Aerospace and Electronics Conference (NAECON)*, IEEE, 2019, pp. 606–613.

[18] Mahony, R., Hamel, T., and Pflimlin, J.-M., "Nonlinear Complementary Filters on the Special Orthogonal Group," *IEEE Transactions on Automatic Control*, Vol. 53, No. 5, 2008, pp. 1203–1218. https://doi.org/10.1109/TAC.2008.923738.

[19] Joshi, G., Jasvir, V., and Chowdhary, G., "Design and Flight Evaluation of Deep Model Reference Adaptive Controller," *AIAA SciTech Forum*, 2020. https://doi.org/10.2514/6.2020-1336.