Reference Governors Based On Offline Training Of Regression Neural Networks

Chuan Yuan Lim[1], Hamid Ossareh[2], Ilya Kolmanovsky[1]

***Abstract*—This paper presents two machine learning-based constraint management approaches based on Reference Governors (RGs). The first approach, termed NN-DTC, uses regression neural networks to approximate the distance to constraints. The second, termed NN-NL-RG, uses regression neural networks to approximate the input-output map of a nonlinear RG. Both approaches are shown to enforce constraints for a nonlinear second order system. NN-NL-RG requires a smaller dataset size as compared to NN-DTC for well-trained neural networks. For systems with multiple constraints, NN-NL-RG is also more computationally efficient than NN-DTC. Finally, promising results are reported by having both approaches implemented on a more complex spacecraft proximity maneuvering and docking application, through simulations.**

***Index Terms*—Reference Governors, Offline Training, Neural Networks, Constraint Management**

I. Introduction

Reference governors (Fig. 1) are add-on control schemes that are augmented to nominal, well-designed controllers, to enforce state and control constraints [1]–[3]. Based on the current state measurement or estimate x_k, the reference governor modifies the desired reference command r_k, whenever is required, to prevent constraint violation. The modified reference command, v_k, is then fed into the nominal controller to generate the control input u_k, which then drives the plant, producing the output y_k. As such, the reference governor essentially acts as a pre-filter, preventing reference commands that may lead to constraint violations. It is worth noting that the term "reference governor" refers to a set of techniques with a common intent of preserving, whenever possible, the closed-loop system response designed with conventional control techniques. A detailed survey on techniques based on reference governors (RGs) can be found in [1], [2].

RGs generally require explicit knowledge of the model of the system and constraints. On the other hand, learning reference governors (LRGs) operate through learning from data, such as a black-box model of the system, or actual system through experimentation, and therefore do not require explicit knowledge of the model. Liu et al. in [4] demonstrated the use of LRG in guaranteeing constraint satisfaction in an automated rendezvous, proximity maneuvering operation, and docking (ARPOD) of a spacecraft. Furthermore, with limited knowledge of the model, Liu et al. have also shown that constraint enforcement is theoretically guaranteed from learning, with the proposed learning algorithm guaranteeing convergence and applied to a fuel truck rollover avoidance in [5]. Li et al. used deep reinforcement learning to learn system constraints, with successful enforcement of actuation and state constraints of an active suspension system in [6]. Reference [7] explores support vector machines to determine stability regions for constrained nonlinear systems. Lanchares et al. in [8] have applied LRG, based on a neural network, to approximate the maximal output admissible set. It was shown that LRG can successfully enforce constraints on several examples. In all of the above references, the LRGs are trained online, resulting in the need for a learning phase, which could be at times a lengthy process, at the start of the system operation. Furthermore, details on determining the neural network topology for LRGs are often lacking. Examples from [8], [9] use neural networks with one hidden layer, [10] with 4 hidden layers, and a specified number of neurons for their results analyses. The process of deriving the number of neurons in particular is insufficiently described.

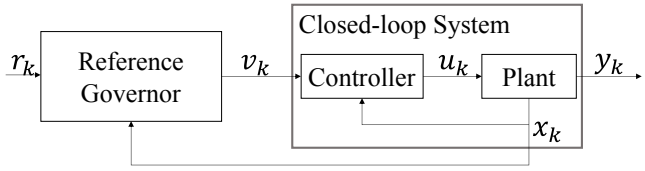

Fig. 1. Reference governor controller scheme. r_k is the reference, v_k is the adjusted reference, u_k is the nominal controller input, x_k is the system state, y_k is the constrained output.

This paper presents two approaches to designing LRGs for systems with soft constraints, with a focus on training regression neural networks offline and on methods for dataset generation. The first approach, termed NN-DTC, is to train a regression neural network to approximate the distance to constraint function. As we will discuss, this may be preferable to a classification neural network in [8] that separates safe and unsafe reference command and state pairs, but requires that the distance to constraints be measured in the collected dataset. It is found that large dataset sizes are needed to build well-trained neural networks for the NN-DTC approach. Our second approach, termed NN-NL-RG, involves the direct training of neural networks to approximate the input-output map of the nonlinear reference governor in [1]. NN-NL-RG predicts the modified reference command directly. It

[1]C. Y. Lim and I. Kolmanovsky are with the Department of Aerospace Engineering, University of Michigan in Ann Arbor, MI, USA (email: cyuanlim, ilya@umich.edu).

[2]H. Ossareh is with the Department of Electrical and Biomedical Engineering, University of Vermont in Burlington, VT, USA (email: hamid.ossareh@uvm.edu).

This work was partially supported by NASA and NIST under cooperative agreements VT-80NSSC20M0213 and 70NANB21H162 as well as by the National Science Foundation Award ECCS-1931738.

is advantageous for systems with multiple constraints, and requires a smaller dataset size to build well-trained neural networks, implying faster, offline, training times. This paper also presents a systematic approach to deriving suitable neural network topologies, which has not been considered in [8]. The developments exploit two case studies, a second order nonlinear system with finite escape time, and a spacecraft ARPOD case study as in [4]. In these case studies, both approaches are shown to lead to solutions which strictly enforce the given constraints.

In summary, the original contributions of this paper are:

- The NN-DTC approach is introduced and is shown to lead to improvements over the previous classification neural networks used in LRGs of [8].
- The NN-NL-RG approach is proposed with the potential to shorten offline training times and lower online computational footprint over RG implementation that uses NN-DTC.
- An approach to deriving a suitable neural network topology for LRGs is proposed.
- Methods to generate the dataset for offline training, and the necessary sizes of the dataset in order to build a well-trained neural network, are described and analysed.

This paper is organized as follows. Section 2 presents a review of the maximal output admissible set, reference governors, and the nonlinear reference governor. Section 3 applies an existing LRG approach of [8] to a second order nonlinear system and shows its limitations. Section 4 uses the same second order nonlinear system to show the development of NN-DTC and NN-NL-RG. Section 5 presents computational results for the spacecraft ARPOD system based on NN-DTC and NN-NL-RG. Concluding remarks and topics for future research are discussed in Section 6.

II. Preliminaries

A. Maximal Output Admissible Set and Reference Governors

Consider a discrete-time, asymptotically stable closed-loop system described by:

$$x_{k+1} = f(x_k, v_k),\ y_k = g(x_k, v_k), \tag{1}$$

where $x_k \in \mathbb{R}^n$ is the state vector, $v_k \in \mathbb{R}^m$ is the adjusted reference governor, and $y_k \in \mathbb{R}^p$ is the constrained output vector. The subscript k refers to the k^{th} discrete time instant of time $t = kT_s$, where T_s is the sampling period. The constraints are imposed as $S(y_k) \leq 0$, for all $k \geq 0$, where $S\colon \mathbb{R}^p \to \mathbb{R}^q$, can be nonlinear, and vector inequalities are to be interpreted element-wise. The set $Y \subset \mathbb{R}^p$ is then defined as one where all q constraints are satisfied,

$$Y = \{y \in \mathbb{R}^p \text{ s.t. } S(y) \leq 0\}. \tag{2}$$

The Maximal Output Admissible Set (MAS), denoted by O_∞, is the safe set of initial states x_0 and constant input commands $v_k = v_0$ that satisfy constraints for all future time steps [11]

$$O_\infty = \{(x_0, v_0) : v_k = v_0,\ k \in \mathbb{Z}_+ \Rightarrow y_k \in Y,\ k \in \mathbb{Z}_+\}. \tag{3}$$

At each time instant kT_s, a version of the RG called the command governor, would select an adjusted v_k by solving the following minimization problem:

$$v_k \in \arg\min_v\ (v - r_k)^T P (v - r_k) \text{ s.t. } (x_k, v) \in O_\infty, \tag{4}$$

where P is a positive definite matrix. From (3), and assuming no model mismatch, the solution to (4) is constraint-admissible; furthermore, the existence of a feasible solution to (4) at the initial time implies, assuming no model mismatch, the existence of a feasible solution for all future times, a property referred to as recursive feasibility.

B. Nonlinear Reference Governor (NL-RG)

An alternative approach to calculating v_k is the Scalar Reference Governor (SRG) [1], [2], in which

$$v_k = v_{k-1} + \kappa(r_k - v_{k-1}), \tag{5}$$

where κ is obtained by solving the following linear program:

$$\kappa = \arg\max_{\kappa \in [0,1]} v_k,\ \text{s.t. } (x_k, v_k) \in O_\infty, \tag{6}$$

and where v_k is defined in (5). The approach originally proposed in [12] exploits an implicit characterization of O_∞ through the use of online simulations to predict the response over a sufficiently long horizon. With this approach, the functional representation of O_∞ is never constructed and (6) is implemented through the bisections. In this paper, we refer to this method as the nonlinear reference governor (NL-RG), while noting that there are other constructions of RGs for nonlinear systems (including those that use subsets of O_∞) which we do not consider.

III. Second Order Nonlinear System and An Existing LRG Approach of [8]

In this section, we review the algorithms proposed in [8], which uses a neural network to generate an approximation of the MAS, and show that training a classification neural network offline does not guarantee a satisfactory approximation of the MAS for the purpose of real-time constraint management. Subsequently, this motivates the development of the NN-DTC. Consider a non-linear second order system given by the following continuous-time model:

$$\dot{x}_1 = x_1^2 + u,\ \dot{x}_2 = x_1 - v, \tag{7}$$

$$u = \mathbf{sat}_{[-20,20]}(-v^2 - k_p(x_1 - v) - k_I x_2), \tag{8}$$

$$k_p = 5.6,\ k_I = 16. \tag{9}$$

The system represents a first order nonlinear system (7) controlled by a Proportional-plus-Integral controller and a feedforward term. The system is subject to control input saturation in (8). The output of the system is $y = x_1$ and

is subject to the constraint $y(t) \leq 3.5$. Denoting this system as Case Study 1, an aggressive reference profile is designed in order to test the ability of NL-RG to enforce constraints. Fig. 2 shows the simulation result with such a profile, and the adjusted v values to ensure constraints are not violated.

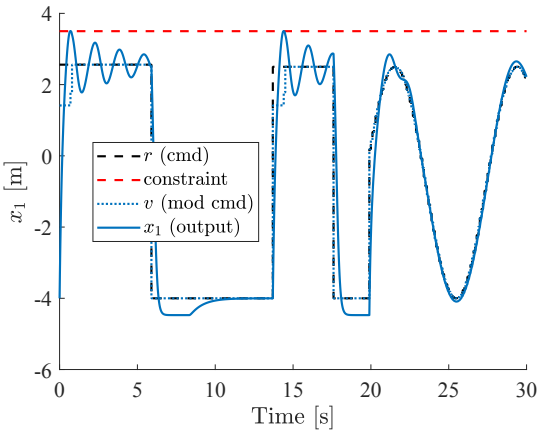

Fig. 2. Simulation result using NL-RG.

We first consider the approach of [8] which uses a classification neural network. For the purpose of neural network training, we chose 20,000 data points where each data point is a sampled pair of (x, v) from a uniform distribution in a region described by: $x_1 \in [-20, 20]$, $x_2 \in [-1, 1]$, $v \in [-4, 4]$. Then (x_1, x_2) is applied as an initial condition, and with v held constant, a simulation over a time interval of length $t_h = 30$ sec is carried out. The value of t_h was chosen to be sufficiently large so that either transients subdue or the trajectories escape prior to that time. The trajectory of the system is then checked for constraint violation. Should there be no constraint violation along the simulated trajectory, the pair is labeled as safe, and it is labeled unsafe otherwise. Fig. 3 shows each (x, v) pair labeled as safe or unsafe, after completing simulations for all (x, v) pairs. For notation purpose, define a function $L(x, v)\colon \mathbb{R}^n \times \mathbb{R}^m \to \{0, 1\}$ such that

$$L(x, v) = \begin{cases} 1, & \text{if } (x, v) \text{ is labeled safe,} \\ 0, & \text{if } (x, v) \text{ is labeled unsafe.} \end{cases} \tag{10}$$

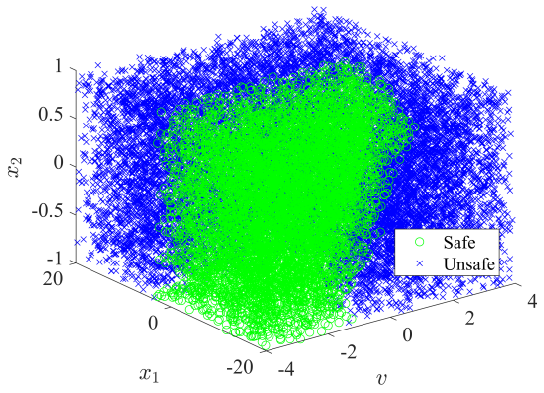

Fig. 3. Pairs of (x, v) labeled as safe or unsafe.

For neural network implementation, the command `patternnet` of MATLAB (ver. R2021b) is used. The training options applied are set as default, whereby the activation function is the tangent-sigmoid function, the training algorithm is the scaled conjugate gradient, and the input data are normalized. The dataset is randomly divided into 3 sets with 70% of data used for training, 15% for validation and 15% for testing. Stopping options were set to recommended default values to prevent overtraining. The input features are the (x, v) pairs, and the target feature is the safe/unsafe label obtained from (10).

For this case study, a neural network with only a single hidden layer is used. This is based on the expectation that a single layer is sufficient given that the system is low order and O_∞ is in a low dimensional space. The hyperparameter of this neural network is the number of neurons in the hidden layer; its values are set to 10, 25, 50, and 100. For each design, 10 trials are made with randomized initialization of biases, weights, and data split, resulting in 40 different neural networks. Let a neural network with α hidden layers and β neurons be denoted as $\text{NN}_{\alpha/\beta}$ and let $\text{NN}_{\alpha/\beta/\gamma}$ refer to the neural network resulting from the trial number γ. After training, the neural network defines a function $\zeta(x, v)\colon \mathbb{R}^n \times \mathbb{R}^m \to [0, 1]$, $\zeta(x, v) = p$. The value p is a confidence level output from the neural network after classifying a point. If p is close to 1, the input pair is likely safe; if it is close to 0, the pair is unsafe. To assess the neural network ability in classifying safe/unsafe pairs and avoid biasing it to be conservative, we assume a fixed threshold value $p_{th} = 0.5$, i.e., we do not tune it for the specific application as in [8].

For evaluation, each neural network is used to inform O_∞ based on the condition $\zeta(x, v) \geq p_{th}$ while RG (3)-(4) is implemented by searching for $v_k \in [-4, 4]$ over a discrete set of points such that $\zeta(x_k, v_k) \geq p_{th}$. The value of v_k is set to v_{k-1} if no feasible solution is found in this search, which is expected to be feasible due to the construction of the RG. Simulations are performed using the same profile of r as in Fig. 2. The initial condition for x_2 is set to 0 while discrete values of initial x_1 from the interval of $(-20, 3)$ in steps of 1 were chosen, yielding 24 cases. To quantify the performance of the neural networks trained, the mean value of constraint violation in x_1, denoted by μ_{x_1}, is computed as

$$\mu_{x_1} = \frac{1}{n} \sum_{k=1}^{n} \max\{0, x_{1k} - \bar{x}_1\}, \tag{11}$$

where $\bar{x}_1$ is the constraint value, which is 3.5 in this case study; n is the total number of discrete timesteps of length $T_s = 0.01$ sec within the simulation time, $t_{\texttt{.sim}} = 30$ sec. A neural network is then determined to be well-trained based on its ability to enforce constraints for all 24 cases based on the value of μ_{x_1}, e.g., $\mu_{x_1} \leq 10^{-5}$.

As Fig. 4 shows, none of the neural networks are able to produce a RG that enforces constraint for all 24 cases. Increasing the number of neurons improves the neural network performance, but at 50 or 100 neurons, the number of neurons

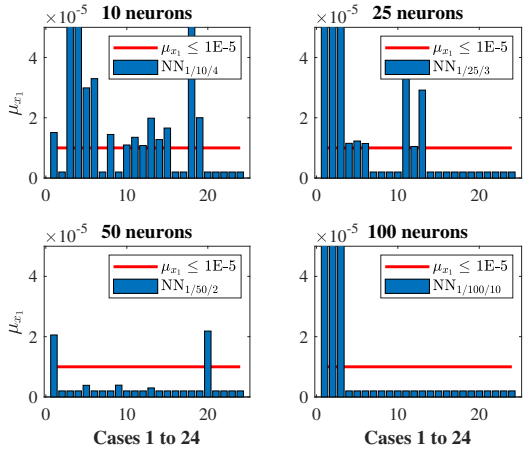

Fig. 4. Classification neural network performance evaluation, showing the best trial result out of 10 trials for the neural networks designed with 1 hidden layer and varying number of neurons.

is already greater than that used by [8] (20 and 40 neurons) and [9] (9 neurons). In conclusion, we were not able to find a classification neural network, trained offline, for the system (7)-(9) following the approach of [8].

IV. Main Results

In this section, we formally introduce the two proposed approaches, namely NN-DTC and NN-NL-RG.

A. Distance-to-constraint Approximation (NN-DTC)

Instead of exploiting safe/unsafe pairs as in [8], we define a function that characterizes the maximum distance to constraint violation. Specifically, we define the function $D(x, v) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$:

$$D(x, v) = \max_k \{y_k - \bar{y}\}, \tag{12}$$

where y_k is the simulated output corresponding to the initial condition x and constant input v, and $\bar{y}$ is the constraint. Note that (12) implies

$$D(x, v) = \begin{cases} \leq 0, & \text{if } (x, v) \text{ is labeled safe,} \\ > 0, & \text{if } (x, v) \text{ is labeled unsafe.} \end{cases} \tag{13}$$

A regression neural network is then trained to approximate $D(x, v)$, i.e., to predict the distance to constraint for given x and v. For the regression neural network implementation, we used the command `fitnet` of MATLAB. The Levenberg-Marquardt training algorithm was applied as the optimization function, while the rest of training options and data splitting follows those described in Section III. The input features are the (x, v) pairs, and the target feature is the distance to constraint values obtained from (12). After training, the neural network defines a function $\psi(x, v)$: $\mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, $\psi(x, v) = d$, where d is the predicted distance to constraint. If $d \leq d_{th}$, the input pair is likely safe; if $d > d_{th}$, the pair is unsafe. To assess the neural network's ability in classifying safe/unsafe pairs and avoid biasing it to be conservative, we assume a fixed threshold value $d_{th} = 0.0$.

Neural network performance evaluation is carried out using the same procedure as described in Section III. One difference is that each neural network is used to inform O_∞ based on the condition $\psi(x, v) \leq d_{th}$, while RG (3)-(4) is implemented by searching for $v_k \in [-4, 4]$ over a discrete set of points such that $\psi(x_k, v_k) \leq d_{th}$. Fig. 5 shows that neural networks with either 10 or 50 neurons could produce RGs that enforce constraints for all 24 cases, while a neural network with 100 neurons is unable to do so. The poorer performance for a greater number of neurons is likely due to overfitting which degrades its generalization abilities [13]. As such, this result is also in line with [8] and [9], where a smaller number of neurons is sufficient for a well-trained neural network.

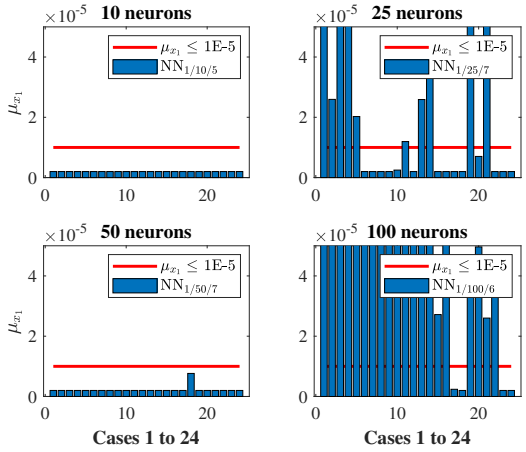

Fig. 5. NN-DTC performance evaluation, showing the best trial result out of 10 trials for the neural networks designed with 1 hidden layer and varying number of neurons.

To further robustify the performance of the neural network predictions, an ensemble of 5 out of the 40 best performing neural networks is assembled and $\phi(x, v)$ is defined so that it is set to the minimum output of these 5 neural networks. With this implementation, no constraint violation occurred in any of the simulations (Fig. 6).

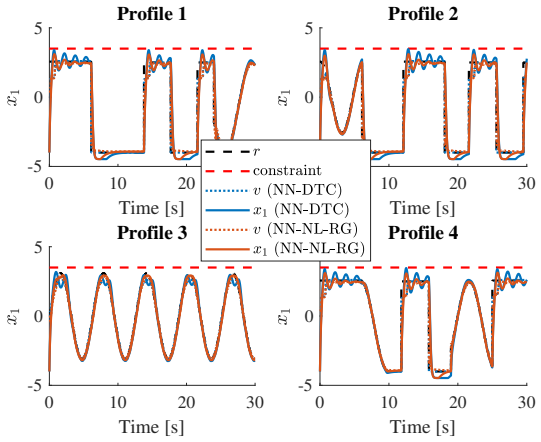

Fig. 6. Simulation results using NN-DTC and NN-NL-RG on various reference profiles. Profile 1 is the same as that in Fig. 2.

B. NL-RG Approximation (NN-NL-RG)

This approach replaces the NL-RG described in [12] with a neural network as an approximation function. Fig. 7 shows the NL-RG which uses r_k, x_k, and v_{k-1} as inputs to generate v_k, and the regression neural network replacing it.

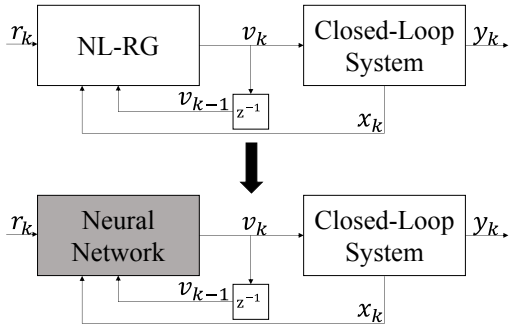

Fig. 7. NL-RG augmenting the nominal closed-loop system, replaced by a regression neural network.

For dataset generation, we note that a control problem could be posed as either a regulation or non-zero reference tracking. Each requires a different procedure for dataset generation.

Regulation: For a regulation problem, r_k is 0 for all times. As such, it is an inconsequential input to NL-RG. To build the dataset for such a system, we first sample x over a uniform distribution across the respective bounds, forming a total of N samples. Each sample is then used to initialize the system, and a simulation over a defined time interval t_h is carried out to obtain v_k, as computed by the NL-RG algorithm. NL-RG ensures that the computed v_k is constraint-admissible for the entire trajectory. The second input feature of the neural network, namely v_{k-1}, is then obtained by applying a unit delay to v_k. For neural network training, the input features are the (x_k, v_{k-1}) pairs and the target feature is v_k.

Reference Tracking: For a control system with a reference tracking objective, a random sequence of steps is generated to represent the command profile r. Simulations with this r profile are then carried out using NL-RG to obtain v that would not violate constraints. As in the regulation problem, v_{k-1} is obtained by applying a unit delay to v. For neural network training, the input features are (x_k, v_{k-1}, r_k) and the target feature is v_k.

For both procedures, the entire simulated trajectory is stored, and is used as a part of the dataset for neural network training. As such, the dataset would either contain datapoints from one single, extended trajectory, or it could be stitched from multiple, completed, trajectories.

The trained neural network is then used in place of the NL-RG in the numerical simulations. At each discrete time instant k, given the state x_k, the previous command v_{k-1}, and, in the case of a tracking problem, the command r_k, the trained neural network is applied to directly predict v_k. This eliminates the need to solve a minimization problem required in NN-DTC. Furthermore, a single neural network is also sufficient for a system with multiple constraints. This greatly improves computational efficiency. Therefore, after training, the neural network defines a function $\xi(x, v_{k-1}, r) : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$, where $\xi(x, v_{k-1}, r) = v_k$.

We now apply NN-NL-RG to Case Study 1. Multiple simulations of 30s were carried out with different random reference profiles. Fig. 6 shows examples of the reference profiles used, together with different initializations of (x_1, x_2). The trajectories are then collated into a dataset that is used for training. There are 2,000 data points in total for this dataset. Note that this is 10 times smaller than the dataset used for training in NN-DTC.

Neural network performance evaluation is carried out using the same procedure as described in Section III. Fig. 8 shows $\text{NN}_{1/10}$ inability to satisfy constraints for a number of cases when trained with a smaller dataset. A minimum of 25 neurons is needed, and similar to NN-DTC, 100 neurons could not enforce constraints for all 24 cases due to loss of generalization abilities. Assembling an ensemble of 5 out of the 40 best performing neural networks, each neural network predicts a v_k for a given input (x_k, v_{k-1}, r_k). The minimum of the predicted v_k values is then applied to the nominal controller. With this implementation, no constraint violation occurred in any simulation (Fig. 6). As such, a significant benefit of using NN-NL-RG is perhaps its ability to generate well-trained neural networks with smaller datasets, which also implies time-savings when training the neural networks. This is further verified by training NN-DTC with a dataset of only 2,000 points, which resulted in no neural network being able to enforce constraints for all 24 cases.

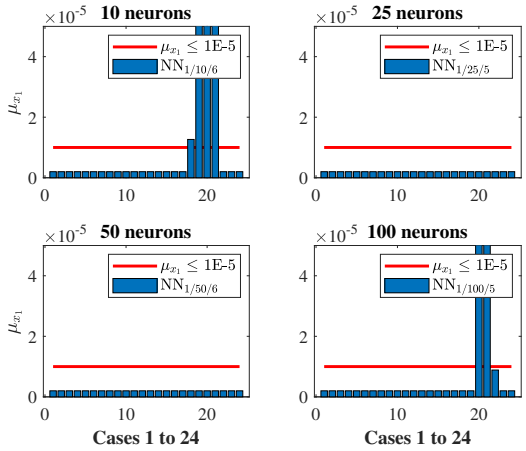

Fig. 8. NN-NL-RG performance evaluation, showing the best trial result out of 10 trials for the neural networks designed with 1 hidden layer and varying number of neurons.

V. Spacecraft ARPOD Case Study

The two approaches presented in the previous section are applied to an ARPOD operation of a chaser spacecraft to a target spacecraft [4], [14]. The objective is to enforce both thrust and line of sight cone constraints, similar to the 3rd case mentioned in [4]. The closed-loop spacecraft relative motion dynamics are described as

$$\begin{aligned} \dot{x}_k &= f(x_k, v_k), \\ \Upsilon_k &= g(x_k, v_k), \end{aligned} \tag{14}$$

where $x_k = [\delta x, \delta y, \delta \dot{x}, \delta \dot{y}, \theta, \dot{\theta}]^T$ denotes the state of the spacecraft at the time instant k, $\delta x, \delta y, \delta \dot{x}, \delta \dot{y}$ are the components of the position and velocity vectors of the spacecraft in Hill's frame, $\theta, \dot{\theta}$ are angle and angular velocity that prescribed the direction of the thrust vector; v_k is the vector of desired δx and δy of the spacecraft coordinates. The LRG is applied to this closed-loop system that consists of a plant being controlled by a nominal controller. A key point to note is that this is a regulation case study; the objective is to have the position of the chaser spacecraft go to $\delta x = \delta y = 0$. Υ_k is the output of the system at time instant k on which the constraints are imposed. The constraints of the system are: a maximum thrust limit where the thrust force, $F_T \leq \bar{F}_T = 1.5$N, and Line of Sight (LOS) cone angle constraint, where the cone angle, $\phi \leq \bar{\phi} = 2.5^\circ = 0.0436$rad. Cone constraint is further split into 2 separate constraints, such that $\phi \leq \bar{\phi} = 0.0436$rad and $\phi \geq -\bar{\phi} = -0.0436$rad. They are denoted as *upcone*, $\bar{\phi}_{up}$, and *dncone*, $\bar{\phi}_{dn}$, respectively. For the sake of brevity, the reader is referred to [4] for the detailed derivation of the constraints.

A nominal controller has already been designed, where a thruster controller is used to provide the desired thrust force, and a Proportional-Derivative (PD) controller is used to track the desired orientation of the spacecraft. As with the implementation in [4], a LRG is then designed to enforce both constraints together with the nominal controllers. Cone angle and thrust constraints would be violated without RGs implemented [4].

As this is a more complex, higher order nonlinear system with multiple constraints, both the number of hidden layers and number of neurons in each layer are set as hyperparameters of the neural network. The number of hidden layers are either 1 or 2, and the number of neurons per layer are set to 10, 25, 50 and 100. As such, there are 8 combinations of hyperparemeters. Similar to Case Study 1, 10 trials are made for each combination.

For evaluation, each neural network is applied into a simulation with 1 of the 9 initial conditions listed in Table I, and all other states $\delta \dot{x} = \delta \dot{y} = \theta = \dot{\theta} = 0$. This yields 9 cases. To quantify the performance of the neural networks after training, the mean values of constraint violation in F_T, ϕ_{up}, and ϕ_{dn}, denoted by μ_{F_T}, $\mu_{\phi_{up}}$ and $\mu_{\phi_{dn}}$, are computed as

$$\begin{aligned} \mu_{F_T} &= \frac{1}{n} \sum_{k=1}^{n} \max\{0, F_{T_k} - \bar{F}_T\}, \\ \mu_{\phi_{up}} &= \frac{1}{n} \sum_{k=1}^{n} \max\{0, \phi_{up} - \bar{\phi}_{up}\}, \\ \mu_{\phi_{dn}} &= \frac{1}{n} \sum_{k=1}^{n} \max\{0, \phi_{dn} - \bar{\phi}_{dn}\}. \end{aligned} \tag{15}$$

A well-trained neural network is then determined based on its ability to enforce the constraints for all 9 cases over a fixed threshold. Similar to Case Study 1, the threshold is set to 10^{-5}.

TABLE I
PAIRS OF δx AND δy USED FOR INITIALIZATION

Case	$[\delta x, \delta y]$	Case	$[\delta x, \delta y]$	Case	$[\delta x, \delta y]$
1	$[-18, -300]$	**2**	$[0, -300]$	**3**	$[18, -300]$
4	$[-12, -200]$	**5**	$[0, -200]$	**6**	$[12, -200]$
7	$[-9, -100]$	**8**	$[0, -100]$	**9**	$[9, -100]$

A. Approach 1: NN-DTC

Similar to Case Study 1, the dataset is generated through sampling of (x, v) pairs from a uniform distribution in a region described as: $\delta x \in [-20, 20]$, $\delta y \in [-350, 10]$, $\delta \dot{x} \in -[0.5, 0.2]$, $\delta \dot{y} \in [-1.2, 1.2]$, $\theta \in [-5, 4]$, $\dot{\theta} \in [-0.7, 0.2]$, $v \in [-350, 10]$. For this ARPOD case study, $t_h = 400s$. Datasets containing 15,000 and 20,000 data points were found to not produce viable neural networks that could enforce all 3 constraints, thus a dataset containing 25,000 data points was used. As this system has 3 constraints, three separate neural networks were trained for predicting distance to each constraint. Training a neural network for a single output would ensure better fitting, and makes evaluation of the neural network performance simpler. Training a neural network for multiple constraints shall be left as future work.

Neural networks trained to manage *upcone* and *dncone* constraints exhibit minimal violations and are hence not discussed here. Fig. 9 shows that thrust constraint is more difficult to enforce, with more cases incurring constraint violations. $\text{NN}_{1/100}$ has the best performance by being able to enforce thrust constraints for all 9 cases. Furthermore, using 2 hidden layers did not yield better performance, likely due to over-parameterization.

Similar steps are taken to robustify the performance of the neural network predictions, with an ensemble of 5 neural networks made for each constraint. With this implementation, the regression neural networks successfully enforced both thrust and cone constraints, and have the chaser spacecraft successfully reaching the target spacecraft for all 9 cases. Subjecting the same ensemble of neural networks to a Monte Carlo simulation with random initialization of the states, Fig. 10 show that constraints are satisfied for these cases as well. The time taken for all these cases is well within 1000 sec. This is far shorter than that from [4], as the need for a lengthy online learning phase has been shifted offline in our approach.

B. Approach 2: NN-NL-RG

As this is a regulation case study, we initially used the method as described in Section IV-B to generate the dataset. The same bounds, as described previously, are used for the random sampling of x. We note that such a dataset used for training yield poorly trained neural networks. This is due to a large portion of the v_{k-1} and v_k values being zeros, as r_k is always zero for all times. As such, a modification is made to address this issue. For the ARPOD case study, v_k is a desired value of δy_k. To enhance the dataset quality, we change the input feature v_{k-1} to $v_{k-1} - \delta y_{k-1}$, i.e., the tracking error along the δy coordinate, and the target feature

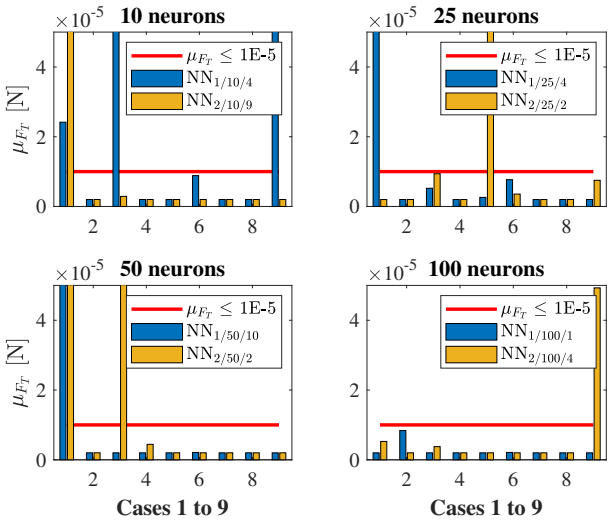

Fig. 9. NN-DTC performance evaluation for thrust constraint, showing the best trial result out of 10 trials for the neural networks designed with either 1 or 2 hidden layers and varying number of neurons.

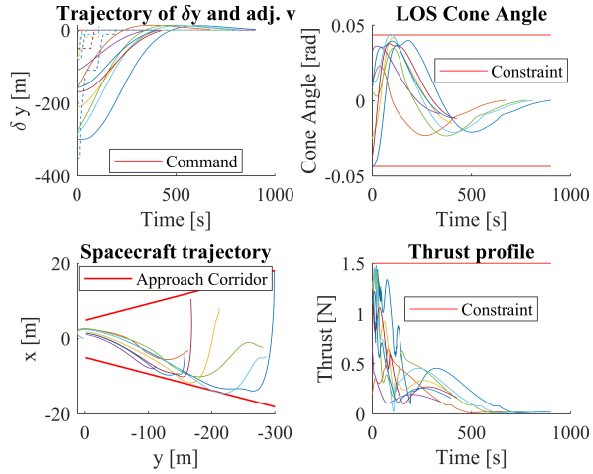

Fig. 10. NN-DTC Monte Carlo simulations.

v_k to $v_k - \delta y_k$. This mitigates the issue of large portions of zeros for both v_{k-1} and v_k values, and significantly improved neural networks training performance. Therefore, to reiterate, the input features are now $(x_k, v_{k-1} - \delta y_{k-1})$ pairs, and the target feature is $v_k - \delta y_k$. Unlike NN-DTC, using NN-NL-RG requires only one neural network to enforce all 3 constraints in the ARPOD case study.

Neural networks are trained with different dataset sizes, where $N = 5000, 10000$ or 20000. For sake of brevity, we based the analysis on just $\text{NN}_{1/10}$ and $\text{NN}_{2/10}$, and illustrate the impact of different N on thrust constraint. Fig. 11 shows that for $N = 5000$ or 10000, constraint violations occur for majority of the 9 cases; having 2 hidden layers did not improve neural network's performance either. At $N = 20000$, the neural networks trained are able to satisfy thrust constraint requirement for all 9 cases. As such, $N = 20000$ dataset is used for subsequent analysis. Furthermore, as the ARPOD case study is a higher order nonlinear system, it is expected that a larger dataset is required to produce well-trained neural networks. However, the dataset size required for NN-NL-RG

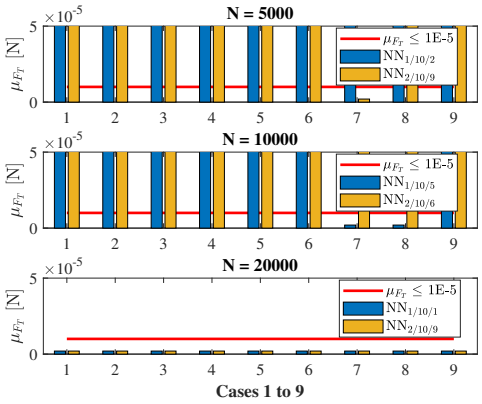

Fig. 11. Effect of dataset size, N, on NN-NL-RG performance

is still smaller than that for NN-DTC.

The performance of the NN-NL-RG is evaluated in Fig. 12 for the thrust constraint. The trend is similar for upcone and dncone constraints and are hence omitted for conciseness. It is remarkable that neural networks with 10 or 25 neurons per layer could enforce all 3 constraints. Furthermore, although one hidden layer shows constraint violations with 50 and 100 neurons per layer, this is resolved with 2 hidden layers. As such, these are all networks that can be used to build the ensemble for robustifying the neural network performance. Selecting 5 best performing networks, the regression neural networks could also satisfy both thrust and cone constraints, and have the chaser spacecraft successfully reaching the target spacecraft for all 9 cases. Subjecting the same ensemble of neural networks to the same Monte Carlo simulations applied with NN-DTC previously, Fig. 13 shows that constraints are satisfied for these cases as well. The computation time is also significantly improved, as the NN-NL-RG only employs 5 networks while the NN-DTC requires 15 networks, 5 for each of the 3 constraints.

This ARPOD case study shows that despite an increase in state dimensions, from 2 in case study 1 to 6 here, the dataset size required for training viable neural networks need not increase exponentially. This is made possible through careful selection of the parameters used as inputs, especially for NN-NL-RG. NN-NL-RG also show that neural networks with two hidden layers provide better solutions than just one hidden layer, an architecture not often considered in previous works.

C. Processing Time

The ARPOD operation is then simulated with NN-DTC, NN-NL-RG and the conventional NL-RG methods, and have their processing times compared. All simulations were performed for 300 time steps in MATLAB on a Dell XPS 13 laptop with INTEL core i7 and 16 GB memory. Simulations with each of the above methods were run 10 times and the averages were calculated, to eliminate possible effects of background processes running on the computer. The NN-DTC, NN-NL-RG and conventional NL-RG took 24.11sec, 9.02sec and 179.99sec respectively. The conventional NL-RG is significantly slower as at each time step, the plant

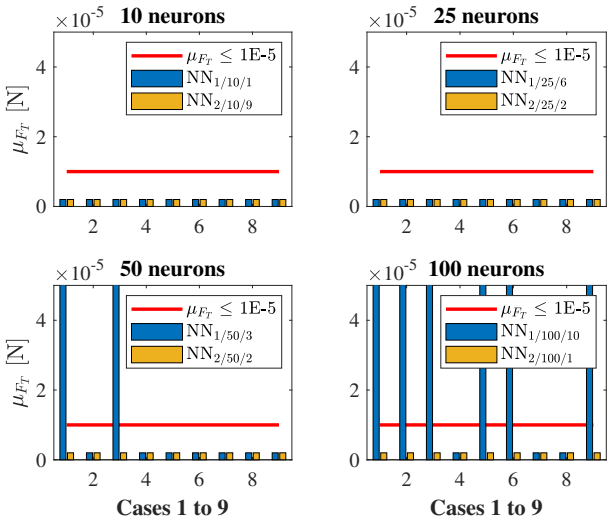

Fig. 12. NN-NL-RG performance evaluation for thrust constraint.

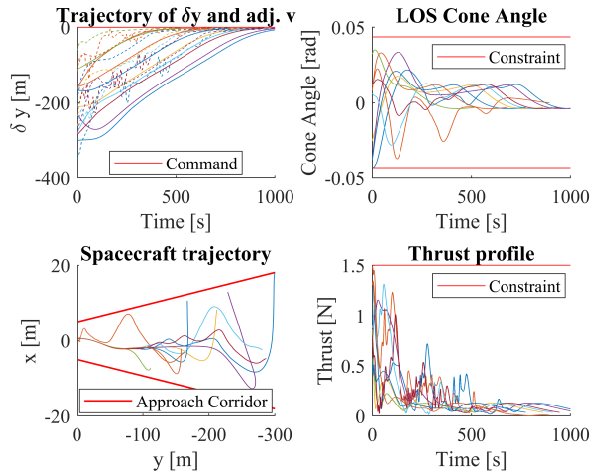

Fig. 13. NN-NL-RG Monte Carlo simulations.

was simulated over the entire 300s while implementing the bisectional search algorithm. However, both proposed methods are almost an order of magnitude faster than the conventional NL-RG. Furthermore, NN-NL-RG is almost 3 times faster than NN-DTC, as NN-NL-RG directly predicts v_k using a single neural network, while NN-DTC is implemented with 3 separate neural networks for each constraint to approximate the MAS, and then solve a minimization problem.

VI. Conclusions

This paper considered alternative approaches to learning reference governor-based solutions to enforcing constraints in nonlinear systems. We first showed the limitations of a classification neural network approach to the MAS approximation. To overcome that, we presented two alternative approaches, both relying on regression neural networks. The first approach, termed NN-DTC, uses neural networks to approximate the distance to constraints. The second approach, termed NN-NL-RG, uses neural networks to approximate the input-output map of the NL-RG. The dataset size required for training of NN-NL-RG is generally smaller than for the NN-DTC approach. There is also no need to train neural networks for each constraint with NN-NL-RG, hence NN-NL-RG has a lower computational footprint than NN-DTC for a system with multiple constraints. While constraints can be aggregated, typically a more complex neural network will still be needed with NN-DTC. Both approaches have been examined in two case studies: a nonlinear second order system with a finite escape time, and a spacecraft ARPOD mission. Satisfactory solutions have been obtained with both approaches. Future work will address additional constraints in ARPOD missions and the use of alternative, more advanced neural network architectures. Comparing the proposed methods with other machine learning models, and testing the proposed methods on public datasets for dynamical systems with constraints, will also be studied.

References

[1] E. Garone, S. Di Cairano, and I. Kolmanovsky, "Reference and command governors for systems with constraints: A survey on theory and applications," *Automatica*, vol. 75, pp. 306–328, 2017.

[2] I. Kolmanovsky, E. Garone, and S. Di Cairano, "Reference and command governors: A tutorial on their theory and automotive applications," in *2014 American Control Conference*, 2014, pp. 226–241.

[3] E. Gilbert and I. Kolmanovsky, "Discrete-time reference governors for systems with state and control constraints and disturbance inputs," in *Proceedings of 1995 34th IEEE Conference on Decision and Control*, vol. 2, 1995, pp. 1189–1194 vol.2.

[4] K. Ikeya, K. Liu, A. Girard, and I. V. Kolmanovsky, "Learning to satisfy constraints in spacecraft rendezvous and proximity maneuvering: A learning reference governor approach," in *AIAA SCITECH 2022 Forum*, 2022, p. 2514.

[5] K. Liu, N. Li, I. Kolmanovsky, D. Rizzo, and A. Girard, "Safe learning reference governor: Theory and application to fuel truck rollover avoidance," *Journal of Autonomous Vehicles and Systems*, vol. 1, no. 4, 2022.

[6] Z. Li, T. Chu, and U. Kalabić, "Dynamics-enabled safe deep reinforcement learning: Case study on active suspension control," in *2019 IEEE Conference on Control Technology and Applications (CCTA)*, 2019, pp. 585–591.

[7] C. J. Ong, S. S. Keerthi, E. G. Gilbert, and Z. Zhang, "Stability regions for constrained nonlinear systems and their functional characterization via support-vector-machine learning," *Automatica*, vol. 40, no. 11, pp. 1955–1964, 2004.

[8] M. Lanchares, I. Kolmanovsky, A. Girard, and D. Rizzo, "Reference governors based on online learning of maximal output admissible set," in *ASME 2019 Dynamic Systems and Control Conference*. American Society of Mechanical Engineers, 2019.

[9] H. Jahagirdar, S. S. Keerthi, and M. Ang, "Reference governor control of constrained feedback systems using neural networks," in *Proceedings of the IEEE Internatinal Symposium on Intelligent Control*. IEEE, 2002, pp. 223–227.

[10] D. Masti, V. Breschi, S. Formentin, and A. Bemporad, "Direct data-driven design of neural reference governors," in *2020 59th IEEE Conference on Decision and Control (CDC)*, 2020, pp. 4955–4960.

[11] E. Gilbert and K. Tan, "Linear systems with state and control constraints: the theory and application of maximal output admissible sets," *IEEE Transactions on Automatic Control*, vol. 36, no. 9, pp. 1008–1020, 1991.

[12] A. Bemporad, "Reference governor for constrained nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 43, no. 3, pp. 415–419, 1998.

[13] B. M. Wilamowski, "Neural network architectures and learning algorithms," *IEEE Industrial Electronics Magazine*, vol. 3, no. 4, pp. 56–63, 2009.

[14] C. D. Petersen, K. Hobbs, K. Lang, and S. Phillips, "Challenge problem: Assured satellite proximity operations," in *31st AAS/AIAA Space Flight Mechanics Meeting*, 2021.