2021 European Control Conference (ECC)
June 29 - July 2, 2021. Rotterdam, Netherlands

Model-free Control Design Using Policy Gradient Reinforcement Learning in LPV Framework

Yajie Bao and Javad Mohammadpour Velni

Abstract— **This paper presents an off-policy policy gradient reinforcement learning (RL) approach to control nonlinear systems in linear parameter-varying (LPV) framework. The (parameter-varying) controller is learned from the closed-loop trajectories using off-policy actor-critic methods for RL, instead of being designed based on the system model. To decrease constraint violation (and hence improve safety), exploration around a valid control sequence is proposed to facilitate learning before applying the real policy. Additionally, the valid control sequence is used to determine the complexity of the actor and critic neural networks (NN). The uncertainties in the evolution of LPV scheduling variables are tackled by generating episodes with varying scheduling trajectories. Adapting to unseen scheduling signals is empirically investigated and fine-tuning is used to refine the learned controller. Furthermore, the prior knowledge of the control law (e.g., static state feedback) is embedded in designing the structure of actor NN. Experiments using two numerical examples and a control moment gyroscope (CMG) simulation model illustrate the success of the proposed approach for stabilization and tracking control in the presence of uncertainty in LPV scheduling variables.**

I. Introduction

Linear parameter-varying (LPV) framework has been a popular approach to model nonlinear dynamical systems in terms of parametric linear models that depend on external *scheduling variables* [1]. More recently, data-driven methods have been developed for global identification of state-space LPV models with uncertainty quantification using only input/output data [2]. In an LPV model, the dynamical mapping between the control inputs and the outputs is linear, which facilitates model-based control synthesis. However, this mapping depends on the scheduling variables, which are used to capture nonlinear behaviors. Most works on LPV identification and control assume an affine scheduling dependency, which limits the expressiveness of models and the scope of application. Arbitrary scheduling dependency can be learned from data [3], [4], [5] but increases the complexity of mathematical analysis.

For control design using LPV models, there are several sources of uncertainty that include disturbance, noise, plant-model mismatch, and uncertainties in the evolution of the scheduling variables. Generally, the scheduling variables can be measured at current time instant but unknown in the future. Prior knowledge on the scheduling variables (e.g., rate-of-variation) can be used to reduce the uncertainty [6]. Using constant scheduling variables within the prediction horizon in the design of model predictive control (MPC) can work for slowly varying scheduling variables but degrades the control performance [7]. To cope with the scheduling uncertainty, robust and stochastic control design are then needed. The authors in [6] presented a tube-based MPC design approach for stabilizing constrained LPV models with affine scheduling dependency as a more conservative but implementable approximation of the min–max feedback control problem [8]. Instead, [9] proposed a stochastic MPC approach for stabilizing constrained LPV systems with arbitrary scheduling dependency using scenario optimization.

Model-based control depends on the accuracy and is subject to the complexity of the model while model-free control can use the trajectories of systems to directly design controller without the need for model identification. For instance, our recent work in [2], [5] used NN to identify LPV models and achieved high accuracy; but unfortunately, the approaches in [6] that assumes an affine scheduling dependency and [9] that assumes that terminal set and terminal control law exist cannot be directly applied to synthesize a controller based on the identified NN-based LPV models. This is because the identified models are not affine and the terminal set and terminal control law are difficult to compute due to the high complexity of the learned NN-based models (it should be noted that data-driven methods can be developed to approximate the terminal set and terminal control law, but that is beyond the scope of this paper). Reinforcement learning (RL) can therefore be used to learn a controller from the closed-loop trajectories. This paper aims to investigate the application of model-free RL for both stabilization and tracking control of (generally) nonlinear systems in LPV framework.

Classical RL assumes a stationary environment model (i.e., the transition probability and reward functions do not vary with time) and finds a sequence of actions (i.e., a policy, a.k.a. control law) for the agent such that the reward function is optimized in an average sense for every initial state of the system. However, LPV models vary with the scheduling variables, resulting in dynamically varying environments. To adapt to the varying contexts, existing RL algorithms for dynamically varying environments distinguish different contexts and find policies for each context [10]. For control design in LPV framework, we propose to use a prior knowledge on the evolution of scheduling variables for generating representative scheduling and system trajectories for RL, investigate the effects of scheduling trajectories on control performance, and use fine-tuning [11] to refine the learned

This work was financially supported by the United States National Science Foundation under award #1912757.

The authors are with School of Electrical & Computer Engineering, University of Georgia, Athens, GA 30602 yajie.bao@uga.edu, javadm@uga.edu.

controller. Moreover, we propose to use a valid control sequence to help exploration and select the structure of NN, and examine the imposition of a-priori known controller structure on the structure of the policy network to facilitate the learning and analyzing of the controller.

To the best of our knowledge, this paper is the first to examine the application of RL to control nonlinear systems in LPV framework. The remainder of this paper is organized as follows: Section II describes the problem statement and introduces off-policy policy gradient reinforcement learning. Control of systems in the LPV framework using RL is introduced in Section III. Section IV presents our experimental results using two numerical examples and a control moment gyroscope (CMG) model, which validate the performance of the proposed methods. Concluding remarks are finally provided in Section V.

II. Problem Statement and Preliminaries

Let us consider a constrained nonlinear system described by the following state-space LPV model with the initial state $x(0)$:

$$x(k+1) = A\left(\theta(k)\right)x(k) + B\left(\theta(k)\right)u(k), \tag{1}$$

$$y(k) = C\left(\theta(k)\right)x(k), \tag{2}$$

$$x(k) \subseteq \mathbb{X}, u(k) \subseteq \mathbb{U}, \tag{3}$$

where $k \in \mathbb{Z}$ is the discrete time variable, $\theta \in \Theta \subseteq \mathbb{R}^{n_\theta}$ is the vector of scheduling variables, u is the vector of control inputs, x is the state vector, and $y \subseteq \mathbb{R}^{n_y}$ is the output vector. Furthermore, $\mathbb{X} \in \mathbb{R}^{n_x}$ is the state constraint set and $\mathbb{U} \in \mathbb{R}^{n_u}$ is the input constraint set. A, B, and C are smooth matrix functions of $\theta(k)$. We assume that $x(k)$, $y(k)$ and $\theta(k)$ can be measured at each time instant k while the future behavior of θ is not exactly known at k. Using Θ to describe the future scheduling variables for control design is too restrictive and thus knowledge on the evolution of θ has been explored. In practice, θ generally varies within a bounded rate-of-variation, i.e., $\forall k \in \mathbb{N}, |\theta(k+1)-\theta(k)| \leq \delta$, which gives a "cone" expanding outwards from the current $\theta(k)$ to describe the possible future trajectories of θ. Other instances of the knowledge on the future θ that have been explored in the literature can be found in [6].

This paper considers both stabilizing and tracking control of nonlinear systems in the LPV framework. For stabilizing control design, the problem is to learn a policy $\pi : \mathbb{X} \times \Theta \to \mathbb{U}$ for (1) using generated trajectories with selected scheduling trajectories such that the system is steered from some initial state x_0 to the origin by π. For tracking control, the problem is to learn a policy π for (1) and (2) such that the tracking error $e(k) = y(k) - y_r(k)$ resides in the ϵ-ball $\mathcal{B}(\epsilon)$ after finite time despite the variations of θ given a reference signal y_r and an error tolerance $\epsilon \in \mathbb{R}_+$.

A. Off-policy Policy Gradient Reinforcement Learning

An RL agent aims to learn a policy function $a = \pi(s)$[*1] that gives action a at a state s such that the expected return $V^\pi(s_0)$ (a.k.a., value function) within a horizon T is maximized, i.e.,

$$\begin{aligned}\pi^* &= \arg\max_\pi V^\pi(s_0) \\ &= \arg\max_\pi \mathbb{E}_{\tau\sim p(\tau|s_0,\pi)}\left[\sum_{t=0}^{T-1}\gamma^t r(s_t,\pi(s_t))\right]\end{aligned} \tag{4}$$

where $\gamma \in (0,1]$ is the discount factor and $p(\tau|s_0,\pi)$ represents the probability of a trajectory $\tau = (s_0, a_0, s_1, \cdots, a_{T-1}, s_T)$ that starts from s_0 under policy π and

$$p(\tau|s_0,\pi) = p(s_0)\prod_{t=0}^{T-1} p(s_{t+1}|s_t,a_t),$$

in which $p(s_{t+1}|s_t,a_t)$ is the state transition model (i.e., dynamics of the environment). For model-free RL, $p(s_{t+1}|s_t,a_t)$ is unknown and τ is generated for learning by interacting with the real system.

Policy gradient algorithms for RL use parameterized policy functions $\pi_\psi(s)$ with parameters ψ (e.g., using a neural network). Moreover, action-value function $Q_t^\pi(s,a) = \mathbb{E}_{a\sim\pi}[G_t|s_t = s, a_t = a]$ is used to assess the expected return of a pair of state and action (s,a) following π where $G_t = \sum_{k=0}^{T}\gamma^k r_{t+k+1}$. The update law of the model is based on the policy gradient theorem in [12]. Furthermore, actor-critic methods use parameterized action-value functions (a.k.a. critic) $Q_\phi(s,a)$ with parameters ϕ while the actor refers to $\pi_\psi(s)$.

Off-policy policy gradient uses a known behavior policy π_D to collect samples and estimate $Q(s,a)$ with regard to the target policy π_ψ. In this way, policy improvement is achieved. Moreover, off-policy approaches do not require full trajectories, and experience replay can be used to improve sample efficiency. Deterministic policy gradient makes deterministic decision but is hard to guarantee enough exploration unless there is sufficient noise in the environment [13]. Either adding noise to the deterministic policy or using stochastic behavior policies can help the exploration for training.

III. Control of Nonlinear Systems in LPV Framework Using RL

In this section, we present the adaptation of off-policy policy gradient RL to LPV control. Specifically, the challenges associated with uncertainties in the scheduling variables and the architecture design of actor NN are tackled.

A. Environment Description of LPV Models

For an RL-based controller design, we first need to set up the environment for the agent to interact with. The environment can be described by a set of states s, a set of actions a, the dynamics/transition model p (which is usually unknown), a reward function r, and a discount factor γ. The state of the environment s should be sufficient for decision making, as model-free RL assumes that the dynamic model p is unknown and selects actions only based on s without explicitly learning p. Authors in [14] show that employing the history of the operation as the environment state s allows

[1]Here, we only consider deterministic policies for control.

the actor to implicitly build a process model. In our case, s consists of the control inputs, observed outputs, tracking errors, and scheduling variables in the past time steps.

Action a is the control input that is computed by the policy function. The family of policy functions affects the analysis of the learned controller. Multi-layer perceptrons (MLP) are commonly used to represent policy functions. To select a proper size for MLP, we propose to find an MLP that can well approximate the initial controller using training data generated offline by sampling the feasible states and evaluating the control law and use it as a benchmark for policy NN design. Moreover, NN architecture can be designed to be in a targeted controller form. For control of LPV models, we use an MLP f_{NN} to model $K = f_{NN}(\theta(k))$ instead of $a = f_{NN}(s)$ such that a controller in the form of $u = K(\theta(k))x$ is obtained, which is more suitable for control synthesis.

The dynamic model p of LPV systems is unknown and varies with the scheduling variable $\theta(k)$. Given a value of $\theta(k)$, (1) and (2) constitute a linear model and given a scheduling trajectory $\{\theta(t)\}_{t=0}^{T}$, the evolution of the model is determined. For training the RL agent, we can generate arbitrary valid scheduling trajectories and the agent makes a decision only based on the present scheduling variable. However, these trajectories for training are not necessarily identical to the scheduling variable for control, i.e., in closed-loop operation. This discrepancy between the scheduling trajectories for training and control can degrade the closed-loop performance when applying the trained agent to control LPV model, and poses the trade-off between performance and sample efficiency. Moreover, matrix functions A, B and C are generally bounded, which results in bounded differences in x and y given different scheduling trajectories. Given the bounded differences of the model, RL can still be used to improve the initial controller. Using $\pi, p(s', r|s, a)$ to represent the target policy and control environment and $\pi_D, p_\theta(s', r|s, a)$ to denote the behavior policy and training environment, Theorem 1 gives a sufficient condition for improving the true returns V^π in the control environment based on the returns $\hat{V}^\pi$ in the training environment.

Theorem 1 ([15]): Let the expected total variation distance (TV-distance) between two transition distributions $\mathbb{E}_{s\sim\pi_{D,t}}\left[D_{TV}\left(p(s', r|s, a)\|p_\theta(s', r|s, a)\right)\right]$ be bounded at each time step by ϵ_m and the policy divergence $\mathbb{E}_{s\sim\pi_t}\left[\pi\|\pi_D\right]$ be bounded by ϵ_π. Then, the true returns and model returns of the policy are bounded as

$$V^\pi \geq \hat{V}^\pi - \underbrace{\left[\frac{2\gamma r_{\max}(\epsilon_m + 2\epsilon_\pi)}{(1-\gamma)^2} + \frac{4r_{\max}\epsilon_\pi}{1-\gamma}\right]}_{C(\epsilon_m,\epsilon_\pi)},$$

The reward function r defined for control of nonlinear systems should be negatively related to the tracking error. $r_k = -\|e(k)\|_1$ is used in our experiments. Discount factor γ shows how important future rewards are to the current state s and is used to moderate the stability problems. Therefore, r is set to be close to 1 for control to facilitate the convergence of the tracking error.

B. Off-policy Policy Gradient Algorithm for Control Design

In this section, we consider deep deterministic policy gradient (DDPG), an off-policy actor-critic algorithm proposed in [16] for control design in LPV framework. DDPG learns both a Q-function and a deterministic policy. Specifically, the critic is trained by minimizing the following mean-squared Bellman error (MSBE) loss with stochastic gradient descent:

$$\begin{aligned} L(\phi, \mathcal{D}_{\text{repl}}) =& \mathbb{E}_{(s,a,r,s',d)\sim\mathcal{D}_{\text{repl}}}[(Q_\phi(s,a) \\ & -(r+\gamma(1-d)Q_{\phi_{\text{targ}}}(s', \pi_{\psi_{\text{targ}}}(s'))))^2] \end{aligned} \quad (5)$$

where $Q_{\phi_{\text{targ}}}$ is the target network with parameters ϕ_{targ} to evaluate the target Q-values, $\pi_{\psi_{\text{targ}}}$ is the target policy network with parameters ψ_{targ} to compute an action which approximately maximizes $Q_{\phi_{\text{targ}}}$, $\mathcal{D}_{\text{repl}}$ is reply buffers of transitions (s, a, r, s', d), and d indicates whether s' is terminal.

For policy learning of the actor, a deterministic policy $\pi_\psi(s)$ is learned by solving

$$\max_{\psi} \ \mathbb{E}_{s\sim\mathcal{D}_{\text{repl}}}[Q_\phi(s, \pi_\psi(s))],$$

where Q-function parameters ϕ are fixed, using gradient ascent with respect to policy parameters only. Since the policy is deterministic, the agent may not be able to explore a sufficient variety of policies at the training time to find the optimal policy. To help with the exploration, noise is added to actions at training time, i.e., the exploration policy $\pi_{\text{expl}}(s) = \pi_\psi(s)+\mathcal{N}$ where $\mathcal{N}$ denotes the selected noise. $\mathcal{N}$ can be an Ornstein-Uhlenbeck process [17] or uncorrelated mean-zero Gaussian noise. When exploiting the learned policy, $\pi_\psi(s)$ is evaluated without adding noise. However, for complex systems, a small noise added at each time step can be insufficient for exploration but significantly change the system long-term behaviors. Therefore, We reduce the scale of the noise over the course of training by $\sigma_n(\mathcal{N}) = \frac{\sigma_0}{1+n}$ where n is the number of episodes and σ_0 is the initial noise scale such that the exploration at the beginning is sufficient and the exploration policy at the end of the training time converges to the deterministic policy.

Moreover, uniform-random action selection before running the real policy can help the exploration but may result in control inputs that cause system failure. Instead, similar to the exploration policy, we add Gaussian noise to the control sequence generated by the initial controller and use the noisy sequences to collect open-loop trajectories. The noisy sequences can contain the optimal control sequence and these trajectories can improve the convergence of the learning. Moreover, randomly initialized parameters of the actor and critic neural networks can be far from the optimal solution. We can use the open-loop trajectories to train the actor and critic network separately before RL, which gives a good initialization and improves the convergence rate.

The training procedure of DDPG for LPV system control is summarized as follows:

1: **procedure** DDPG FOR CONTROL OF LPV SYSTEMS
2: Input: initial control sequence $\{u(k)\}_{k=0}^{T}$, selected scheduling sequence $\{\theta(k)\}_{k=0}^{T}$, initial policy parameters ψ, Q network parameters ϕ, empty replay buffer

$\mathcal{D}$; the maximal number of steps $N_{\max}$ for training, the number of steps N_{start} before applying π_ψ.
Set target parameters: $\psi_{\text{targ}} \leftarrow \psi$, $\phi_{\text{targ}} \leftarrow \phi$; $n = 0$.
while $n < N_{\max}$ **do**
Observe state s and select action $a = \text{clip}\left(\pi_{\psi(s)} + \epsilon, a_{low}, a_{high}\right)$, where $\epsilon \sim \mathcal{N}$.
if $n < N_{start}$ **then**
Select $a = \text{clip}\left(u(k) + \epsilon, a_{low}, a_{high}\right)$, where $k = \text{MOD}(n/T)$ and $\epsilon \sim \mathcal{N}$
end if
Execute a in the environment where the scheduling variable is $\theta(k)$; $n = n + 1$.
Observe next state s', reward r and done signal d; store (s, a, r, s', d) in replay buffer $\mathcal{D}$.
If s' is terminal, reset environment state.
if update **then**
for each update **do**Randomly sample a batch B of transitions from $\mathcal{D}$.
Evaluate targets: $y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \pi_{\psi_{\text{targ}}}(s'))$
Update Q network: $\phi \leftarrow \phi - \alpha \bigtriangledown_\phi \frac{1}{|B|}\sum_{(s,a,r,s',d)\in B}\left(Q_\phi(s,a) - y(r,s',d)\right)^2$
Update policy network: $\psi \leftarrow \psi + \beta \bigtriangledown_\psi \frac{1}{|B|}\sum_{s\in B} Q_\phi\left(s, \pi_\psi(s)\right)$
Update target networks: $\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi$ and $\psi_{\text{targ}} \leftarrow \rho\psi_{\text{targ}} + (1-\rho)\psi$
end for
end if
end while
end procedure

The scheduling trajectories for training are selected to contain or be similar to the scheduling variables for system operations such that the discrepancy between the training environments and real operations is bounded and the learned controller still works by Theorem 1. Moreover, the collected closed-loop trajectories from operations can be used for fine-tuning the actor and critic networks to adapt to new environments. Fine-tuning reuses parts of the previously trained networks and parameters to facilitate learning. A detailed discussion on fine-tuning can be found in [7].

IV. Experimental Results and Validation

In this section, the proposed methods are validated using numerical examples and a complex physical system model.

A. Stabilizing Control

Consider a second-order LPV model in [6] described by

$$x(k+1) = \left(\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix} + \begin{bmatrix}0.1 & 0\\ 0 & 0.1\end{bmatrix}\theta_1(k) + \begin{bmatrix}0.5 & 0.5\\ 0 & 0\end{bmatrix}\theta_2(k) + \begin{bmatrix}0 & 0\\ 0 & 0.2\end{bmatrix}\theta_3(k)\right)x(k) + \begin{bmatrix}0.5\\ 1\end{bmatrix}u(k) \tag{6}$$

with constraints and scheduling sets as

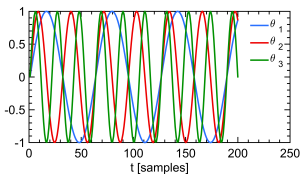
$$\mathbb{X} = \{x \in \mathbb{R}^2 | \|x\|_\infty \le 6\}, \mathbb{U} = \{u \in \mathbb{R} | \, |u| \le 1\}$$
$$\Theta = \{\theta \in \mathbb{R}^3 | \|\theta\|_\infty \le 1\}.$$

This example is to demonstrate the proposed method for stabilizing control and investigate the effects of scheduling trajectories on control performance.

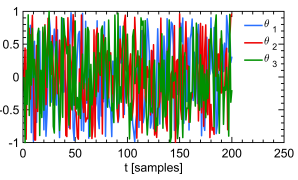
1) Network Architecture and Hyperparameters: We use an MLP with three hidden layers to model the policy network π_ψ. Each of the hidden layers contains 16 units and uses the rectified linear unit (ReLU) activation function while the output layer uses a linear activation function. Moreover, another MLP with three hidden layers is used to model the Q network Q_ϕ. Each of the hidden layers contains 32 units and uses ReLU as activation function while the output layer uses linear activation function. Furthermore, we choose $\rho = 0.001$ for target network update. The reward function is chosen as $r_t = -x(t+1)^{\mathrm{T}}x(t+1) - a_t^2$ and the discount factor is $\gamma = 0.99$. The agent takes actions sampled from a uniform distribution over the action space for the first 100 steps to help with the exploration and warm up the network. Moreover, Ornstein–Uhlenbeck process [17] with standard deviation $\sigma = 0.3$ is added to the action during training for exploration. The maximal length of an episode is considered to be $T = 200$. We trained the agent for $50,000$ steps with limit of the sequential memory as $100,000$ and batch size as 32 using Adam optimizer with a learning rate of 10^{-3}.

2) Experimental Setup: First, we train and test the agent using the same scheduling trajectory to examine the capability of the proposed method to stabilize the system. Next, we test the learned controller for different scheduling variables to examine the adaptability of the learned control law. Then, we use fine-tuning to refine the learned controller and examine the improvement.

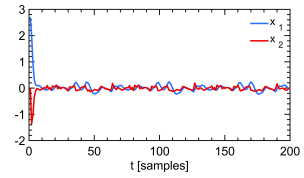
3) Results and Discussion: The scheduling signals and control results are shown in Fig. 1. Mean absolute error (MAE) is used to measure the control performance. We considered sinusoidal in (a) and random in (b) scheduling signals. Subplots (c) and (d) show the control results using the same scheduling trajectory for both training and testing, which demonstrates that the proposed method can stabilize the system with small errors when there is no discrepancy between the learning environment and real operations. Subplots (e) and (f) present the control results using one scheduling signal for training and another for testing, which shows that the learned controller can still stabilize the system when the discrepancy in scheduling signal(s) is limited although the control performance slightly degrades. Moreover, we fine-tuned the controller learned in (c) for $40,000$ steps using a decreased learning rate 10^{-6} to adapt to the scheduling signal (b). Similarly, we fine-tuned the controller in (d) for $40,000$ steps using a decreased learning rate 10^{-4} to adapt to the scheduling signal (a). Subplots (g) and (h) show the control results using fine-tuning, which demonstrates that fine-tuning can facilitate the RL agent with less training steps and even better performance than starting from the scratch (comparing (c) and (h)). It is noted that the transferability is not symmetric (i.e., adapting from (b) to (a) is easier than from (a) to (b)), as discussed in [18], which demonstrates the importance of scheduling signal selection for training.



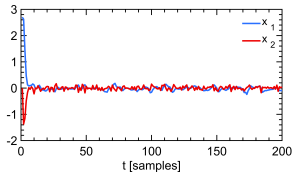
(a) Sinusoidal scheduling signal.



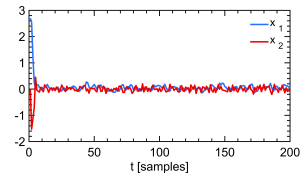
(b) Random scheduling signal.



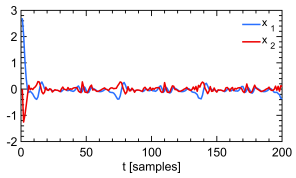
(c) Closed-loop system states with scheduling signal in (a) for both training and testing. $MAE = [0.0765 \quad 0.0549]^{\mathrm{T}}$.



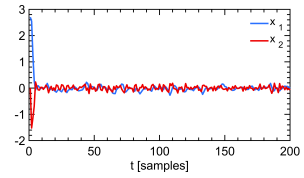
(d) Closed-loop system states with scheduling signal in (b) for both training and testing. $MAE = [0.0553 \quad 0.0541]^{\mathrm{T}}$.



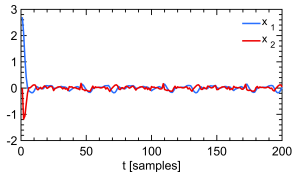
(e) Closed-loop system states with scheduling signal in (a) for training and (b) for testing. $MAE = [0.0720 \quad 0.0750]^{\mathrm{T}}$.



(f) Closed-loop system states with scheduling signal in (b) for training and (a) for testing. $MAE = [0.0869 \quad 0.0716]^{\mathrm{T}}$.



(g) Fine-tuning the controller in (c) for (b). $MAE = [0.0627 \quad 0.0679]^{\mathrm{T}}$.



(h) Fine-tuning the controller in (d) for (a). $MAE = [0.0572 \quad 0.0480]^{\mathrm{T}}$.

Fig. 1: Scheduling signals and control results using a stabilizing RL agent. MAE is calculated after 10 steps.

B. Tracking Control

Consider a single-input single-output LPV model with two states in [19] described by

$$x(k+1) = \left(\begin{bmatrix} 0.95 & 1 \\ 0 & -0.59 \end{bmatrix} + \begin{bmatrix} 0 & 0.5 \\ 0 & 0 \end{bmatrix} \theta_1(k) + \begin{bmatrix} 0 & 0 \\ 0.2 & 0 \end{bmatrix} \theta_2(k) \right) x(k) + \begin{bmatrix} 1 \\ 0.5 \end{bmatrix} u(k) \tag{7}$$

$$y(k+1) = \left(\begin{bmatrix} 0.8 & -0.6 \end{bmatrix} + \begin{bmatrix} 0 & -0.03 \end{bmatrix} \theta_1(k) + \begin{bmatrix} 0.04 & 0 \end{bmatrix} \theta_2(k) \right) x(k) \tag{8}$$

with constraints and scheduling sets as

$$\mathbb{X} = \{x \in \mathbb{R}^2 | -6 \leq x_1 \leq 4, -4 \leq x_2 \leq 6\},$$
$$\mathbb{U} = \{u \in \mathbb{R} | \, |u| \leq 1\}, \Theta = \{\theta \in \mathbb{R}^2 | \|\theta\|_\infty \leq 1\}.$$

The same network architecture and hyperparameters as in Section IV-A.1 are used for this example except for the learning rate that is set to 10^{-5} and that we trained the agent for $450,000$ steps. The state of environment is $s_k = [x(k) \quad e(k) \quad \theta(k)]^{\mathrm{T}}$ and the reward function is defined as $r(s_k, a_k) = -e^2(k) - a_k^2$, where $e(k) = y(k) - y_r(k)$, $y_r(k)$ denotes the setpoint, and $a_k = -u(k)$ is the control input. Random scheduling signals similar to Fig. 1(b) were used for this example. Fig. 2 shows the control result, which demonstrates the capability of the proposed method for tracking control purposes.

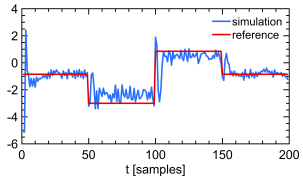


Fig. 2: Piecewise constant reference tracking control results for second example.

C. Policy Network Design

In this example, we demonstrate that the proposed method for policy network design can facilitate the RL control, using experiments on a 4 degree-of-freedom control moment gyroscope (CMG) simulation model from [20]. The detailed plant description can be found in [21]. The states of the simulation model consist of angles q_i and angular speeds $\omega_i, i = 1, 2, 3, 4$ of the 4 gimbals and it is generally required to control q_3 and q_4 using torques τ_1 and τ_1 provided by two dc motors. We use the LPV representation of CMG in [21] to inform the policy network design. The states, outputs and scheduling variables of the obtained LPV model are

$$\begin{aligned} x &= [q_3 \quad q_4 \; \omega_2 \quad \omega_3 \quad \omega_4]^{\mathrm{T}}, \\ y &= [q_3 \quad q_4]^{\mathrm{T}}, \\ \theta &= [q_2 \quad q_3]^{\mathrm{T}} \end{aligned} \tag{9}$$

The controller is designed to be in the static state-feedback form with parameter-varying gain, i.e., $u = K(\theta)x$, where K is a smooth nonlinear matrix function represented by NN.

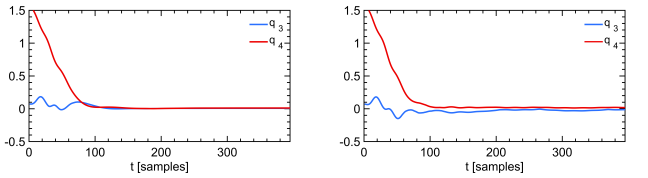
1) Network Architecture and Hyperparameters: An MLP with two hidden layers is used to model $K(\theta)$. Each of the hidden layers contains 512 units, uses ReLU as activation function and is followed by a batch normalization layer[2] to normalize different physical units of features. The output layer uses linear activation function and the output of this MLP is multiplied by x to constitute the policy network. For Q network, the state s is transformed by an MLP with one 16-unit layer followed by one 32-unit layer and action a is transformed by an MLP with one 32-unit hidden layer. The transformed s and a are concatenated and then transformed by an MLP with two 512-unit hidden layers to estimate Q value. All the hidden layers are followed by a batch normalization layer. Gaussian noise $\mathcal{N}(0, 0.03u(k))$ is added to an initial control sequence $\{u(k)\}_{k=1}^{T}$ from an MPC at time k for the first $40,000$ steps to help with the exploration. Moreover, an Ornstein–Uhlenbeck process with

[2]We refer to DDPG in Keras [22] for implementing the proposed method.

initial standard deviation $\sigma = [0.1 \quad 0.4]^{\mathrm{T}}$ is added to the action at training time for exploration. Also considered are the discount factor $\gamma = 0.99$ and $\rho = 0.001$ for target network update. The maximal length of an episode is 400. We trained the agent for $1,500$ episodes with limit of the sequential memory as $50,000$ and batch size as 64 using Adam optimizer with a learning rate as 10^{-4} for the Q network and 10^{-5} for the policy network.

2) Experimental Settings: The state of environment s_t consists of $\{u(t+k-l), \theta(t+k-l), x(t+k-l), e(t+k-l)\}_{k=1}^{l}$ where $l = 4$ denotes the memory length and is used for the agent to implicitly build the process model as discussed in Section III-B. The reward function is chosen as $r_t = -\|e(t+1)\|_1$ and the control objective is to track reference $q_3 = q_4 = 0$. We first use the network architecture described in Section IV-C.1 to learn a controller in the form of $u = K(\theta)x$. Then, we learn $u = f_{NN}(s)$ where f_{NN} shares the same architecture with the network that represents $K(\theta)$ for comparison using the same hyperparameters except for the learning rate that is assumed to be 10^{-3} for the Q network and 10^{-4} for f_{NN}.

3) Results and Discussion: Fig. 3 shows the control results. Despite the CMG platform being far more complex than the two numerical examples discussed earlier, using the techniques proposed in Section III-B for exploration and network design and initialization, very precise tracking performance was achieved in moderate training episodes, which demonstrates that the proposed method can improve the convergence. Moreover, subplot (a) with constraints on the network structure achieved better tracking performance than (b), which shows that the policy network can be designed for analysis without degrading the performance.



(a) Control result using $u = K(\theta)x$.

(b) Control result using $u = f_{NN}(s)$.

Fig. 3: Control results for the CMG platform. The unit for angles is radian and the sampling frequency is 0.1 kHz.

V. Concluding Remarks

In this paper, an off-policy deep deterministic policy gradient approach was proposed to control nonlinear systems in LPV framework using closed-loop trajectories. The proposed method proved to perform well when the discrepancy between the training environment and operations is bounded. An algorithm was presented to tackle uncertainties in the scheduling variable by selecting representative scheduling trajectories for training such that the learned controller could be generalized to unseen scheduling signals. Moreover, techniques for assisting exploration, network design and initialization, and refining the learned controller were studied using an initial control sequence. Experiments on two numerical examples, as well as a complex nonlinear model of control moment gyroscopes demonstrated that the proposed approaches could effectively control (both stabilize and track reference trajectories) LPV systems with small errors and improve the convergence of reinforcement learning.

References

[1] J. Hanema, "Anticipative model predictive control for linear parameter-varying systems," Ph.D. dissertation, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, 2018.

[2] Y. Bao, J. Mohammadpour Velni, and M. Shahbakhti, "Epistemic uncertainty quantification in state-space LPV model identification using bayesian neural networks," *IEEE Control Systems Letters*, vol. 5, no. 2, pp. 719–724, 2020.

[3] R. Tóth, V. Laurain, W. X. Zheng, and K. Poolla, "Model structure learning: A support vector machine approach for LPV linear-regression models," in *2011 50th IEEE Conference on Decision and Control and European Control Conference*. IEEE, 2011, pp. 3192–3197.

[4] S. Z. Rizvi, J. Mohammadpour Velni, F. Abbasi, R. Tóth, and N. Meskin, "State-space LPV model identification using kernelized machine learning," *Automatica*, vol. 88, pp. 38–47, 2018.

[5] Y. Bao, J. Mohammadpour Velni, A. Basina, and M. Shahbakhti, "Identification of state-space linear parameter-varying models using artificial neural networks," in *21st IFAC World Congress*. IFAC, 2020.

[6] J. Hanema, M. Lazar, and R. Tóth, "Heterogeneously parameterized tube model predictive control for LPV systems," *Automatica*, vol. 111, p. 108622, 2020.

[7] Y. Bao, J. Mohammadpour Velni, and M. Shahbakhti, "An online transfer learning approach for identification and predictive control design with application to RCCI engines," in *ASME Dynamic Systems and Control Conference*. ASME, 2020.

[8] J. H. Lee and Z. Yu, "Worst-case formulations of model predictive control for systems with bounded parameters," *Automatica*, vol. 33, no. 5, pp. 763–781, 1997.

[9] G. C. Calafiore and L. Fagiano, "Stochastic model predictive control of LPV systems via scenario optimization," *Automatica*, vol. 49, no. 6, pp. 1861–1866, 2013.

[10] S. Padakandla, "A survey of reinforcement learning algorithms for dynamically varying environments," *arXiv preprint arXiv:2005.10619*, 2020.

[11] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, vol. 113, pp. 54–71, 2019.

[12] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[13] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *ICML*, 2014.

[14] N. A. Spielberg, M. Brown, N. R. Kapania, J. C. Kegelman, and J. C. Gerdes, "Neural network vehicle models for high-performance automated driving," *Science Robotics*, vol. 4, no. 28, 2019.

[15] M. Janner, J. Fu, M. Zhang, and S. Levine, "When to trust your model: Model-based policy optimization," in *Advances in Neural Information Processing Systems*, 2019, pp. 12 519–12 530.

[16] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[17] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion," *Physical review*, vol. 36, no. 5, p. 823, 1930.

[18] Y. Bao, Y. Li, S. Huang, L. Zhang, L. Zheng, A. Zamir, and L. Guibas, "An information-theoretic approach to transferability in task transfer learning," in *2019 IEEE International Conference on Image Processing (ICIP)*, 2019, pp. 2309–2313.

[19] J. Hanema, M. Lazar, and R. Tóth, "Tube-based LPV constant output reference tracking MPC with error bound," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 8612–8617, 2017.

[20] T. R. Parks, "Manual for model 750: Control moment gyroscope," *Educational Control Products, Bell Canyon, CA*, 1999.

[21] H. S. Abbas, A. Ali, S. M. Hashemi, and H. Werner, "LPV state-feedback control of a control moment gyroscope," *Control Engineering Practice*, vol. 24, pp. 129–137, 2014.

[22] F. Chollet *et al.*, "Keras," https://keras.io, 2015.