Asynchronous Deep Model Reference Adaptive Control

Girish Joshi[*]
girishj2@illinois.edu

Jasvir Virdi[**]
jvirdi2@illinois.edu

Girish Chowdhary[†]
girishc@illinois.edu

[*]Department of Aerospace Engineering
[**]Department of Mechanical Science and Engineering
[†]Department of Computer Science & Department of Agricultural and Biological Engineering
University of Illinois at Urbana-Champaign, United States.

Abstract: In this paper, we present Asynchronous implementation of Deep Neural Network-based Model Reference Adaptive Control (DMRAC). We evaluate this new neuro-adaptive control architecture through flight tests on a small quadcopter. We demonstrate that a single DMRAC controller can handle significant nonlinearities due to severe system faults and deliberate wind disturbances while executing high-bandwidth attitude control. We also show that the architecture has long-term learning abilities across different flight regimes, and can generalize to fly different flight trajectories than those on which it was trained. These results demonstrating the efficacy of this architecture for high bandwidth closed-loop attitude control of unstable and nonlinear robots operating in adverse situations. To achieve these results, we designed a software+communication architecture to ensure online real-time inference of the deep network on a high-bandwidth computation-limited platform. We expect that this architecture will benefit other deep learning in the closed-loop experiments on robots.

Keywords: Adaptive Control, Deep Learning, Lyapunov stability, Flight systems

1 Introduction

Learning to control mobile robots online on computation limited platforms while ensuring that the learning transients do not destabilize the system in the presence varied disturbances and operating environments is a challenging problem. The key challenge is that robot dynamics or environmental conditions can significantly change during operation. For example, for a flying robot, uncertainty can appear in environment variables, payloads, system degradation, failures and so on. When such changes happen, heuristic, hand-crafted, or model-based controllers fail to achieve the control objective. Learning based controllers hold the promise to be more robust. However, traditional methods such as reinforcement learning (RL) can fail to produce sensible behavior on real robots, especially when the robot dynamics changes beyond what the RL agent was trained on. On the other hand, traditional adaptive controllers are designed to change their behavior *online and in real-time* to adapt to changes while guaranteeing system stability. Such controllers have been studied extensively in the controls community [1, 2], however, a key problem with these traditional methods is the lack of long-term learning. More recently, Model Reference Adaptive Controllers (MRAC) using shallow networks have demonstrated some long term learning [3, 4], however, the results have been limited to mild changes in the dynamics and in general limited by the learning ability of shallow networks. Yet, their capability to adapt, stability guarantees, and computational efficiency have enabled these methods to emerge as the leading method for learning-based flight control, including for highly unstable rotorcraft with a number of real-world demonstrations [5, 6, 4, 7, 8]. A key drawback of these existing methods has been the lack of long-term learning: The shallow networks in

the above mentioned methods produce a policy which optimize a Lyapunov loss function and hence can instantaneously adapt to mitigate the disturbance, but do not generalize to similar disturbances or operating conditions [9, 6, 3]. Deep Neural Networks (DNNs) trained with dropouts and batch updates could potentially help alleviate these short-term learning issues [10], but it has been difficult to train and update these networks in real-time on aerial robots with limited onboard computing while guaranteeing stability in presence of significant faults.

1.1 Contributions and Significance

Our main contribution in this paper are algorithms and a hardware-software architecture for a real-time updated deep learning based adaptive flight controller for dynamic robots. The system is implemented on a small quadrotor (Parrot Mambo Fly) with limited onboard compute. The system utilizes Deep MRAC algorithm, with contribution here being a novel fast-slow asynchronous update framework where the last layer of the network is updated online on the embedded controller on-board the aerial robot, and the deeper layers of the network are updated on a separate computer using batch updates and dropouts. Both the fast update computing on-board embedded computer, and the batch update computing off-board controllers communicate with each other in an intermittent (asynchronous) fashion, with the off-board computer intermittently sending updated inner layer parameters to the on-board computer and can tolerate significant communication dropouts.

The major significance of our contributions is that we demonstrate that when implemented using our fast-slow update strategy, deep learning based controllers hold tremendous promise for high-bandwidth adaptive control of agile robots with limited onboard compute. Our results show not only a fully-connected deep network can be included in the closed loop and updated asynchronously in a way such that the real-time flight control is possible, but also that a control framework that utilizes DNNs can adapt to handle significant disturbances such as wind bias and damage. We expect that our system can be adopted for other deep learning based controllers, including reinforcement learning and MPC, thereby enabling the incorporation of deep learning for robot control.

1.2 Background: Attitude Control of Quadrotor Aerial Robots

Attitude control of aerial robots is a challenging problem due to the highly nonlinear, unstable, and fast dynamics [11]. Attitude controllers require fast update rates (typically over 100 Hz), furthermore, adaptation is necessary because the flight dynamics can significantly change due to degradation, change in operating conditions, or failures [12, 13]. Due to the unstable dynamics and highly variable operating conditions, learning based control has been hard for quadrotors. One body of methods that have had some success is Model Reference Adaptive Control (MRAC) [5, 6, 12, 13, 14, 15, 16]. The key idea in MRAC is to augment a baseline (linear) flight controller, with a learning element that is updated online to improve tracking performance while ensuring Lyapunov stability. The benefit of this approach is that Lyapunov stability can be proven, and the learning problem can be extended to policy gradient like updated with a Lyapunov constraint. However, the stability results have only held for shallow learning elements, which are limited in their generalization and learning capabilities. Unlike their shallow counterparts deep networks learn features by learning the weights of nonlinear compositions of weighted features arranged in a directed acyclic graph [17]. This ability to learn features can help deep networks learn and generalize across similar examples in supervised learning problems [18]. However, recent success of deep learning has required significant amount of training with large data on powerful computers. This has made it hard for methods that use deep learning to be implemented for control of robots where learning has to be online, in real-time, and on low-power embedded computers on small robots. This is one reason real-world demonstrations controllers with DNNs, such as deep RL or neuro adaptive controllers have been largely missing, especially on challenging systems like attitude control of quadrotors. Specifically, there have been methods in which Deep Reinforcement Learning (DRL) has been used to generate trajectories when the inner loop attitude control was achieved with PIDs [19, 20, 21, 22]. A notable early result was Ng. et al's (non deep) RL in the outerloop for inverted flight of a rotorcraft [23] with the inner loop attitude control executed with LQR. However, such learning has not generalized to learning to control the much faster and unstable attitude loop on real aircraft [24]. Ideas of separately training different layers of the network asynchronously have been used in computer vision [25], however, not yet for closed-loop attitude control. The goal of this paper is to alleviate this bottleneck by creating a system that enables the use of deep learning based controllers on low-power robots with fast unstable dynamics..

2

2 Flight Control System Description

2.1 Flight control problem formulation

The attitude control system of a quadrotor subject to unknown disturbances can be abstracted as the below with the nonlinear and unknown function $\Delta(x)$:

$$\dot{x}(t) = Ax(t) + B(u(t) + \Delta(x)). \tag{1}$$

where $x(t) \in \mathbb{R}^n$ is the state vector, $u(t) \in \mathbb{R}^m, \forall t \geqslant 0$ is the control input, $A \in \mathbb{R}^{n \times n}, B \in \mathbb{R}^{n \times m}$ are known system matrices and we assume the pair (A, B) is controllable. The term $\Delta(x) : \mathbb{R}^n \to \mathbb{R}^m$ is matched system uncertainty and is assumed to be Lipschitz continuous in $x(t) \in \mathcal{D}_x$. $\mathcal{D}_x \subset \mathbb{R}^n$ is assumed to be a compact set and the control $u(t)$ is assumed to belong to a set of admissible control inputs of measurable and bounded functions [1, 2].

The flight control task is to have the closed-loop system follow a reference trajectory. There are many ways to generate the reference trajectory, including parameterizing it with polynomials or other smooth functions. Here we generate the reference trajectory using a linear reference model. This is a preferred approach in flight control because the desired transient and steady-state performance can be defined by a selecting the reference model's eigenvalues in the negative half plane [26]. The desired closed-loop response of the reference system is given by

$$\dot{x}_{rm}(t) = A_{rm}x_{rm}(t) + B_{rm}r(t). \tag{2}$$

where $x_{rm}(t) \in \mathcal{D}_x \subset \mathbb{R}^n$ and $A_{rm} \in \mathbb{R}^{n \times n}$ is Hurwitz and $B_{rm} \in \mathbb{R}^{n \times r}$. Furthermore, the command $r(t) \in \mathbb{R}^r$ denotes a bounded, piece wise continuous, reference signal and we assume the reference model (2) is bounded input-bounded output (BIBO) stable [1].

The uncertainty function $\Delta(x)$ is unknown, but it is assumed to be piece-wise-continuous over a compact domain $\mathcal{D}_x \subset \mathbb{R}^n$, which is a reasonable assumption for flight control problems [26, 8, 27]. We use a Deep Neural Networks (DNN) model to learn online a representation for this unknown function. Using DNNs, a non linearly parameterized network estimate of the uncertainty can be written as $f_\theta(x) \triangleq \theta_n^T \Phi(x)$, where $\theta_n \in \mathbb{R}^{k \times m}$ are network parameters for the final layer and $\Phi(x) = \phi_n(\theta_{n-1}, \phi_{n-1}(\theta_{n-2}, \phi_{n-2}(...))))$, is a k dimensional feature vector which is function of inner layer weights, activations and inputs. The basis vector $\Phi(x) \in \mathcal{F} : \mathbb{R}^n \to \mathbb{R}^k$ is considered to be Lipschitz continuous to ensure the existence and uniqueness of the solution (1).

2.2 Adaptive Controller

The aim of the adaptive controller is to construct a feedback law $u(t)$ such that the state of the uncertain dynamical system (1) asymptotically tracks the state of the reference model (2) in-spite of the unknown function $\Delta(x)$. In our formulation, an adaptive controller is designed to augment an existing baseline linear controller. Hence, the total control action consists of a linear feedback term $u_{pd} = Kx(t)$, a linear feed-forward term $u_{crm} = K_r r(t)$ and an adaptive term $\nu_{ad}(t)$:

$$u = u_{pd} + u_{crm} - \nu_{ad}. \tag{3}$$

The baseline full state feedback and feed-forward controller (without the adaptive term ν_{ad}) is designed to ensure $A_{rm} = A - BK$ and $B_{rm} = BK_r$ when $\Delta(x) = 0$. The intuition being that if the adaptive controller can ensure that $\nu_{ad}(t) \approx \Delta(x(t))$, then the closed loop system will essentially perform like the desired reference model. MRAC literature [5, 6, 7, 8] has relied on shallow networks to model the adaptive term ν_{ad}. Here we present a framework to implement the DMRAC using DNN for a quadrotor control under unknown system faults using embedded computers.

3 Deep Model Reference Adaptive Control

Unlike traditional MRAC or even single-hidden-layer neural network based MRAC weight update rule [9, 28, 29], where the network weights are learned in the direction of minimizing the tracking error, training a deep Neural network is much more involved. Feed-Forward networks like DNNs are trained in a supervised manner over a batch of i.i.d data. Deep learning optimization is based on Stochastic Gradient Descent (SGD) or its variants.

The SGD update rule relies on a stochastic approximation of the expected value of the gradient of the loss function over a training set or mini-batches. To train a deep network to estimate the system uncertainties, a batch of i.i.d samples of labeled pairs of state-true uncertainties $\{x(t), \Delta(x(t))\}$ are required. Since we do not have access to true values of $\Delta(x)$, we use the idea of using Model Reference Generative Network (MRGeN), introduced in [30] to create estimates of $\Delta(x)$. This generative network is derived from separating the DMRAC-DNN into inner feature layer and the final output layer of the network. The parameter update of these two layers are temporally separated making it possible to online update the DMRAC, details of which are presented in following sections.

3.1 DMRAC-Online Parameter Estimation law

The last layer of DMRAC-DNN with learned features from inner layer forms the Deep-Model Reference Generative Network (D-MRGeN). We use the classical MRAC learning rule to update point-wise in time the weights of the outermost layer of the DNN in the direction of achieving asymptotic tracking of the reference model by the actual system. This is now explained further in detail.

The adaptive term in the controller (3) with a DNN model can be written as:

$$\nu_{ad}(t) = W^T \phi_n(\theta_{n-1}, \phi_{n-1}(\theta_{n-2}, \phi_{n-2}(...)))). \tag{4}$$

To differentiate the parameters of DMRAC from last layer weights "θ_n" of DNN, we denote DMRAC weights as "W". Recall, the goal of the MRAC controller is to ensure $\nu_{ad} \to \Delta(x)$ as $t \to \infty$.

Assumption 1: Appealing to the universal approximation property of Neural Networks [31] we have that, for every given basis functions $\Phi(x) \in \mathcal{F}$ there exists (unknown) ideal weights $W^* \in \mathbb{R}^{k \times m}$ and $\epsilon_1(x) \in \mathbb{R}^m$ such that the following approximation holds

$$\Delta(x) = W^{*T}\Phi(x) + \epsilon_1(x), \quad \forall x(t) \in \mathcal{D}_x \subset \mathbb{R}^n \tag{5}$$

The network approximation error $\epsilon_1(x)$ is upper bounded, s.t $\bar{\epsilon}_1 = \sup_{x \in \mathcal{D}_x} \|\epsilon_1(x)\|$, and can be made arbitrarily small given sufficiently large number of basis functions [32]

We can define the reference model tracking error as $e(t) = x_{rm}(t) - x(t)$. Using (1) & (2) and the controller of form (3) with adaptation term ν_{ad} (4), the tracking error dynamics can be written as

$$\dot{e}(t) = A_{rm}e(t) + \tilde{W}^T\Phi(x) + \epsilon_1(x). \tag{6}$$

where $\tilde{W} = W^* - W$ is error in parameter. The estimate of the unknown true network parameters W^* are calculated on-line using the weight update rule (7); correcting the weight estimates in the direction of minimizing the instantaneous tracking error $e(t)$. The resulting update rule for network weights in estimating the total uncertainty in the system is as follows [33, 1, 2],

$$\dot{W} = \Gamma proj(W, \Phi(x)e(t)'P), \qquad W(0) = W_0 \tag{7}$$

where $proj$ is a projection operator, $\Gamma \in \mathbb{R}^{k \times k}$ is the learning rate and $P \in \mathbb{R}^{n \times n}$ is a positive definite matrix such that for a given Hurwitz A_{rm}, the matrix $P \in \mathbb{R}^{n \times n}$ is a positive definite solution of Lyapunov equation $A_{rm}^T P + PA_{rm} + Q = 0$ for given $Q > 0$.

Assumption 2: For uncertainty parameterized by unknown true weight $W^* \in \mathbb{R}^{k \times m}$ and known nonlinear basis $\Phi(x)$, the ideal weight matrix is assumed to be upper bounded s.t $\|W^*\| \leq \mathcal{W}_b$.

Theorem 1 *Consider the actual and reference plant model (1) & (2). If the weights parameterizing total uncertainty in the system are updated according to identification law (7) Then the tracking error $\|e\|$ and error in network weights $\|\tilde{W}\|$ are bounded for all $\Phi \in \mathcal{F}$.*

The proof of above theorem is provided in the Appendix-A.2.

From Theorem-1 & (6) and using system theory [34] we can infer that as $e(t) \to 0$, and $\nu_{ad} \to \Delta(x)$ in point-wise sense. Hence uncertainty estimates $y_\tau = \nu_{ad}(x_\tau)$ are admissible target values for training DNN features over the data $Z^M = \{\{x_\tau, y_\tau\}\}_{\tau=1}^M$.

The details of DNN training and implementation details of DMRAC controller is presented in the following section.

4

3.2 Deep Feature Training for DMRAC controller

This section provides the details of the DNN training over data samples observed over n-dimensional input subspace $x(t) \in \mathcal{X} \in \mathbb{R}^n$ and m-dimensional target subspace $y \in \mathcal{Y} \in \mathbb{R}^m$. The sample set is denoted as $\mathcal{Z}$ where $\mathcal{Z} \in \mathcal{X} \times \mathcal{Y}$.

We are interested in the function approximation task for DNN. Let the function $f_{\boldsymbol{\theta}}(x) = \theta_n \phi_n(\theta_{n-1}, \phi_{n-1}(\theta_{n-2}, \phi_{n-2}(...))))$ s.t $f_{\boldsymbol{\theta}} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ be the network approximating the model uncertainty with parameters $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, where $\boldsymbol{\Theta}$ is the space of parameters. We assume a training data buffer $\mathcal{B}$ can store p_{max} training examples, such that the set $Z^{p_{max}} = \{Z_i | Z_i \in \mathcal{Z}\}_{i=1}^{p_{max}} = \{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}_{i=1}^{p_{max}}$. A batch of samples can be drawn independently from the buffer $\mathcal{B}$ over probability distribution P for DNN training.

The loss function, which measures the discrepancy between true target y and algorithm's estimated target function value $f_{\boldsymbol{\theta}}$ is denoted by $L(y, f_{\boldsymbol{\theta}}(x))$. Specific to work presented in this paper, we use a ℓ_2-norm between values i.e. $\mathbb{E}_p(\ell(y, f_{\boldsymbol{\theta}}(x))) = \mathbb{E}_P(\|y_i - f_{\boldsymbol{\theta}}(x_i)\|_2)$ as loss function for DNN training. An empirical loss $L(\boldsymbol{Z}, \boldsymbol{\theta}) = \frac{1}{M}\sum_i^M \ell(\boldsymbol{Z_i}, \boldsymbol{\theta})$ is used to approximate the loss function since the true distribution P is unknown to learning algorithm. The network parameters are updated using SGD in the direction of minimizing the loss function as follows

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \frac{1}{M}\sum_i^M \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}). \tag{8}$$

Unlike the conventional DNN training where the true target values $y \in \mathcal{Y}$ are available for every input $x \in \mathcal{X}$, in DMRAC true system uncertainties as the labeled targets are not available for the network training. We use the part of the network itself (the last layer) with pointwise weight updated according to MRAC-rule as the generative model for the data. The DMRAC uncertainty estimates $y = W^T \Phi(x, \theta_1, \theta_2, \dots \theta_{n-1})$ along with inputs x_i make the training data set $Z^{p_{max}} = \{x_i, \Delta'(x_i)\}_{i=1}^{p_{max}}$. Note that we use interchangably x_i and $x(t)$ as discrete representation of continuous state vector for DNN training. The main purpose of DNN in the adaptive network is to extract relevant features of the system uncertainties, which otherwise is very tedious to obtain with unknown bounds of the domain of operation.

We also analyse the features through data visualization techniques [35] and demonstrate empirically, that the DNN features trained over past i.i.d representative data retains the memory of the past instances and can be used as the frozen feed-forward network over similar reference tracking tasks without loss of the guaranteed tracking performance.

3.3 Recording Data using MRGeN for DNN Training

The DNN in DMRAC controller is trained over training dataset $Z^M = \{x_i, y_i\}_{i=1}^M$, where the y_i are fast updating outer layer of DNN estimates of the uncertainty. The training dataset Z^M is randomly drawn from a larger data buffer $\mathcal{B}$. Not every pair of data $\{x_i, y_i\}$ from generative network is added to the training buffer $\mathcal{B}$. We qualify the input-target pair based on kernel independence test to ensure that we collect locally exciting independent information which provides a sufficiently rich representation of the operating domain. Since the state-uncertainty data is the realization of a Markov process, such a method for qualifying data to be sufficiently independent of previous data-points is necessary. The algorithm details to qualify and add a data point to the buffer is provided in detail in [36].

4 DMRAC Flight control Experimental Setup

This section provides the hardware and vehicle details for evaluation of DMRAC for flight control of quadrotor system. The flight tests were done on a commercially available quadcopter platform, Parrot Mambo Mini-Drone in a VICON facility. This drone was chosen for flight testing due to its small size, low cost, and relative ease of implementing and testing algorithms. The implementation of DMRAC was done using our onboard, off-board architecture as shown in Figure-1. The off-board component consists of software modules which run entirely on a host computer and are responsible for training the neural network using data received from the drone via UDP bluetooth protocol. The data received are the drone's current roll, pitch, yaw angles and the angular rates, and onboard

---

Algorithm 1 D-MRAC Controller Training

1: **Input:** $\Gamma, \eta, \zeta_{tol}, p_{max}$
2: **while** New measurements are available **do**
3:     Update the DMRAC weights W using Eq:(7)
4:     Compute $y_{\tau+1} = \hat{W}^T \Phi(x_{\tau+1})$
5:     Given $x_{\tau+1}$ compute $\gamma_{\tau+1}$ [36].
6:     **if** $\gamma_{\tau+1} \geqslant \zeta_{tol}$ **then**
7:         Update $\mathcal{B} : \boldsymbol{Z}(:) = \{x_{\tau+1}, y_{\tau+1}\}$ and $\mathbb{X} : \Phi(x_{\tau+1})$
8:         **if** $|\mathcal{B}| > p_{max}$ **then**
9:             Delete element in $\mathcal{B}$ by SVD maximization [36]
10:         **end if**
11:     **end if**
12:     **if** $|\mathcal{B}| \geq M$ **then**
13:         Sample a mini-batch of data $\boldsymbol{Z}^M \subset \mathcal{B}$
14:         Train the DNN network over mini-batch data using Eq-(8)
15:         Update the feature vector Φ for D-MRGeN network
16:     **end if**
17: **end while**

---

estimate of the total uncertainty using the features provided from off-board computer. The above information is stored in a memory buffer and random batches from this buffer are used to train the deep neural network on the off-board computer. Moreover, the host computer also receives drone's current position inside the Vicon Arena from the Vicon system via Wi-Fi which is then communicated to the drone using UDP protocol. Also, as soon as the inner layer network parameters are
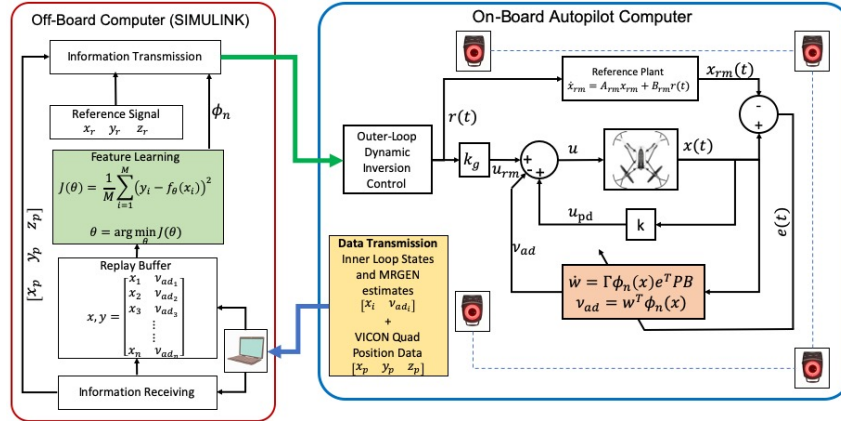

Figure 1: Our On-board - Off-board Implementation of Deep Model reference Adaptive controller for Quadrotor control. The system ensures that the outer layer weights can be updated onboard in a manner to ensure Lyapunov stability (at 200Hz), while the inner layer weights are updated asynchronously off-board to improve learning performance.

updated using the data collected from the drone, the host computer communicates the updated inner-layer parameters back to the vehicle, which are used in updating the DMRAC controller to produce adaptive torque. The onboard software modules are responsible for state estimation along with computing the entire control effort. The main reason behind spitting the controller into on-board and off-board module is because of the limited processing power of the drone. It is possible that if a more powerful processor can be embedded on-board, the algorithm can be implemented completely on the vehicle. Even in this case however, it is desirable to separate the inner layer learning so that the outer layer weights can be updated to ensure Lyapunov stability and inner layer weight only updated whenever the test data loss values are significantly high. Furthermore, in this case, an interesting extension could be to use the presented on-board - off board architecture to asynchronously update the inner-layer weights using data from multiple drones simultaneously. However, this is left for future work.

5 Fault tolerance: Rotor blade chipping in mid-flight

To test the fault-tolerance capability of the controllers, we test and compare the performance of PID, MRAC, and DMRAC when rotor chipping occurs during mid-flight. The quadrotor is commanded to hover at $1m$ above the ground. The chipped blade breaks off due to outward centrifugal force and causes sudden fault into the system. Since this is not a controlled fault, to ensure the reliability of the controller, we report controller performance over multiple runs. The results presented in Fig-3 clearly shows that DMRAC outperforms PID and MRAC. In the case of PID, only two runs were carried out, since in both cases, the drone underwent severe oscillation and crashed. Tests conclusively demonstrated that PID is not capable of handling such extreme faults in the system even with extensive tuning (Flight video for PID, MRAC & DMRAC[1]). In the case of MRAC and DMRAC, eight flight tests are carried out. Out of eight test runs, MRAC managed to control only in four flights. Whereas no failure were observed in the case of DMRAC. Also, on comparing flights where no crash occurred, one can see that MRAC produces poor reference tracking when compared to DMRAC. The Figure-2 and 3 show for successful flights (8 flights for DMRAC and 4 for MRAC) mean and variance of the reference tracking in the x-y-z position for each algorithm. The dotted lines in MRAC plots are the state history for failed flights. Additional results with nonlinear disturbance, sim-to-real transfer and learning retention experiments are provided in Appendix-A.3.
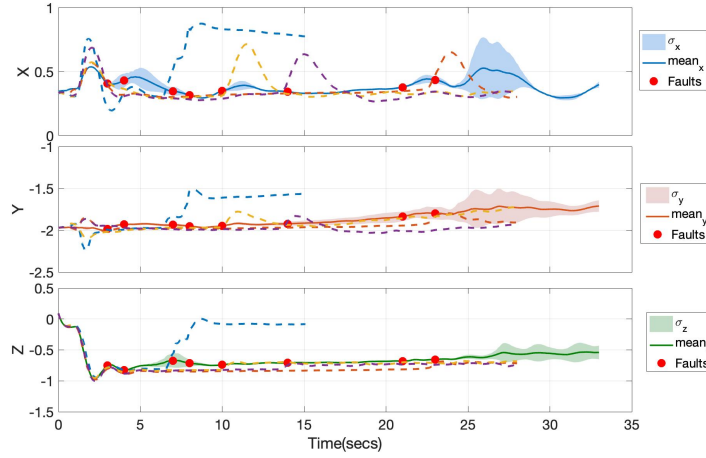

Figure 2: MRAC Trajectory tracking performance in X-Y-Z under system fault for eight flight test. Out of eight flights we observe four times the quadrotor either crashed or produced bad tracking
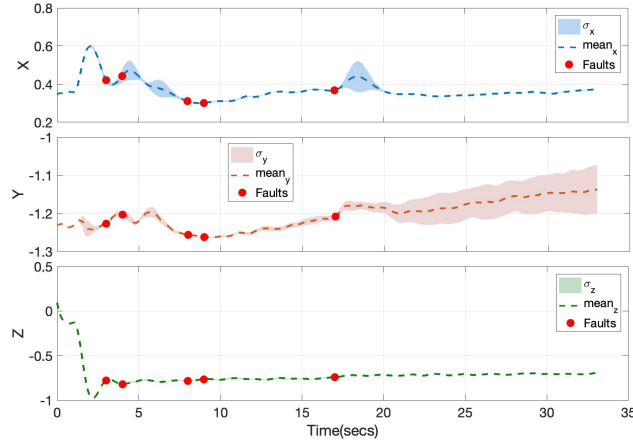

Figure 3: DMRAC Trajectory tracking performance in X-Y-Z under system fault for Eight flight test (Red dot: Time at which Fault occurred)

---

[1]https://girishvjoshi.github.io/gj_blog/dmrac/update/2019/12/18/Deep-Model-Reference-Adaptive-Control

6 Feature Analysis of DMRAC controller

Analyzing a deep network success in performing inference tasks is mathematically not feasible and has become a challenging question. We often treat the deep neural network models as black boxes without clear understanding on internal mechanics. Understanding how the network is able to store and present adaptive controller the appropriate features for the appropriate flight regime is a hard question. As a result, we face a challenge that demand visualizing these high dimension network data for better understanding and analyzing machine learning models, especially their inner working mechanism.

Principle Component Analysis(PCA) [37] and t-Distributed Stochastic Neighborhood Embedding (t-SNE) [35] are two very important techniques in the point based visualization of higher dimensional data in to lower dimensional feature vectors. These methods will help us explore and analyze the relationship between the cluster in the deep feature for different flight regime the network has seen through training.

We conducted the flight test with varying wind bias cases and rotor breaking case. We deployed a DMRAC network to control a physical quadrotor tracking figure-8 reference signal under wind bias and Rotor fault. The same DNN is trained progressively over the ensemble of data sampled from Low, Medium, High wind bias and Fault tolerant flight cases. Figure-4b show the t-SNE
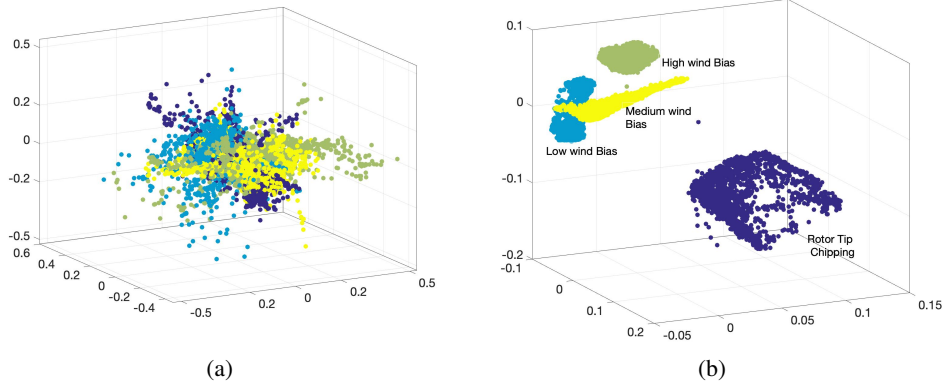

Figure 4: t-SNE visualization of DMRAC features (a) Before training and (b) After training to handle Low, Medium, High Wind Bias and Rotor Tip breaking case

visualization of the $20-$dimensional feature vector in $3-$dimensional scatter-plot. In Figure-4b each point denote one feature representation of test state from either of the experiments. As shown in the Figure-4, after training, the spatial separation between classes is significantly improved. This observation provides evidence to our hypothesis that neural networks learn to detect representations that are useful for different flight regimes and uncertainties experienced by the vehicle. The feature visualization also helps with the understanding of similar flight regimes. As we can observe the windy and Fault cases are classified as two very distinct clusters. But the wind cases from Low to Medium to High wind case cluster are together but progressively move away from each other as the wind disturbance is increased.

7 Conclusion

In this paper, we presented a system to implement DMRAC adaptive controllers using model reference generative network architecture. We demonstrated that our fast-slow architecture utilizing asynchronous onboard and off-board processing can be used to incorporate deep learning in the closed-loop for high-bandwidth flight control of unstable aircraft in the presence of significant disturbances. The results clearly show that when utilized in the closed loop, DMRAC can provide significant performance and generalization benefits over shallow MRAC and PIDs. The results are significant, not only for flight control, but for other robotic control applications involving deep learning. This is because our approach of separating the learning in asynchronous manner can be adopted to other learning based controllers, including learning based MPC and reinforcement learning.

8

Acknowledgments

References

[1] P. Ioannou and J. Sun. Theory and design of robust direct and indirect adaptive-control schemes. *International Journal of Control*, 47(3):775–813, 1988.

[2] G. Tao. *Adaptive control design and analysis*, volume 37. John Wiley & Sons, 2003.

[3] G. Chowdhary, H. A. Kingravi, J. P. How, and P. A. Vela. Bayesian Nonparametric Adaptive Control Using Gaussian Processes. *Neural Networks and Learning Systems, IEEE Transactions on*, PP(99):1, 2014. ISSN 2162-237X. doi:10.1109/TNNLS.2014.2319052.

[4] R. Grande, G. Chowdhary, and J. P. How. Experimental Validation of Bayesian Nonparametric Adaptive Control using Gaussian Processes. *AIAA Journal of Aerospace Information Systems (formerly JACIC)*, 1 2014.

[5] E. N. Johnson and S. Kannan. Adaptive Trajectory Control for Autonomous Helicopters. *Journal of Guidance Control and Dynamics*, 28(3):524–538, 5 2005. ISSN 0731-5090. doi:10.2514/1.6271. URL http://doi.aiaa.org/10.2514/1.6271.

[6] G. Chowdhary, M. Mühlegg, and E. N. Johnson. Exponential parameter and tracking error convergence guarantees for adaptive controllers without persistency of excitation. *International Journal of Control*, 87(8):1583–1604, 2014. doi:10.1080/00207179.2014.880128. URL http://dx.doi.org/10.1080/00207179.2014.880128.

[7] I. Gregory, C. Cao, E. Xargay, N. Hovakimyan, and X. Zou. L1 adaptive control design for nasa airstar flight test vehicle. In *AIAA guidance, navigation, and control conference*, page 5738, 2009.

[8] G. Chowdhary, E. N. Johnson, R. Chandramohan, M. S. Kimbrell, and A. Calise. Guidance and control of airplanes under actuator failures and severe structural damage. *Journal of Guidance, Control, and Dynamics*, 36(4):1093–1104, 2013.

[9] G. Chowdhary and E. N. Johnson. Theory and Flight Test Validation of a Concurrent Learning Adaptive Controller. *Journal of Guidance Control and Dynamics*, 34(2):592–607, 3 2011.

[10] G. Joshi and G. Chowdhary. Deep model reference adaptive control. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 4601–4608. IEEE, 2019.

[11] B. L. Stevens, F. L. Lewis, and E. N. Johnson. *Aircraft control and simulation: dynamics, controls design, and autonomous systems*. John Wiley & Sons, 2015.

[12] E. Johnson and S. Kannan. Adaptive flight control for an autonomous unmanned helicopter. In *Proceedings of the AIAA Guidance Navigation and Control Conference, held at Monterrery CA*, 2002.

[13] M. Steinberg. Historical overview of research in reconfigurable flight control. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 219(4): 263–275, 2005.

[14] B. Michini and J. How. L1 adaptive control for indoor autonomous vehicles: Design process and flight testing. In *AIAA Guidance, Navigation, and Control Conference*, page 5754, 2009.

[15] V. Patel, C. Cao, N. Hovakimyan, K. Wise, and E. Lavretsky. Adaptive Controller for Tailless Unstable Aircraft in the Presence of Unknown Actuator Failures. *International Journal of Control*, 82(4):705–720, 2009.

[16] E. Lavretsky and K. Wise. Flight Control of Manned/Unmanned Military Aircraft. In *Proceedings of American Control Conference*, 2005.

[17] D. Yu, M. L. Seltzer, J. Li, J.-T. Huang, and F. Seide. Feature Learning in Deep Neural Networks - Studies on Speech Recognition Tasks. *arXiv e-prints*, art. arXiv:1301.3605, Jan 2013.

[18] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, B. Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29, 2012.

[19] S. Lupashin, A. Schöllig, M. Sherback, and R. D'Andrea. A simple learning strategy for high-speed quadrocopter multi-flips. In *2010 IEEE international conference on robotics and automation*, pages 1642–1648. IEEE, 2010.

[20] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in neural information processing systems*, pages 1–8, 2007.

[21] P. Abbeel, A. Coates, and A. Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, 2010.

[22] S. Kim. Autonomous cleaning robot: Roboking system integration and overview. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 5, pages 4437–4441. IEEE, 2004.

[23] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental robotics IX*, pages 363–372. Springer, 2006.

[24] J. A. Bagnell and J. G. Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, volume 2, pages 1615–1620. IEEE, 2001.

[25] I. Song, H.-J. Kim, and P. B. Jeon. Deep learning for real-time robust facial expression recognition on a smartphone. In *2014 IEEE International Conference on Consumer Electronics (ICCE)*, pages 564–567. IEEE, 2014.

[26] A. J. Calise, M. Sharma, and S. Lee. Adaptive Autopilot Design for Guided Munitions. *AIAA Journal of Guidance, Control, and Dynamics*, 23(5), 2000.

[27] G. Chowdhary, M. Mühlegg, and E. N. Johnson. Exponential parameter and tracking error convergence guarantees for adaptive controllers without persistency of excitation. *International Journal of Control*, 87(8):1583–1603, 2014.

[28] F. L. Lewis. Nonlinear Network Structures for Feedback Control. *Asian Journal of Control*, 1: 205–228, 1999.

[29] K. Kannan Suresh. *Adaptive Control of Systems in Cascade with Saturation*. PhD thesis, Ph. D. Thesis, Georgia Institute of Technology Atlanta Ga, 2005.

[30] G. Joshi and G. Chowdhary. Adaptive control using gaussian-process with model reference generative network. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 237–243. IEEE, 2018.

[31] J. Park and I. W. Sandberg. Universal approximation using radial-basis-function networks. *Neural computation*, 3(2):246–257, 1991.

[32] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[33] P. A. Ioannou and J. Sun. *Robust Adaptive Control*. Prentice-Hall, Upper Saddle River, 1996. URL http://www-rcf.usc.edu/~ioannou/Robust_Adaptive_Control.htm.

[34] T. Kailath. *Linear systems*, volume 156. Prentice-Hall Englewood Cliffs, NJ, 1980.

[35] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[36] G. Chowdhary and E. Johnson. A singular value maximizing data recording algorithm for concurrent learning. In *Proceedings of the 2011 American Control Conference*, pages 3547–3552, June 2011. doi:10.1109/ACC.2011.5991481.

[37] S. Wold, K. Esbensen, and P. Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[38] G. Larchev, S. Campbell, and J. Kaneshige. Projection operator: A step toward certification of adaptive controllers. In *AIAA Infotech@ Aerospace 2010*, page 3366. 2010.

[39] K. S. Narendra and A. M. Annaswamy. *Stable adaptive systems*. Courier Corporation, 2012.

A Appendices

A.1   Network Partitioning details for On-board and Off-board training.

The DNN architecture for MRAC is trained in two steps. We separate the DNN into two networks as shown in Fig-5a. The faster learning outer adaptive network and slower deep feature network. DMRAC learns underlying deep feature vector to the system uncertainty using locally exciting uncertainty estimates obtained using a generative network. Between successive updates of the inner layer weights, the feature provided by the inner deep network is used as the fixed feature vector for outer layer adaptive network update and evaluation Figure-5b. The algorithm for DNN learning and DMRAC controller is provided in Algorithm-1. Through this architecture of mixing two-time scale learning, we fuse the benefits of DNN memory through retention of relevant, exciting features and robustness, boundedness guarantee in reference tracking. This key feature of the presented framework ensures robustness while guaranteeing long term learning and memory in the adaptive network.
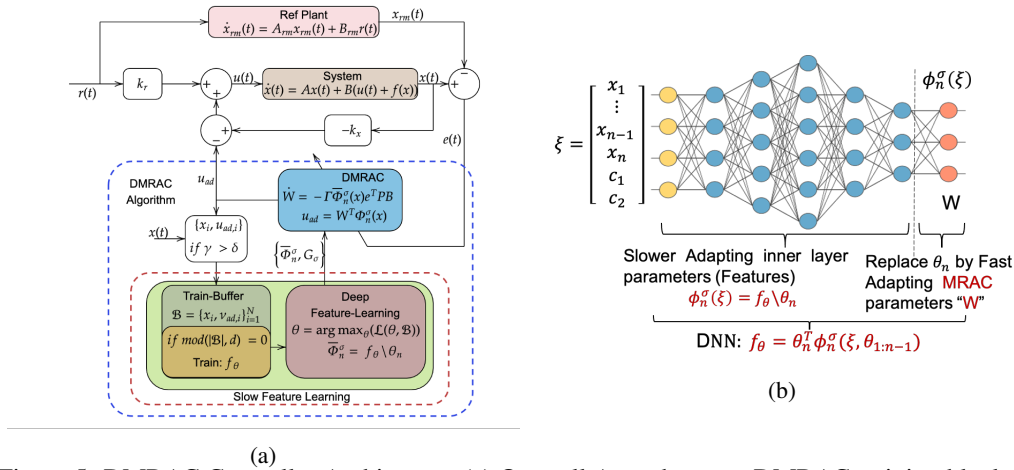

(a)

(b)

Figure 5: DMRAC Controller Architecture (a) Over-all Asynchronous DMRAC training block diagram (b) Network Partitioning scheme for On-board and Off-board update.

Also as indicated in the controller architecture Fig-5a we can use contextual state 'c_i' other than system state $x(t)$ to extract relevant features. These contextual states could be relevant model information not captured in system states. For example for an aircraft system, vehicle parameters like pitot tube measurement, the angle of attack, engine thrust and so on. These contextual states can extract features which help in decision making in case of faults. The work on DMRAC with contextual states will be dealt with in the follow on work.

A.2   Proof of Theorem-1

Theorem-1 Consider the actual and reference plant model (1) & (2). If the weights parameterizing total uncertainty in the system are updated according to identification law (3) Then the tracking error $\|e\|$ and error in network weights $\|\tilde{W}\|$ are bounded for all $\Phi \in \mathcal{F}$.

The feature vectors belong to a function class characterized by the inner layer network weights θ_i s.t $\Phi \in \mathcal{F}$. We will prove the Lyapunov stability under the assumption that inner layer of DNN presents us a feature which results in the worst possible approximation error compared to network with features before switch.

For the purpose of this proof let $\Phi(x)$ denote feature before switch and $\bar{\Phi}(x)$ be the feature after switch. We define the error $\epsilon_2(x)$ as,

$$\epsilon_2(x) = \sup_{\bar{\Phi} \in \mathcal{F}} \left| W^T \bar{\Phi}(x) - W^T \Phi(x) \right| \tag{9}$$

12

Similar to **Fact-1** we can upper bound the error $\epsilon_2(x)$ as $\bar{\epsilon}_2 = \sup_{x \in \mathcal{D}_x} \|\epsilon_2(x)\|$. By adding and subtracting the term $W^T \bar{\Phi}(x)$, we can rewrite the error dynamics (6) with switched basis as,

$$
\begin{aligned}
\dot{e}(t) &= A_{rm} e(t) + W^{*T} \Phi(x) - W^T \Phi(x) \\
&\quad + W^T \bar{\Phi}(x) - W^T \bar{\Phi}(x) + \epsilon_1(x)
\end{aligned} \tag{10}
$$

From **Assumption-1** we know there exists a $W^* \forall \Phi \in \mathcal{F}$. Therefore we can replace $W^{*T} \Phi(x)$ by $W^{*T} \bar{\Phi}(x)$ and rewrite the Eq-(10) as

$$
\dot{e}(t) = A_{rm} e(t) + \tilde{W}^T \bar{\Phi}(x) + W^T (\bar{\Phi}(x) - \Phi(x)) + \epsilon_1(x) \tag{11}
$$

For arbitrary switching, for any $\bar{\Phi}(x) \in \mathcal{F}$, we can prove the boundedness by considering worst possible approximation error and therefore can write,

$$
\dot{e}(t) = A_{rm} e(t) + \tilde{W}^T \bar{\Phi}(x) + \epsilon_2(x) + \epsilon_1(x) \tag{12}
$$

Now lets consider $V(e, \tilde{W}) > 0$ be a differentiable, positive definite radially unbounded Lyapunov candidate function,

$$
V(e, \tilde{W}) = e^T P e + \frac{\tilde{W}^T \Gamma^{-1} \tilde{W}}{2} \tag{13}
$$

The time derivative of the lyapunov function (13) along the trajectory (12) can be evaluated as

$$
\dot{V}(e, \tilde{W}) = \dot{e}^T P e + e^T P \dot{e} - \tilde{W}^T \Gamma^{-1} \dot{\tilde{W}} \tag{14}
$$

Using (12) & (7) in (14), the time derivative of the lyanpunov function reduces to

$$
\dot{V}(e, \tilde{W}) = -e^T Q e + 2 e^T P \epsilon(x) \tag{15}
$$

where $\epsilon(x) = \epsilon_1(x) + \epsilon_2(x)$ and $\bar{\epsilon} = \bar{\epsilon}_1 + \bar{\epsilon}_2$.
Hence $\dot{V}(e, \tilde{W}) \leq 0$ outside compact neighborhood of the origin $e = 0$, for some sufficiently large $\lambda_{min}(Q)$.

$$
\|e(t)\| \geq \frac{2 \lambda_{max}(P) \bar{\epsilon}}{\lambda_{min}(Q)} \tag{16}
$$

Using the BIBO assumption $x_{rm}(t)$ is bounded for bounded reference signal $r(t)$, thereby $x(t)$ remains bounded. Since $V(e, \tilde{W})$ is radially unbounded the result holds for all $x(0) \in \mathcal{D}_x$. Using the fact, the error in parameters $\tilde{W}$ are bounded through projection operator [38] and further using Lyapunov theory and Barbalat's Lemma [39] we can show that $e(t)$ is uniformly ultimately bounded in vicinity to zero solution.

A.3 Additional Results

In this section, we present flight test results that compare the performance of DMRAC algorithm over other popular flight control algorithms like MRAC and PID.

A.3.1 Trajectory tracking under highly non linear wind disturbance

In this experiment, we simulated a highly nonlinear and unpredictable external disturbance on the quadrotor. We attached a piece of cloth underneath the frame of the vehicle, which is then subjected to high wind bias. The erratic flapping of the cloth produces unpredictable disturbance torques and forces on the quadrotor. The experiment is designed to push each controller to their limits and is conducted thrice to demonstrate repeatability. Here, we present the best case performance of each controller. We observe that PID fails very early in the experiment, whereas both the adaptive controllers give a stable performance. However, the tracking error observed for (shallow) MRAC was relatively higher when compared to DMRAC. Refer Figure-6 and Figure-7. Figure-8 shows plots for control torques generated in for each algorithm for the above case.
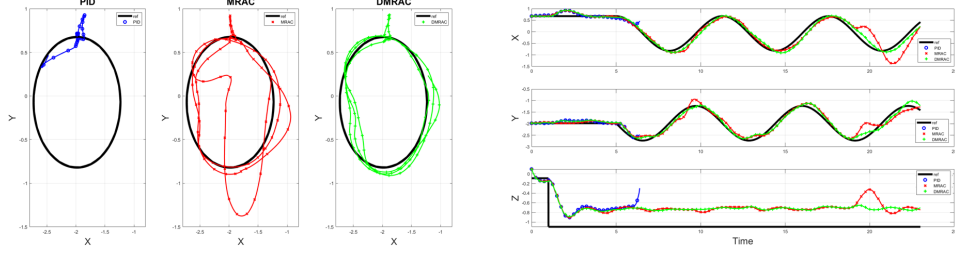
Figure 6: Tracking performance of quadrotor in x-y-z, for a circular trajectory under high wind bias with nonlinear and unpredictable disturbance.
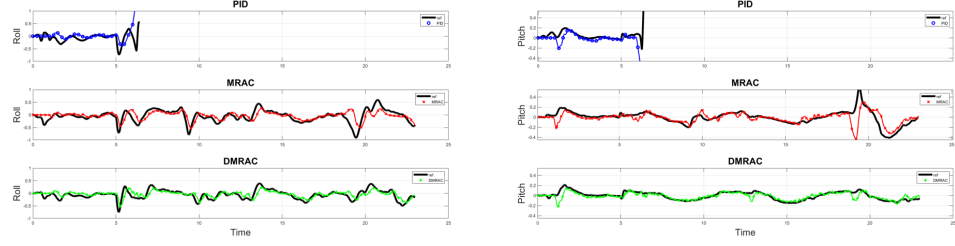

Figure 7: Tracking performance of quadrotor in Roll-Pitch, for a circular trajectory under wind bias with nonlinear and unpredictable disturbance.
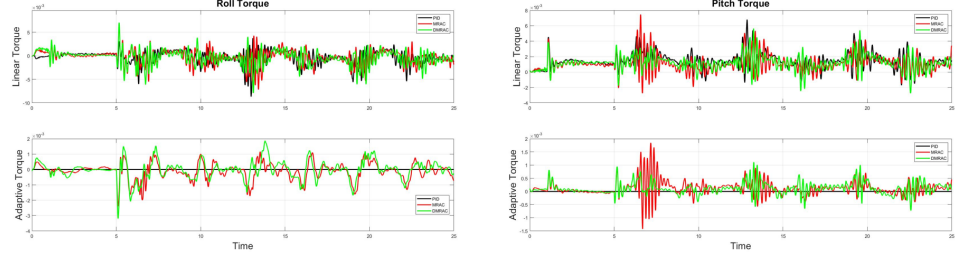

Figure 8: Feedback,Feed-forward and Adaptive control torque for PID, MRAC and DMRAC for Circular trajectory tracking under high wind bias

A.3.2 Evaluating Transfer Learning with DMRAC

Lately, Transfer learning (TL) has been a much-researched topic in machine learning and reinforcement learning. In similar lines in these experiments, we aim to test the advantages of representation transfer in an adaptive control setting. We test transfer learning through sharing the network parameters between tasks. TL is performed by first running DMRAC on related tasks and learning the network weights, which incorporate some feature knowledge. These learned weights are then used to initialize a fresh DMRAC network executing a new unseen task. We use a flight test of a drone executing a basic figure of 8 trajectories as a source task for representation transfer through the warm-start of the networks. The target task is an unseen but related task for which an initialized network is used for the drone executing figure of 8 trajectories under high wind bias, refer Fig-9. A clear improvement of controller performance in achieving smaller transients and better steady-state tracking is observed with warm-started DMRAC. The deep network weights learnt over the quadrotor executing figure of 8 trajectories encodes the feature knowledge about modeling uncertainties. When we transfer this learning to a new drone executing figure of 8 with wind bias, it is able to adapt faster and also quickly learn features corresponding to wind bias.

A.3.3 Simulation to Real-World Transfer Learning

The following experiments are similar to one in the previous section. Here we are investigating the network representation transfer from simulation to the real world. In this experiment, DMRAC is run in a simulation environment, where the network is trained over data collected through the
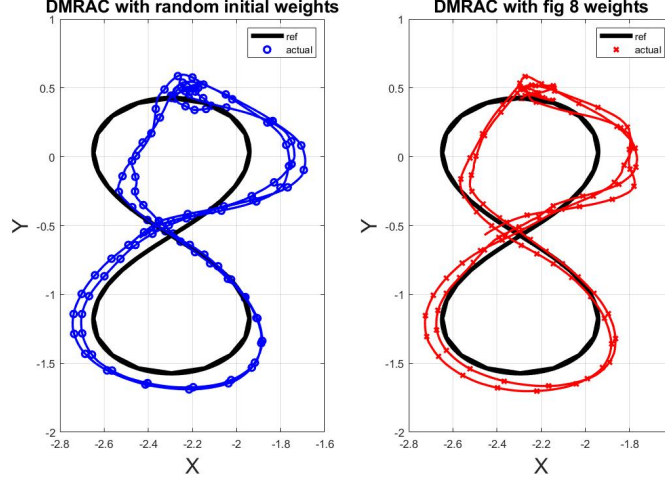
14

Figure 9: Figure of 8 Trajectory tracking under wind bias with random initialization vs. Feature transfer in DMRAC

simulated drone follow a figure of 8 trajectories without any disturbance. These trained network weights are then used as initialization weights for the case where DMRAC is experimented on the actual physical quadrotor. The controller performance is compared between the randomly initialized DMRAC vs. DMRAC initialized with network weights from the simulated quadrotor. We test the two controllers performing a figure of 8 trajectory maneuvers under high wind bias. Figure-10 shows the improvement in DMRAC's performance when the initial weights are from simulation rather than being initialized entirely randomly.
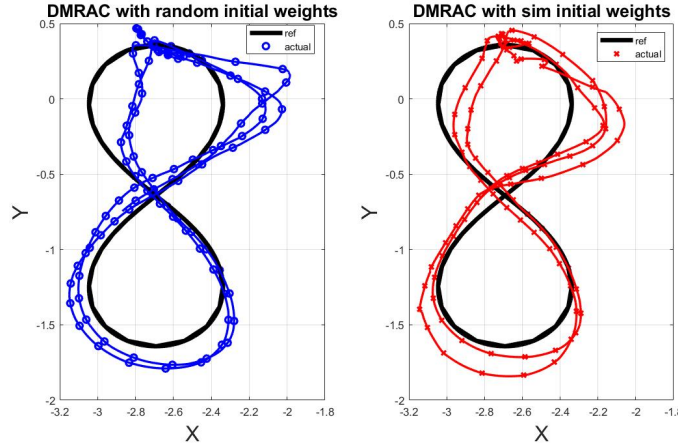

Figure 10: Figure of 8 Trajectory tracking under wind bias with random initialization vs. Feature transfer from simulation to Real in DMRAC

A.3.4 Learning Retention

Figure-11,12 and Fig-13 show the closed loop system performance in tracking the reference signal for DMRAC controller and learning retention when used as the feed-forward network on a similar trajectory (Circular) with no learning. We demonstrate the proposed DMRAC controller under uncertainty and without domain information is successful in producing desired reference tracking. Since DMRAC, unlike traditional MRAC, uses DNN for uncertainty estimation is hence capable of retaining the past learning and thereby can be used in tasks with similar features without active online adaptation Figure-12-13. Whereas traditional MRAC which is "pointwise in time" learning

15

algorithm and cannot generalize across tasks. The presented controller achieves tighter tracking with smaller tracking error in both outer and inner loop states as shown in Fig-12 and Fig-13 in both with adaptation and as a feed-forward adaptive network without adaptation. Figure-14 demonstrate the DNN learning performance vs epochs. The Training, Testing and Validation error over the data buffer for DNN, demonstrate the network performance in learning a model of the system uncertainties and its generalization capabilities over unseen test data.
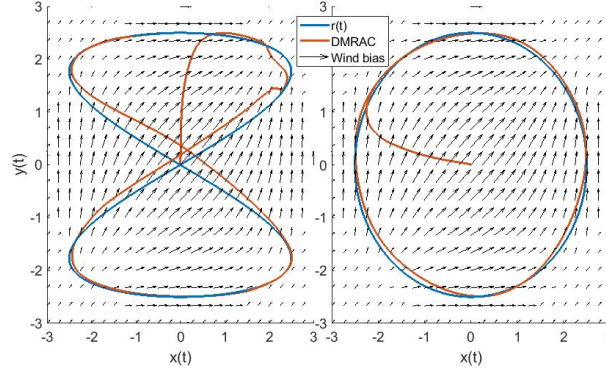
Figure 11: DMRAC Controller Evaluation on 6DOF Quadrotor dynamics model: DMRAC Controllers on quadrotor trajectory tracking with active learning for figure-8 tracking and DMRAC as frozen feed-forward network (Circular Trajectory) to test network generalization
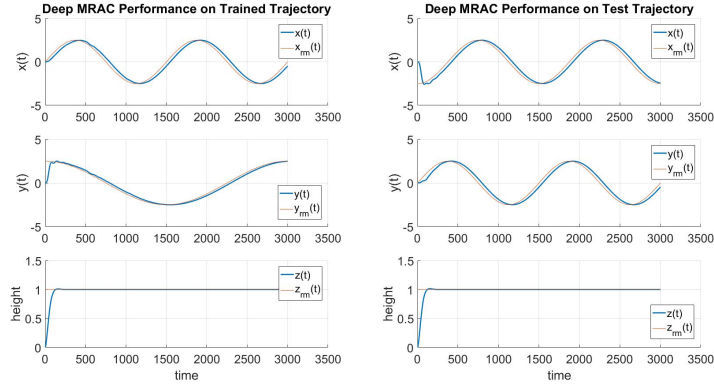

Figure 12: DMRAC Controller Evaluation on 6DOF Quadrotor dynamics model: Closed-loop system tracking performance in position x, y and height z
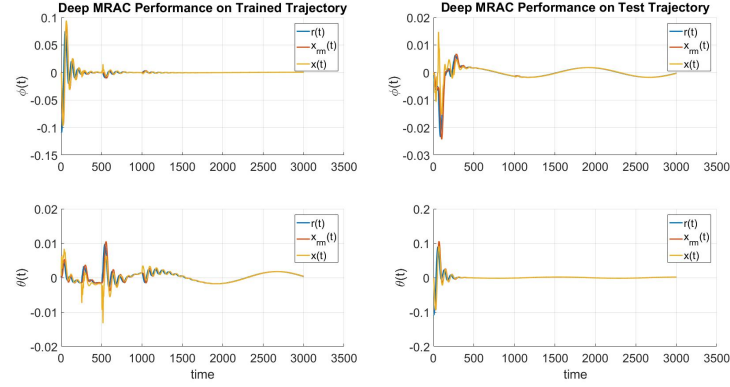
16

Figure 13: DMRAC Controller Evaluation on 6DOF Quadrotor dynamics model: Closed-loop system tracking performance in roll rate $\phi(t)$ and Pitch $\theta(t)$
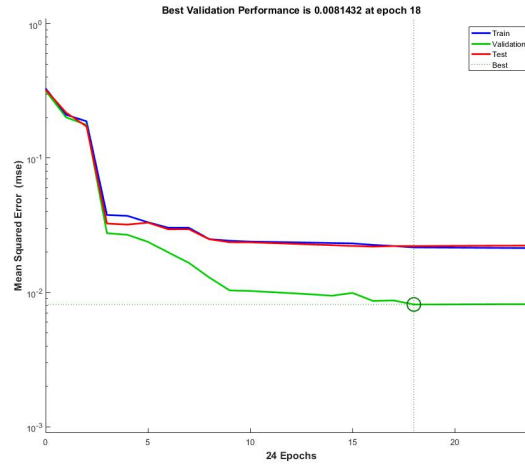

Figure 14: DMRAC Controller Evaluation on 6DOF Quadrotor dynamics model: Deep neural network training performance over train, test and validation sets.