Robust Speed Control of Ultrasonic Motors Based on Deep Reinforcement Learning of A Lyapunov Function

ABDULLAH MUSTAFA1, TATSUKI SASAMURA1, AND TAKESHI MORITA1
1Graduate School of Frontier Sciences, The University of Tokyo, 5-1-5 Kashiwanoha, Kashiwa, Chiba, Japan
Corresponding author: Abdullah Mustafa (e-mail: ammustafa@s.h.k.u-tokyo.ac.jp).

ABSTRACT Speed control of ultrasonic motors (USM) needs to be precise, fast, and robust; however, this becomes a challenging task due to the nonlinear behavior of these motors including nonlinear response, pull-out phenomenon, and speed hysteresis. However, linear controllers would be suboptimal and unstable, and nonlinear controllers would require expert knowledge, expensive online calculations, or costly model estimation. In this paper, we propose a model-free nonlinear offline controller that can significantly mitigate these challenges. Using deep reinforcement learning (DRL) algorithms, a neural network speed controller was optimized. A soft actor-critic (SAC) DRL algorithm was chosen due to its sample efficiency, fast convergence, and stable learning. To ensure controller stability, a custom control Lyapunov reward function was proposed. The steady-state USM behavior was mathematically modeled for easing controller design under simulation. The SAC agent was designed and trained first in simulation and then further trained experimentally. The experimental results support that the trained controller can successfully expand speed operation range ([0,300] rpm), plan optimal control trajectories, and stabilize performance under varying load torque and temperature drift.

INDEX TERMS Ultrasonic Motors, Speed Control, Deep Reinforcement Learning, Control Lyapunov Function, Soft Actor-Critic

I. INTRODUCTION

Unlike their electromagnetic counterparts, ultrasonic motors (USM) have superior features in terms of their compact structure, silent operation, lightweight, high energy density, no electromagnetic interference, and high torque at low speeds [1]–[3]. USMs are operated by exciting mechanical vibration at the stator surface through applying a high frequency (ultrasonic) voltage signal to a piezoelectric material. This (resonance/off-resonance) vibration is then transferred to a rotor/slider through frictional forces. The slider/rotor speed (a function of vibration amplitude) could be adjusted by controlling the voltage amplitude, driving frequency, or phase difference. Bidirectional speed control can be realized by incorporating phase difference as a control input to adjust the elliptical trajectory on the stator surface. For a single input unidirectional speed control of traveling wave rotary USMs (studied system), the driving frequency is most popular and most adopted in commercial controllers due to its wide speed control range [2]. However, the USM control problem is still rather challenging due to the nonlinearity of the speed response under driving frequency control. USMs are inherently nonlinear due to the nonlinearity of the piezoelectric materials [4], and the nonlinear contact mechanics between the stator and the rotor [5]. As a result, the frequency/speed response is nonlinear under changes in driving frequency, voltage amplitude, load torque, and motor temperature. The response is also discontinuous due to the pull-out phenomenon. As the driving frequency is swept down, the motor speed will continuously increase before it suddenly stops (pull-out) at some critical frequency. This pull-out behavior is a result of the nonlinear contact mechanics and the excessive vibration amplitude near the resonance region. Finally, depending on the frequency sweep direction, speed hysteresis is induced. Compensating these nonlinearities is essential for proper controller design.

As a result, an off-the-shelf linear controller like proportional integral derivative (PID) or linear quadratic regulator (LQR) would be suboptimal in terms of response time, robustness, and stability. With proper controller gain tuning, these linear models can only prove effective within a lin-
earized operation region. With some expert knowledge, gain scheduling of linear controller can improve motor response within wider operation region [6]. Besides linear control, many nonlinear controllers can be applied to handle USM nonlinearity [7]. In [8], [9], a neural network controller was proposed due to its high flexibility for approximating complex functions. In [10], a fuzzy controller was developed using expert knowledge of PID tuning for constructing the fuzzy rules. Furthermore, in [11], neural networks were combined with fuzzy logic to get a controller with neural networks flexibility, and fuzzy logic expertise. In [12], [13], iterative learning and model predictive controllers could adapt to changes in motor behavior by online learning. Despite their clear advantage over linear controllers, the proposed nonlinear controllers are still lacking in a few aspects.

Fuzzy controllers require expert knowledge to craft fuzzy rules that can remain suboptimal. In [10], the PID gains were tuned for three different operation conditions, and the gains were interpolated through a sum of Gaussians for intermediate operation conditions. In addition to the sub-optimality of the interpolation region, extrapolation might result in instability. On the other hand, online iterative controllers in [12], [13] can realize a zero steady error; however, they require running multiple iterations for control output optimization. This would be a feasible solution for constant trajectory tracking but would underperform (higher errors) for continuously changing unknown trajectories. Due to their high flexibility and over-parameterization, the stability and convergence properties of classic neural network controllers remain a challenge. The reported controllers in [8], [9] were trained online with gradient descent through backpropagating the tracking errors. Although the exact training procedure was not reported, we believe the training approach was on-policy (cannot reuse previous interactions with the system) resulting in slow convergence, and local optimality as will be further discussed.

Due to its popularity for both commercial and scientific purposes, SHINSEI USR60 was studied in this research. For reliable speed control, SHINSEI commercial driver limits the driving frequency from reaching resonance frequency with some sufficient margin. In addition, the driving voltage amplitude is relatively high to reduce speed hysteresis. Yet, high voltages are less efficient and less stable due to overheating. Similarly, proposed controllers in the literature (linear controllers and controllers with inherently linear structure (i.e. Fuzzy-PID [10], MIT Control [13])) limit the frequency control range to avoid discontinuity and hysteresis in the speed response. Such constraints can produce reliable controllers at the expense of limiting the motor output.

To fully utilize the capabilities of the USM, the controller should operate without limits while handling nonlinearity and discontinuity. In this paper, we propose a new control scheme based on deep reinforcement learning (DRL). The flexible topology of deep neural networks can adapt to all sorts of USM nonlinearities. In addition, reinforcement learning (RL) can optimize a global controller efficiently from online interactions. The controller is then operated offline to realize a faster response than online model predictive controllers (MPC). The main contributions of this paper are as follows:

1) A mathematical model was developed to simulate the steady-state speed/frequency response of USM under varying torque and temperature drift. This model is essential for controller design.

2) A USM speed deep neural network controller was designed and trained based on a modified Soft Actor-Critic (SAC) algorithm. The reward function was designed based on control Lyapunov function (CLF) to realize an almost stable and robust controller.

3) The proposed controller was validated in both simulation and experiment for the USR60 motor. The results were compared to other controllers; proving the optimality, stability, and robustness of the controller.

The paper will proceed as follows. Section II further discusses the nonlinearity of speed response experimentally, and a simulation model is developed to explain and replicate the nonlinear phenomenon. In section III, the proposed SAC algorithm is discussed, and the controller design is formulated. In section IV, the simulation and experiment results are presented showing the superior performance of the DRL controller. Finally, section V further discusses the controller stability along with some comparative results with other controllers.

II. USM CHARACTERIZATION AND MODELING

A. EXPERIMENTAL SETUP

Shinsei USR60 rotary traveling wave ultrasonic motor was used as a target system for speed control. The motor is driven by two sinusoidal signals with 90° phase difference, 300 V peak-to-peak voltage amplitude, and driving frequency around 40 kHz. Using the commercial driver, a pseudo-sinusoidal signal is generated; however, to get more controllability over the driving signal, a custom setup was developed. Two pure sinusoidal signals were generated using an NF WF1968 function generator. Then, two HSA 4052 amplifiers amplified the signal to the desired level. The motor speed was measured using a UNIPULSE UTM III encoder, and the speed analog signal was received through a UNIPULSE TM380 monitor. The motor’s internal temperature was estimated through a k-type thermocouple fitted inside the motor. The thermocouple was connected to an NF DM2561A digital multimeter for analog temperature measurement. Finally, the external load torque was applied using a MITSUBISHI 2KB-1.2XN power brake.

The setup shown in Fig. 1 allowed communication across these devices. A Python environment was used instead of other popular environments like C or MATLAB due to its flexibility as an open-source language, and its support for state-of-art neural networks libraries. Using Python’s Pyserial library, the PC and the digital signal processor (DSP) were connected via an RS232 connection. Through this connection, commanded driving frequencies and load torques...
were communicated from the PC to DSPACE 1104 board. The commanded frequency was then applied as an analog signal from DSP to the function generator. Varying the analog signal between ±1 varied the driving frequency between 39-45 kHz through frequency modulation. Similarly, for temperature measurement, Python’s PyVisa library enabled communication between the PC and the digital multimeter via general-purpose interface bus (GPIB) connection.

B. EXPERIMENTAL EVALUATION

Using the developed setup, a few nonlinear features of USM speed/frequency response were evaluated. The experiment was conducted several times to confirm the reproducibility of the behavior. In Fig. 2, the bold lines represent the mean behavior, and the transparent area is the standard deviation. In Fig. 2a, under a peak-to-peak voltage amplitude of 300 V, the driving frequency was swept down between 45-39 kHz for different load torques. For any load torque, as the frequency got lower, the speed increased at a continuously changing rate before suddenly dropping to a stop due to the pull-out phenomenon. The pull-out phenomenon is mostly due to the nonlinear contact mechanics. Most controllers would set a lower bound on the driving frequency to avoid this pull-out region which would limit the motor output. With increasing load torque, the maximum speed decreases, and it becomes infeasible to operate the motor at high frequencies. Additionally, the pull-out frequency \( f_{\text{pull-out}} \) increases which forces conventional control schemes to set a more conservative limit on the minimum driving frequency. Another nonlinear feature of speed response is speed hysteresis where the response depends on the driving frequency sweep direction as in Fig. 2b. Under a no-load and 300 V voltage amplitude, the driving frequency was swept both down and up. Unlike the sweep down the case, the sweep up response resumes operation at a different frequency (pull-in frequency \( f_{\text{pull-in}} \)) which is higher than the pull-out frequency. The sweep-up speed is lower and suffers from high variations; making the control more challenging. Temperature drift is experienced by USMs where temperature rise changes the behavior of the piezoceramic and the overall system. Fig. 2c shows the speed response under increasing temperature. As the temperature increased, the speed response was almost identical for low speeds where the slope was relatively low. However, at higher speeds, temperature rise caused a drift of speed response (pull-out frequency) to lower frequencies, as well as a slight increase in peak speed.

C. SIMULATION MODEL

To design, train, and evaluate the proposed DRL controller, a simulation model is necessary for prototyping before experimental studies. To simulate the USM behavior, there exist several approaches that vary in complexity and accuracy including finite element method (FEM) [14], variational methods [5], [15], and equivalent circuit model (ECM) [16]. The finite element method is one approach with high accuracy and can be applied during the design phase; however, the contact mechanics cannot be simulated as electric components. As a result, the load torque effect, and pull-out discontinuity cannot be reproduced. Finally, variational methods can accurately model the system by developing a set of differential equations describing the system behavior. Again, starting with some initial condition, reaching a steady-state requires costly calculations. For our purposes, only a steady-state solution is desired, and thus, we extend the differential model proposed by Hagood in [15] to solve directly for steady-state solutions efficiently.

For the USR60 model, two 9th order bending mode vibrations (B09) are excited and superimposed to form a traveling wave. Under a given voltage amplitude \( V_0 \), and angular frequency \( \omega = 2\pi f \), the stator vibrates with amplitude \( w_0 \) and phase difference \( \psi \). The stator vibration is excited by a piezo ring with a coupling factor \( \Theta \) representing electromechanical conversion. The stator is modeled as a second-order system parameterized by \( L_m, R_m, \) and \( C_m \) representing mechanical inductance (mass), resistance (damping), and capacitance (inverse stiffness) respectively. When preload is applied between the rotor and the stator, mechanical power is transferred through frictional forces. Thus, the contact forces are introduced into the stator model as in (1). The contact force has two components \( F_N \) and \( F_T \); representing normal and tangential forces respectively.

\[
L_m \dot{w} + R_m w + \frac{1}{C_m} w = \Theta V + F_N + F_T \tag{1}
\]

where \( V = V_0 e^{j(\omega t + \psi)}, w = w_0 e^{j\omega t} \)

The sinusoidal traveling wave on the circular stator surface forms \( N \) identical peaks under idealized assumptions. At steady-state, a rotating reference frame fixed to a unique traveling peak is used to model the contact mechanics. In Fig. 3, an illustration of the contact mechanics is shown. When a rotor of normal stiffness \( K_r \) is pressed against the stator, the rotor deforms to balance the preload force \( F_{\text{preload}} \). With a rotor height \( w_f \) and vibration amplitude \( w_0 \), a pressure distri-
FIGURE 2. Nonlinearities of the speed/frequency response [Experiment]. The aggregated USM behavior for different experimental runs has the mean behavior in bold line plot and the standard deviation in area transparent plot. (a) Varying load torque. (b) Speed hysteresis. (c) Temperature Drift.

FIGURE 3. Illustration of the contact mechanics between the stator and rotor.

The function $P(\theta)$ is formed as in (2). The pressure is proportional to the intersection area with coefficients $K_r$. The contact area is bounded within region $[-\theta_b, \theta_b]$, and the pressure becomes zero outside this area. The contact angle $\theta_b$ is the outermost contact point and is defined as in (3). At a steady-state, the integral of the contact pressure across the stator surface is equal to the preload as in (4). If the vibration amplitude $w_0$ is relatively low and the applied preload is relatively high, the rotor is in full contact with the stator ($\theta_b = \pi/N$). The rotor is then maximally displaced to a minimum height $w_f \theta_{min}$ as in (5).

$$P(\theta) = \begin{cases} K_r(w_0 \cos(N\theta) - w_f), & \text{if } \theta \leq \theta_b \\ 0, & \text{if } \theta > \theta_b \end{cases}$$

$$\theta_b = \begin{cases} \frac{1}{N} \arccos\left(\frac{w_f}{w_0}\right), & \text{if } w_f \geq -w_0 \\ \frac{\pi}{N}, & \text{if } w_f < -w_0 \end{cases}$$

Using Kirchoff’s thin plate theory, the tangential velocity ($v_s$) of the stator surface is defined as (6), where $h$ and $r$ are the stator height and radius respectively. The velocity is maximum at the wave crest and zeroes at the nodal point. The rotor speed ($v_r$) is equal to the stator speed ($v_s$) at contact angle $\theta_a$ as in (7). At $\theta_a$, there exists a stick condition; otherwise, the rotor slips over the stator. The tangential traction forces at the contact surface can either be positive or negative due to the relative velocity difference as in (8). At steady state, the integral of tangential forces ($\mu P(\theta)$) balances the applied load torque ($\tau$) as in (9).

$$v_s(\theta) = \frac{\omega h}{r} - w_0 \cos(N\theta)$$

$$v_r[m/s] = \frac{2 \pi r}{60} v_r[rpm] = v_s(\theta_a)$$

$$\text{Sign}(v_\theta) = \begin{cases} +1, & \text{if } v_s(\theta) > v_r \\ -1, & \text{if } v_s(\theta) < v_r \\ 0, & \text{if } v_s(\theta) = v_r \end{cases}$$

$$\tau = N_r \int_{-\theta_b}^{\theta_b} \mu P(\theta) \text{Sign}(v_\theta) \, d\theta$$

$$= 2 N r \mu \left( \int_{0}^{\theta_a} P(\theta) \, d\theta - \int_{\theta_a}^{\theta_b} P(\theta) \, d\theta \right)$$

$$= 4 \mu K_r r (w_0 \sin(N\theta_a) - w_f N\theta_a) - \tau_{max}$$

where: $\tau_{max} = r \mu F_{preload}$

As discussed in [15], for a mode shape ($\Phi_\omega(\theta) = e^{j N \theta}$), the variational contact forces $F_N$ and $F_T$ can be calculated using (10) and (11) respectively. Furthermore, (1) can be decoupled into two perpendicular components by substituting the expressions for $w$ and $V$. By eliminating the shared term ($e^{j \omega t}$), equations (12) and (13) are obtained.
The phase difference $\Psi$ is eliminated by squaring and summing the two equations through the trigonometric identity $(\cos(x)^2 + \sin(x)^2 = 1)$ to obtain (14) with $w_0$, $F_N$, and $F_T$ as unknowns.

$$F_N = -N \int_{-\theta_b}^{\theta_b} \Phi_w(\theta) P(\theta) \, d\theta$$
$$= -K_r \left( w_0 \left( \frac{\sin(2N\theta_b)}{2} + N\theta_b \right) - 2w_f \sin(N\theta_b) \right)$$

(10)

$$F_T = -N \mu \frac{h}{r} \int_{-\theta_b}^{\theta_b} \frac{d\Phi_w(\theta)}{d\theta} \sin(\theta) P(\theta) \, d\theta$$
$$= -2\mu K_r \frac{h}{r} \left( w_0 \left( \frac{\sin(2N\theta_a)}{2} - \frac{\sin(2N\theta_b)}{4} + \frac{N\theta_a - N\theta_b}{2} \right) - w_f \left( 2N \sin(\theta_a) - \sin(N\theta_b) \right) \right)$$

(11)

$$L_{m}\omega^2 w_0 + \frac{1}{C_m} w_0 = \Theta V \cos(\Psi) + F_N$$

(12)

$$R_{m}\omega w_0 = \Theta V \sin(\Psi) + F_T$$

(13)

$$\left( (L_{m}\omega^2 + \frac{1}{C_m}) w_0 - F_N \right)^2 + \left( R_{m}\omega w_0 - F_T \right)^2 = \Theta^2 V^2$$

(14)

Using the set of developed equations, the steady state speed response can be calculated as shown in Fig. 4. The main objective is finding $w_0$ that minimizes the squared difference between LHS and RHS of (14). The amplitude $w_0$ is swept within its normal range $\left(10e^{-8}, 10e^{-5}\right)$. By conditioning over $w_0$ against $w_{f\text{min}}$, the contact condition is identified as full contact ($\theta_b = \pi/N$) or partial contact ($\theta_b < \pi/N$). In case of partial contact, (4) becomes a nonlinear equation of one unknown $\theta_b$ and can be solved with common nonlinear solvers. Similarly, after identifying $w_f$ and $\theta_b$, (9) can be solved for one unknown $\theta_a$ under a given torque $\tau$. Finally, the contact forces $F_N$ and $F_T$ are calculated and substituted back in (14). When (14) is minimized, the rotor speed is calculated using (6) knowing $w_0$, and $\theta_a$. Though the pull-out phenomenon can be reproduced using the developed model, the speed hysteresis cannot yet be simulated due to prior state dependence. Thus, the speed hysteresis was hardcoded as in (15). For a pull-in frequency $f_{in}$, the rotor speed is zero if the driving frequency ($f$) is less than $f_{in}$, and last speed ($v_{r_{i-1}}$) is zero. $f_{in}$ is approximated to be a multiple of resonance frequency ($f_r$). In addition to mathematical modeling of speed/frequency response under varying torque, shifts of the speed response associated with temperature changes were modeled empirically by adjusting the stator parameters according to the operating temperature. Equation (16) models the change in equivalent stiffness where the piezo ring softens as the temperature rises (capacitance increases). Depending on deviation of the current temperature ($T$) from the minimum temperature ($T_{min}$), the capacitance ($C_m$) deviates from the initial capacitance ($C_{m0}$) by factor ($\alpha_T$). Similarly, (17) models the change in equivalent damping where the damping reduces slightly causing an increase in peak speed as temperature increases. Parameters $\alpha_T$ and $\alpha_r$ were empirically estimated. Square rooting the temperature difference results in a saturation of the parameter change as the temperature excessively increases.

$$v_r = \begin{cases} 0, & \text{if } f < f_{in} \text{ & } v_{r_{i-1}} = 0 \\ v_r, & \text{otherwise} \end{cases}$$

(15)

where: $f_{in} = 1.03 f_r = 1.03 \sqrt{1/(L_m C_m)}/2\pi$

$$C_m = C_{m0} (1 + \alpha_T \sqrt{T - T_{min}})$$

(16)

$$R_m = R_{m0} (1 - \alpha_r \sqrt{T - T_{min}})$$

(17)

Using an Agilent 4294A impedance analyzer, the equivalent parameters of a preloaded stator were estimated. Table I lists main simulation parameters for modeling USM speed response. The model was simulated under a Python environment using SciPy optimization package. Using the proposed model, we could simulate the nonlinear speed response of USM under varying torque as shown in Fig. 5a. Also, the speed hysteresis and temperature dependence were simulated as shown in Fig. 5b and 5c. Even though exact matching is not necessary, the simulation model results resemble the experimental behavior to a great extent which would ease transitioning from simulation to experiment during controller design.

III. DRL USM SPEED CONTROLLER
A. CLASSICAL NEURAL NETWORK USM SPEED CONTROLLERS

To address the nonlinearity of USM, neural networks have been an attractive candidate due to their flexible mapping of complex functions. However, the developed NN controllers in the literature (i.e. [8], [9]) suffered from two main drawbacks; poor gradient estimation and on-policy training. To optimize the controller, gradient descent of the network weights ($W$) was used to minimize the loss ($L$) as in (18). The weights ($W$) are updated by stepping the weights with rate $\eta$ in the direction of the loss function gradient ($\frac{\partial L}{\partial W_{ij}}$). This term is deconstructed to the gradient of the loss with respect to the network output ($\frac{\partial L}{\partial V_s}$), and the gradient of the network output with respect to its weights ($\frac{\partial V_s}{\partial W_{ij}}$). The loss gradient $\frac{\partial L}{\partial U}$ is dependent on the USM dynamics which is often hard to model accurately for control purposes. Thus, [8], [9] has approximated the gradient based on expert knowledge of speed/frequency response as in (19). The gradient of speed with respect to frequency action ($\frac{\partial v_s}{\partial v_f}$) is assumed to be negative. However, this assumption doesn’t hold under pullout phenomenon and speed hysteresis, and thus the frequency range will be limited. Also, the estimated $\frac{\partial L}{\partial U}$ ignores the nonlinearity of speed/frequency response which would lead to slow and noisy convergence during training. Additionally, the control output gradient $\frac{\partial U}{\partial W_{ij}}$ is calculated relative to the current network weights ($W_{ij}$); as a result, the network is updated using most recent data (on-policy) causing low sample-efficiency and local optimality. To mitigate these problems, we propose Deep Reinforcement Learning (DRL).

$$\Delta W_{ij} = -\eta \frac{\partial L}{\partial W_{ij}} = -\eta \frac{\partial L}{\partial V_s} \frac{\partial V_s}{\partial W_{ij}}$$ \hspace{1cm} (18)

$$\frac{\partial L}{\partial U} = \varepsilon \frac{\partial v_s(U)}{\partial U} \approx \varepsilon \text{Sign} \left( \frac{v_s(U)}{\partial U} \right) \approx -\varepsilon$$ \hspace{1cm} (19)

B. DEER REINFORCEMENT LEARNING

Reinforcement learning (RL) is a branch of machine learning concerned with developing optimal control strategies by setting an agent (controller) to interact with an environment (plant) and improve its policy (control law) through iterative trial and error. RL can take optimal actions over an infinite planning horizon which makes it superior to model predictive control (MPC), as it can offer more optimal decisions with model-free offline calculations [17]. RL was proposed for speed control of electromagnetic motors and simulation results showed superior performance compared to PID controllers [18]. DRL was also applied to robotic hand manipulation [19], energy consumption optimization [20], molecular optimization [21], and hypersonic vehicles control [22].

An RL problem defines the optimal action ($a_t$) to take at a state ($s_t$) as the one that maximizes not only the immediate reward ($r_t$) but rather total expected reward ($R_t$). A common representation of an RL problem is as a Markov decision process (MDP) [23]. An MDP assumes that a future state ($s_{t+1}$) and a reward ($r_{t+1}$) are only dependent on the current state ($s_t$) and action ($a_t$), and none of the past states or actions. Also, the choice of the action ($a_t$) is only dependent on current state ($s_t$). By following the MDP, we obtain the following sequence ($s_t \rightarrow a_t(s_t) \rightarrow r_{t+1}(s_t, a_t) \rightarrow s_{t+1}(s_t, a_t) \rightarrow a_{t+1}(s_{t+1}) \rightarrow r_{t+2}(s_{t+1}, a_{t+1}) \rightarrow \ldots$). Under Markovian assumption, we define $Q$-value as the total expected future reward (finite horizon sum or discounted infinite-horizon sum) starting in the state ($s_t$) and taking action ($a_t$) as in (20). The $Q$-value is calculated as an expectation over policy ($\pi$), since the set of trajectory actions is a
function of policy \( (\pi_t = \pi(s_t)) \). Additionally, the current \( Q \)-value \( Q(s_t, a_t) \) can be represented as a Bellman optimality equation by summing the immediate reward and the expected future \( Q \)-value \( Q(s_{t+1}, a_{t+1}) \). Since the choice of action \( (a_t) \) depends on the current policy \( \pi_0(s_t) \), the learning objective becomes to find a policy \( \pi \) that maximizes the \( Q \)-function as in (21).

\[
Q^\pi(s_t, a_t) = E_\pi \left[ \sum_{t=0}^{\infty} \gamma^t r_{t+1} | s = s_t, a = a_t \right] \\
= E_\pi \left[ r_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1}) \right] \\
\pi^* = \text{argmax}_{\pi} \left[ E_\pi [Q(s, a)] \right]
\]

(20)

(21)

For continuous state and action spaces, a tabular \( Q \)-representation is no longer feasible. Thus, the function approximation of the \( Q \)-value or the policy becomes necessary. Due to their flexibility, deep neural networks can be a popular choice for approximating complex functions; resulting in deep reinforcement learning (DRL) algorithms. Using "Banach fixed point theorem", a tabular \( Q \)-representation learned by dynamic programming is guaranteed to find an optimal policy [24]. This is a result of the maximization over the Bellman operator being an \( \infty \)-norm contraction. Unfortunately, guarantees on policy optimality are lost under function approximation. Although approximating \( Q \)-function is a 2-norm contraction, maximization over an approximate \( Q \)-function is not a contraction of any type. Yet, various DRL algorithms were proposed to mitigate these theoretical limitations in practical settings. To find the optimal policy, there exist different model-free and model-based algorithms with different merits and drawbacks. In [25], a single-network adaptive critic design could reduce the required computation by using a single network. However, to solve the Hamilton–Jacobi–Bellman equation, some model of the system is required. Given the difficulty of developing accurate mathematical models of USMs, our focus is on model-free algorithms.

With growing interest in DRL, many model-free algorithms were developed including REINFORCE [26], deep Q-learning [27], proximal policy gradient (PPO) [28], Deep deterministic policy gradient (DDPG) [29], twin delayed DDPG (TD3) [30], and Soft Actor-Critic (SAC) [31]. To implement DRL for USM speed control, the SAC algorithm was chosen due to its stability, optimality, and robustness. SAC is also a better fit for the stochastic nature of USM where speed noise is a common challenge. Unlike DDPG, TD3, and PPO, SAC is more robust to parameters initialization and random seeds. Nevertheless, more algorithms are to be investigated in future research.

C. SOFT ACTOR-CRITIC ALGORITHM

SAC is an off-policy algorithm that uses two main networks one that learns the \( Q \)-function (critic), and another that learns the policy (actor). The critic can implicitly capture the system behavior offering a better estimation of the loss gradient \( \frac{\partial L}{\partial \theta} \) than that introduced in section III-A. As a result, better policy generalization and convergence are realized. The soft behavior of the SAC algorithm is due to its stochastic actor \( (\pi_\theta) \) that outputs the mean \( (\mu(s_t)) \) and the standard deviation \( (\sigma(s_t)) \) of a Gaussian distribution given a state \( (s_t) \). The action is then sampled randomly and squashed (bounded) using a tanh activation function as in (22). The probability of sampling action \( a_t \) at state \( s_t \) under policy \( \pi \) is \( \pi(a_t | s_t) \). The stochasticity of the action allows for better exploration of the action space at a given state, and thus, leads to better policies and more robustness to disturbances. To address the actor stochasticity, the \( Q \)-Bellman optimality function is modified by introducing the entropy term \( H(a_t | s_t) \) that encourages exploration of actions with low probability as in (23). The \( Q \)-value is a weighted function of the expected reward and the action probability under current policy. The parameter \( \alpha \) can balance the agent behavior between the exploration and the exploitation by weighting the entropy term \( (H) \). Finally, the optimal policy is obtained by maximizing the modified objective function as in (24). As the training progresses, the entropy is minimized by lowering the standard deviation, and the collected reward is maximized by taking nearly optimal deterministic actions.

\[
av_t \sim \pi(s_t) = \tanh \left( \mathcal{N}(\mu(s_t), \sigma(s_t)^2) \right)
\]

(22)

\[
Q^\pi(s_t, a_t) = E_\pi \left[ r_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1}) + \alpha H(a_t | s_t) \right]
\]

where: \( H(a_t | s_t) = -\log (\pi(a_t | s_t)) \)

\[
\pi^* = \text{argmax}_{\pi} \left[ E_\pi \left[ \sum_{t=0}^{\infty} \gamma^t (r_{t+1} + \alpha H(a_t | s_t)) \right] \right]
\]

(23)

(24)

D. CONTROL LYAPUNOV FUNCTION

To study the stability of continuous dynamical systems, Lyapunov stability theory is a common approach. For a dynamical system \( (\dot{x} = f(x, u)) \), the system is considered stable if there exist a Lyapunov function \( (V) \) such that (25) is satisfied. Starting at a state \( (x_0) \), the Lyapunov function of a stable system will asymptotically decrease towards a stable state due to the negative Lyapunov derivative \( (\dot{V}(x, u_t) < 0) \). To learn a stable controller, control Lyapunov function (CLF) has been proposed; especially for learning-based controllers [32]. To guarantee asymptotic convergence with rate \( (\lambda > 0) \), (26) should be satisfied. As a result, the Lyapunov function will decay at worst exponentially such that: \( V(x_t) = V(x_0)e^{-\lambda t} \). A higher \( \lambda \) encourages faster convergence but imposes more strict conditions to be satisfied by the controller.

\[
V(x_t) \geq 0 \ , \dot{V}(0) = 0 \ , \dot{V}(x_t, u_t) < 0
\]

(25)

\[
\dot{V}(x_t, u_t) + \lambda V(x_t) \leq 0
\]

(26)

For learning-based controllers with Lyapunov stability guarantees, [33]–[35] has been proposed. In [33], a Lyapunov actor critic (LAC) algorithm builds upon SAC algorithm by
learning a Lyapunov critic \( (L(s_t, a_t)) \) that behaves almost like a \( Q \)-function. To optimize the policy, the explicit difference of future and current critic values \( (L(s_{t+1}, a_{t+1}) - L(s_t, a_t)) \) is minimized. However, such explicit difference can destabilize the controller if the policy increased the second term \( (L(s_t, a_t)) \) instead of decreasing the first one \( (L(s_{t+1}, a_{t+1})) \). In [34], a model-free Lyapunov function \( (V(s_t)) \) was learned by minimizing some risk function that meets the criteria of a Lyapunov function. The learnt \( V(s_t) \) was then integrated within a PPO algorithm to optimize control policy. Despite the unrestricted topology of \( V(s_t) \), the learnt risk function is dependent on the current dynamics; slowing the learning process and decreasing sample efficiency due to on-policy learning. In [35], a model-based Lyapunov function is learnt by minimizing a control Lyapunov barrier function (CLBF). The learnt \( V(x) \) is then used for online optimization using a quadratic programming solvers.

For our purposes, a model-free controller is desired due to the model complexity of USM. Additionally, an off-policy learning algorithm will be more sample-efficient reducing the required training time. Unlike LAC, the desired algorithm should learn a more stable Lyapunov difference. Thus, we propose a modified SAC algorithm (CLF-SAC) which is model-free, off-policy, and stable by directly embedding a CLF within the reward function.

**E. PROPOSED CLF-SAC CONTROLLER FOR USM SPEED CONTROL**

For USMs, the stator vibrates as a second-order system; as a result, a frequency-controlled USM will have bounded speed output depending on the input driving frequency, and thus the system is always stable. However, another stability concern is the discontinuity of the speed response under frequency control. The controller should operate within the full frequency range to expand the motor’s output while being robust against common nonlinearities including pull-out and speed hysteresis. Given the great success of DRL over the past few years, it is only fair to apply it for overcoming current challenges with USM speed control. The speed/frequency response is dependent on operation conditions including preload, load torque, voltage amplitude, and temperature. Yet, we simplified the problem by focusing on a single control signal (Driving frequency) while fixing voltage amplitude and preload.

To implement a SAC agent (controller) for USM speed control, we need to define the input state, control action, and reward function. A simplified MPD of the proposed speed control is shown in Fig. 6. The state vector is chosen such that the Markovian property is satisfied, and the system is fully-observable for speed control. As discussed in section II, the rotor speed is dependent on the driving frequency \( (f_t) \) and the load torque \( (\tau_t) \). Additionally, current speed \( (v_{r_t}) \) can affect future speed due to speed hysteresis as in (15). Temperature variations can also cause changes in speed response which would cause suboptimal behavior if the temperature \( (T_t) \) is not included in the state definition. To learn an optimal policy for different target speeds, target speed \( (v_{t_{target}}) \) should be added to the state vector. Thus, the state vector should include the driving frequency \( (f_t) \), motor temperature \( (T_t) \), load torque \( (\tau_t) \), current speed \( (v_{r_t}) \), and target speed \( (v_{t_{target}}) \) as in (27). The control action \( (a_t) \) is simply the frequency update applied to the current frequency as in (28).

\[
\begin{align*}
    s_t &= [f_t, T_t, \tau_t, v_{r_t}, v_{t_{target}}] \\
    a_t &= \Delta f_t, \quad f_{t+1} = f_t + a_t
\end{align*}
\]

The most challenging part of RL is the proper definition of the reward function which would affect the learning process and the optimality of the agent. There are no yet concrete guidelines on reward function design, and there is no one universal function that works for all problems. To tackle our stability concern, a Lyapunov-based reward function is designed. In addition, the reward function can use the control Lyapunov function (CLF) to encourage asymptotic convergence. However, unlike LAC algorithm [33], we propose learning a \( Q \)-function directly from a CLF reward to overcome the instability of explicitly introducing CLF during policy optimization. Equation (29) is the proposed reward function to satisfy a CLF condition and regulate controller effort. The time derivative of Lyapunov function \( (\dot{V}(s_t, a_t)) \) is a function of the current state \((s_t)\) and system dynamics \((a_t)\). From sampled experiences of steady-state transitions \((s_{t+1}, a_t, s_{t+1})\), \( \dot{V}(s_t, a_t) \) can be empirically estimated from the difference of Lyapunov values \( (\dot{V}(s_{t+1}) - \dot{V}(s_t)) \). To satisfy the CLF condition in (26), the reward function penalizes any violations of CLF (positive values). A major design choice is the form of the Lyapunov function \((\dot{V}(s_t))\) that satisfies (25). For the speed control objective, the proposed Lyapunov function is the square root of absolute speed error normalized by a gain \( K_V \) as in (30). The form is chosen among others based on simulation results as discussed in section V. If the controller satisfies the CLF in discrete settings, the Lyapunov function will decay according to (31). Additionally, the reward includes the weighted \((w_a)\) absolute action to penalize extreme actions and minimize controller effort. The choice of \( w_a \) depends on the application requirements and the target hardware capabilities.

\[
\begin{align*}
    r_{t+1} &= -\max \left(0, \dot{V}(s_t, a_t) + \lambda V(s_{t+1}) - w_a |a_t| \right) \\
    \text{where: } \dot{V}(s_t, a_t) &= E_a [V(s_{t+1}) - V(s_t)] \\
    V(s_t) &= K_V \sqrt{|v_{r_t} - v_{t_{target}}|} \\
    V(s_{t+1}) &= \frac{V(s_t)}{1 + \lambda}
\end{align*}
\]

**F. SAC TRAINING PROCEDURE**

To train the SAC agent, algorithm 1 was implemented in a Python environment using PyTorch neural networks library. As an actor-critic algorithm, SAC uses two core NNs; an actor (policy \( \pi_a \)) network that outputs stochastic actions, and
a critic (Q-value \( Q_{\phi} \)) network that evaluates actions quality at a given state. A minor modification to the original SAC algorithm is replacing the two single-output critic networks with one critic with multiple outputs. Using a single network reduces the computations required during training for regulating the Q-value.

Starting with random network initializations, the agent interacts with the environment to collect experiences and append them to an experience buffer \( (B) \). The training runs for a total of \( N \) episodes; each lasts for \( l \) steps. Periodically, a batch of \( m \) experiences is used to optimize the two networks using gradient descent with rates \( l_{r_{2}} \) and \( l_{r_{a}} \). First, the critic is updated by minimizing the quadratic loss \( (L_Q) \) between the expected Q-value of each of the \( n \) output Q-values and a target value \( (y_t) \) as in (32). The target \( y_t \) is approximated from the Bellman form in (23). Instead of having the Q-network regressing towards a moving objective, a target Q-network \( (Q_{\hat{\phi}_{targ}}) \) is used to estimate a stable target Q-value \( (y_t) \) to stabilize the convergence process. The target network is a delayed copy of the Q-network through a Polyak averaging parameterized with update rate \( \varepsilon \). To evaluate the target \( y_t \), a future action \( (a_{t+1}) \) is evaluated from current policy \( (\hat{a}_{t+1} = \pi_{\theta}(s_{t+1})) \). Following the critic update, the actor is updated by minimizing the policy loss \( (L_\pi) \) defined in (35). For a given state \( s_t \), the policy loss aims to maximize the expected Q-value from taking action \( \hat{a}_t \) under current policy \( (\hat{a}_t = \pi_{\theta}(s_t)) \) while encouraging exploration of low probability actions (High entropy). A conservative estimation of critic output is obtained by taking the minimum over the \( n \) network outputs (i.e (33), (35)).

\[
L_{Q_{\phi}} = \left( Q_{\phi_{Q}}(s_t, a_t) - y_t \right)^2 \tag{32}
\]

\[
y_t = r_{t+1} + \gamma \left( \min_{i=1..5} \left[ Q_{\phi_{targ}}(s_{t+1}, \hat{a}_{t+1}) \right] - \alpha \log \left( \pi_{\theta}(\hat{a}_{t+1}|s_{t+1}) \right) \right) \tag{33}
\]

\[
\phi_{Q_{targ}} = (1 - \varepsilon)\phi_{Q_{targ}} + \varepsilon\phi_{Q} \tag{34}
\]

\[
L_{\pi} = -\left( \min_{i=1..5} \left[ Q_{\phi_{\pi}}(s_t, \hat{a}_t) \right] - \alpha \log \left( \pi_{\theta}(\hat{a}|s_t) \right) \right) \tag{35}
\]

Similar to other DRL algorithms, a few techniques are implemented to further stabilize convergence and encourage generalization under continuous state/action spaces. The experience buffer \( (B) \) can store all interactions between the agent and the environment. The better the representation of the collected experiences to the true system behavior, the more optimal the trained policy would be. To avoid local optimization, a batch of experiences is sampled randomly from the experience buffer \( (B) \) at each optimization step. Such randomization de-correlates the training samples and results into a more general policy. Generalization can be further improved by exploring different states of the environment through initial state randomization. At the start of each episode, a uniformly-random initial state \((s_0 = [f_0, T_0, \tau_0, v_{r_{0}}, v_{targ_{0}}])\) is sampled from their respective state space. Finally, to ensure faster convergence of the learning process, the input state vector and the output reward need to be normalized to have all of the entries on the same scale.

**Algorithm 1 SAC for USM Speed Control**

**Input:** Policy network \((\theta_{\pi})\), Q-value network \((\phi_{Q})\), target Q-value network \((\phi_{Q_{targ}})\), and an experience buffer \((B)\)

**Initialization:**

1. Initialize networks \((\theta_{\pi}, \phi_{Q})\) randomly.
2. Copy weights to the target Q-network \((\phi_{Q_{targ}})\).
3. Train for \( N \) episodes, each with \( l \) steps.

**for** \( ep = 1 \) to \( N \) **do**

**for** \( t = 0 \) to \( l \) **do**

1. Sample an uniformly-random initial state \((s_0)\).
2. Sample action \((a_t \sim \pi_{\theta}(s_t))\)
3. Transit to new state \((s_{t+1})\)
4. Calculate reward \((r_{t+1})\) from (29).
5. Add the experience \((s_t, a_t, s_{t+1}, r_{t+1})\) to buffer \( B \).

**end for**

**if** UpdateNetworks **then**

**for** \( i = 0 \) to \( k \) **do**

1. Sample an \( m \)-sized batch of experiences from \( B \)
2. Estimate Q-targets from (33)
3. Compute Q-loss from (32) and update \( \phi_{Q} \).
4. Compute policy-loss from (35) and update \( \theta_{\pi} \).
5. Update target Q-network \( \phi_{Q_{targ}} \) from (34).

**end for**

**end if**

**return** Trained policy network \((\theta_{\pi})\)

Since the choice of the hyperparameters can significantly affect the algorithm behavior, they should be chosen carefully to meet the desired objective. Parameter \( \gamma \) affects the long-term optimality of the agent. A discount factor of 1 cares about all future rewards equally, whereas a discount factor of 0 cares only about immediate rewards. To control the exploration/exploitation tradeoff, the parameter \( \alpha \) is selected such that lower \( \alpha \) causes more exploitation, and vice versa. With higher \( \alpha \), the agent collects a higher reward by exploring low probability actions as in (35). The choice of the
learning rates ($l_{r_\tau}, l_{r\xi}$) controls the size of gradient steps for network update and can affect learning speed and stability. The parameter $\varepsilon$ controls the rate at which the target network gets updated. A small $\varepsilon$ results in slower but stable learning; on the contrary, a larger $\varepsilon$ allows faster learning but might cause instability. Also, a large batch size ($m$) results in stable gradient steps but is computationally expensive. Finally, a proper choice of CLF coefficient $\lambda$ can ensure a stable and fast response. However, too high $\lambda$ would encourage larger actions which might cause instability. Yet, too low $\lambda$ might slow the system response. All hyper parameters were fixed during training; however, varying the hyperparameters online can realize more optimal policies and faster convergence. For example, the entropy weight $\alpha$ can be set high at first to encourage exploration and then decayed gradually to exploit the current optimal policy at later phases of training. In future work, online hyperparameter tuning is to be studied.

### IV. RESULTS

#### A. AGENT TRAINING

The SAC agent proposed in Section III was trained and evaluated both in simulation and experiment using design parameters in Table 2. A common challenge in machine learning is parameter tuning; thus, these parameters might require further tuning depending on the agent objective (i.e., faster response, higher accuracy, shorter training). The network sizes can be further optimized to suit processors with low computation power. However, the controller (actor) can be designed to have a smaller size compared to the critic that captures all system complexity. For USM speed control, the planning horizon is relatively short, and thus a small discount factor ($\gamma = 0.5$) was sufficient. To further stabilize the learning process, and the motor behavior during the experiment, the frequency action was bounded between $\pm 2$ kHz.

For proper training of the neural network, the inputs and outputs are normalized. The Lyapunov normalizing gain ($K_V$) was chosen to scale $V(s_t)$ between $[0,10]$ as the speed error varied between $[0,300]$ rpm. To normalize the input state, a min-max normalization was applied to the state vector to rescale all features between $[-1,1]$ as in (36). the ranges of input variables were set as $[f | 39 \rightarrow 45 | kHz, T | 20 \rightarrow 60 | ^\circ C, \tau | [0 \rightarrow 1] | N.m, v | [0 \rightarrow 300] | rpm]$. Within this frequency range $[39-45]$ kHz, the motor can operate between a speed of $0$ rpm and a peak speed around $300$ rpm with torque varying from $0$ to $1$ N.m. The temperature range was set between the ambient temperature ($20^\circ C$) and the maximum recommended operation temperature ($60^\circ C$). During current experiments under no-load and a relatively low voltage amplitude, the maximum recorded temperature was $40^\circ C$. Increasing the load torque, voltage amplitude, and operation period can further increase the operating temperature in future studies. Yet, these ranges do not have to match exactly the actual operation ranges since the main objective of such scaling is proper convergence of the neural network.

\[
\dot{x} = 2\frac{x - x_{\min}}{x_{\max} - x_{\min}} - 1 \tag{36}
\]

First, the controller was trained using the simulation model developed in Section II. The agent was trained for 4000 episodes; each of which is 10 steps. At the start of each episode, the load torque ($\tau_n$), target speed ($V_{\text{target}}$), and motor temperature ($T_0$) were sampled uniformly from their respective ranges and kept constant through the rest of the episode. Fig. 7 shows the learning curve in both simulation and experiment. The episodic reward fluctuates severely due to torque variations where the motor cannot realize high target speeds under relatively high torques. The average reward (over 10 episodes window) is plotted to show a more stable learning behavior. As the network was initialized with random weights, the first few episodes had low rewards. As the agent learned the proper policy, the reward increased to a nearly stable value around episode 500. Further training was carried out for further optimization given the low training cost in simulation (under 10 minutes). The expected reward is the reward expected by the $Q$-network given the initial state. Initially, there was quite a mismatch between true reward and expected reward; however, they converge to almost the same values at later episodes. This confirms that the $Q$-network could accurately predict the system behavior, and thus, a successful training. Additionally, the averaged absolute speed error ($|v_{r_\tau} - v_{\text{target}}|$) had reduced as training progressed. The target speed (black) varied between $[0-300]$ rpm; however, the motor speed (red) cannot realize relatively high speeds under relatively high torques. Thus, the average speed error could not converge to zero.

Following the success of the agent in simulation, the SAC agent was implemented experimentally. The agent networks weren’t randomly initialized as in simulation, but using the weights of the trained simulation agent. Such initialization can allow faster learning and more stable behavior initially which is safer for the hardware than complete randomness. The agent was trained for 2000 episodes and was able to...

---

**TABLE 2. SAC Agent Design Parameters**

| Parameter                  | Value  |
|----------------------------|--------|
| Actor network size         | 64x64  |
| Critic network size        | 64x64x64x64 |
| Activation function        | Relu   |
| Optimizer                  | Adam   |
| Actor learning rate $l_{r_\tau}$ | 2.5e-3 |
| Critic learning rate $l_{r\xi}$ | 5e-3   |
| Buffer size                | 1e6    |
| Batch size $m$             | 64     |
| Discount factor $\gamma$  | 0.5    |
| Entropy weight $\alpha$    | 0.5    |
| Update rate $\varepsilon$ | 0.01   |
| Action Weight $w_a$        | 0.1    |
| CLF Weight $\lambda$      | 2      |
| Normalizing Gain $K_V$     | $10/\sqrt{300}$ |
Simulation | Experiment
---|---
\begin{figure}
\begin{subfigure}{0.45\textwidth}
\centering
\includegraphics[width=\textwidth]{simulation}
\caption{Simulation}
\end{subfigure}
\begin{subfigure}{0.45\textwidth}
\centering
\includegraphics[width=\textwidth]{experiment}
\caption{Experiment}
\end{subfigure}
\end{figure}

immediately realize a stable performance that improved with further training. It is noted that the actual speed was on average higher in simulation than in experiment due to modeling inaccuracies. It is worth noticing that using the current experimental setup, one step of the agent is approximately 20 ms. This is mainly due to communication delay through the RS232 and the GPIB connections, and the response time of USM under high inertia.

### B. AGENT EVALUATION

After training the agent successfully, a set of experiments were designed to verify the agent meets the desired control objectives. First, the controller stability under a wide frequency range for a wide range of load torques was verified. Additionally, the controller should be optimal by taking minimum frequency update steps to reach the target state. Finally, the controller should be robust to USM nonlinearities (i.e., frequency dead zone, pull-out, speed hysteresis). During the evaluation phase, the parameters of the networks were fixed, and no online optimization was carried out. As a result, an evaluation step was much faster than a training step. Also, to increase the expected reward during evaluation, the optimal action is deterministic using only the mean output ($\mu(s_t)$) of the stochastic actor. The horizontal axis is labeled as a step rather than time. Depending on the hardware capabilities, application requirements, and system response, one step of the SAC agent can vary from a fraction of a second to a few seconds. The presented tests were conducted in both simulation and experiment settings, and the results showed high agreement in both cases. However, only experimental results are presented to limit the number of figures and to prove the practicality of the proposed controller.

First, the controller response is evaluated when initialized at frequency dead zone to verify its optimality and far-sightedness as shown in Fig. 8. For different constant target speeds [100, 200, 300] rpm, the motor was initialized at a driving frequency of 39 kHz where the motor cannot operate (dead zone). The far-sighted agent commanded a frequency action to escape the pull-out region and resume motor operation by increasing the driving frequency to reach the pull-in region, and then the frequency was reduced to realize the target speed. The number of steps taken to reach a target speed was minimum which supports the controller’s optimality. Such trajectory planning is challenging to implement using fuzzy rules or supervised learning techniques. It is possible to realize long-sighted planning with model predictive controllers at the cost of online model identification and optimization. Also, unlike conventional controllers, the controller can work safely close to the resonance region, and thus higher target speed (i.e 300 rpm) can be commanded without losing stability. If the target speed is too high, the controller would get as close as possible without losing stability. Yet, there is also a non-zero steady-state error for feasible target speeds which is under current study.

Beyond the rated speed of 150 rpm of the USR60, the target speed is sinusoidally varied between [0-300] rpm for different constant load torques as in Fig. 9. The sinusoidal signal lasted for 5 periods; 50 steps each. For the no-load case (0 N.m load torque), there was nearly perfect tracking. As the load torque increased, proper tracking was possible at
low target speeds; however, higher target speeds couldn’t be realized. Yet, the system remained stable.

Typically, USM is operated under varying load torque. Thus, the controller was commanded to track constant target speeds under sinusoidally varying load torque between [0-1] N.m. The rated torque of USR60 is 0.5 N.m; as a result, at extreme load torque, the motor would stall. Even when the load torque is reduced, conventional controllers would fail to resume rotation due to USM nonlinearity. In Fig. 10, our proposed controller successfully tracked target speeds at relatively low load torques. However, the speed error increased as the load torque increased due to the inverse relationship between rotor speed and load torque. Despite the controller’s stability, the behavior was inconsistent. For example, under a target speed of 200 rpm, the USM changed behavior at later cycles; even when nearly the same frequency was commanded. This phenomenon is under current study.

Under continuous operation, the accumulated power losses cause a temperature rise of USM and consequently a drift of the speed/frequency response as shown in Fig. 2c. Under no-load operation, the agent can successfully track target speed under continuous operation lasting for 100 seconds as shown in Fig. 11. Higher load torques or driving voltage amplitudes would result in excessive heating and might damage the USM. Fig. 11a shows proper tracking of different target speeds by adjusting the driving frequency as in Fig. 11b under temperature rise shown in Fig. 11c. Yet, similar to Fig. 8, a non-zero steady-state error was produced by the controller. Higher speeds caused more losses, and thus, higher temperature rise. In turn, higher temperatures caused a further shift of the speed response and required a larger adjustment of the driving frequency. The temperature would continuously increase until the generated heat by the motor is balanced by the lost heat due to conduction/convection heat transfer.

V. DISCUSSION
A. COMPARATIVE RESULTS
A typical benchmark is to compare our controller to linear controllers. Though it might be an unfair comparison, it clearly shows the advantage of using our proposed non-linear controller. Our controller is compared to three other controllers: proportional integral derivative (PID), linear quadratic regulator (LQR), and model-free adaptive control (MFAC [36]). The LQR was not obtained through the knowledge of system dynamics ($\dot{X} = AX + BU$), but rather an optimization of a full state feedback controller of scaled state ($\hat{s}_i$). Under no-load settings, a sinusoidal target speed was commanded limited to 200 rpm peak speed to avoid the pull-out phenomenon. Also, the driving frequency was initialized at 45 kHz to escape the frequency dead zone. These two challenges cannot be handled by linear controllers. To optimize the different controllers gains, particle swarm optimization [37] was utilized using Pyswarms Python package. The cost function was the mean absolute speed error. The optimization ranges were set based on expert knowledge, and trial and error in simulation settings. Following setting up proper ranges, the controllers’ gains were optimized experimentally using 5 particles for 10 iterations. The final optimized gains are listed in Table 3. As shown in Fig. 12, the proposed SAC controller could outperform other controllers in tracking desired target speed and minimizing speed error. Other linear controllers failed to track a fast-changing sinusoidal at low target speeds due to their relatively low gains. On the other hand, higher gains might cause instability due to USM nonlinearity.
and fell into the frequency dead zone. The SAC controller outperformed PID which started losing stability after a few steps. For a target speed of 300 rpm, all linear controllers failed to track the target speed, while SAC successfully did. For all target speeds, the SAC agent was the most optimal; converging in under 5 steps. Unlike linear controllers (i.e. PID) that converge to a zero steady-state error (i.e. 100 rpm), the main challenge with the current SAC architecture is the non-zero steady-state error. An ensemble of these two controllers to mitigate these challenges is under the current study.

### B. CONTROLLER STABILITY

Two important aspects of SAC stability is the stability of the learning process and the stability of the learned policy (controller). One common issue of many DRL algorithms is their sensitivity to parameter choices and random seeds. Yet, the SAC algorithm is believed to be more robust to these issues than other algorithms. To validate this behavior under the USM speed control problem, the agent was trained multiple times for different random seeds under simulation. Furthermore, different Lyapunov functions were studied to show the optimality of the proposed Lyapunov function. Four different functions were proposed including square, absolute, root, and log error as in Fig. 14a. All functions were rescaled between [0,300] rpm (10 periods, 100 episodes each). For each random seed, a different initialization of the networks was obtained. In Fig. 14b, the averaged speed error through the training phase is shown. Speed error is a more reliable metric than episodic reward due to different function behavior. As the training progressed, the speed error generally decreased for all functions. However, the log error function (green) maintained higher peaks for more episodes showing slower convergence properties. This can be explained by the low slope of the log function at high values which reduces the gradient steps. In Fig. 14c, a close-up of the learning curve at final episodes shows that root error and log error functions had the lowest speed errors. This is a direct result

![FIGURE 11. Evaluating SAC controller constant speed tracking under temperature rise due to continuous operation. (a) Constant speed tracking. (b) Driving frequency drift. (c) Temperature rise.](image)

![TABLE 3. Comparative Study: Controller Optimized Gains](table)

| Controller | Optimized Gains | Cost |
|------------|-----------------|------|
| LQR        | \(K = [0, 0, 0, -0.65, 0.65]\) | 7.61 |
| PID        | \([K_p, K_i, K_d] = [1.06, 2.44, 0] \times 10^{-3}\) | 14.4 |
| MFAC       | \([\rho, \mu, \lambda, \eta] = [0.29, 4.9, 65, 0.18]\) | 8.3 |

![FIGURE 12. Comparison of sinusoidal speed tracking results of the proposed SAC controller against LQR, PID, and MFAC controllers.](image)
VI. CONCLUSION

In this paper, we proposed deep reinforcement control as a robust, stable, long-sighted, and optimal controller for USM speed control. As verified experimentally, the developed controller can extend the operation range of USR60 by overcoming the USM nonlinearities. Despite optimal speed tracking, there still exists some non-zero steady-state error even during simulation. Additionally, the current implementation is sensitive to parameters variations that are not accounted for in the input state definition (i.e., friction coefficient or equivalent mass and stiffness). As a result, applying the same trained agent to a different motor can yield suboptimal performance.

Future work would focus on reducing steady-state error and enhancing robustness to parameters variations. One approach is to integrate DRL with other controllers (i.e., PID) to improve robustness and reduce tracking error. To extend DRL beyond USM speed control, the agent input state would be expanded to fit changes in the operating conditions including preload, voltage amplitude, and phase difference. For improved controllability and performance, the SAC agent can output multiple actions including driving frequency, phase difference, voltage amplitude, and preload.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3170995, IEEE Access
The agent reward function can be further modified to fit new objectives including torque control, bidirectional speed control, position control, compliance control, or efficiency optimization. Finally, further study of the agent architecture is to be conducted to improve the sample efficiency and establish some concrete theoretical stability guarantees.

ACKNOWLEDGMENT

The authors would like to thank SHINSEI Corp. for providing the USR60 motors used in current research.

REFERENCES

[1] S. Ueha and Y. Tomikawa, “General outline of ultrasonic motors,” in Ultrasonic Motors: Theory and Applications, Oxford, USA: Clarendon Press, 1993, pp. 1-7.
[2] K. Uchino, “Piezoelectric ultrasonic motors: overview,” Smart Mater. Struct., vol. 7, no. 3, pp. 273-285, 1998.
[3] T. Morita, “Miniature piezoelectric motors,” Sens. Actuator A Phys., vol. 103, no. 3, pp. 291-300, 2003.
[4] Y. Y. Liu, R. Ozaki, and T. Morita, “Investigation of nonlinearity in piezoelectric transducers,” Sens. Actuator A Phys., vol. 227, pp. 31-38, 2015.
[5] J. Wallaschek, “Contact mechanics of piezoelectric ultrasonic motors,” Smart Mater. Struct., vol. 7, pp. 369-381, 1998.
[6] Y. Izuino, T. Izumi, H. Yasutsune, E. Hiraki, and M. Nakaoka, “Speed tracking servo control system incorporating traveling-wave-type ultrasonic motor and feasible evaluations,” IEEE Trans. on Ind. Appl., vol. 34, no. 1, pp. 126-132, 1998.
[7] W. Wei, C. Yi, and H. Ming, “An Overview of the Control Methods for Ultrasonic Motor,” in Proc. IEEE 3rd ICTNFC, 2019, pp. 1545-1548.
[8] F. Lin, R. Wai, and R. Duan, “Neural-network controller for parallel-resonant ultrasonic motor drive,” IEEE Trans. Control Syst. Technol., vol. 7, no. 4, pp. 494-501, 1999.
[9] T. Senjyu, H. Miyazato, S. Yokoda, and K. Uezato, “Speed control of ultrasonic motors using neural network,” IEEE Trans. on Power Electr., vol. 13, no. 3, pp. 381-387, 1998.
[10] T. Jingzhuo, H. Wenwen, and Z. Ying, “T-S Fuzzy Control of Travelling-Wave Ultrasonic Motor,” J. Control Autom. Electr. Syst., vol. 31, no. 2, pp. 319-328, 2020.
[11] K. T. Chau, S. W. Chung, and C. C. Chan, “Neuro-fuzzy speed tracking control of travelling-wave ultrasonic motor drives using direct pulse width modulation,” IEEE Trans. on Ind. Appl., vol. 39, no. 4, pp. 1061-1069, 2003.
[12] S. Jingzhuo and W. Huang, “Predictive Iterative Learning Speed Control with On-Line Identification for Ultrasonic Motor,” IEEE Access, vol. 8, pp. 78202-78212, 2020.
[13] S. Jingzhuo, W. Huang, and Z. Liuqing, “Improved Indirect Iterative Learning MIT Control Method for Ultrasonic Motor,”IEEE Access, vol. 9, pp. 100308-100318, 2021.
[14] W. Ren, L. Yang, M. Chengcheng, X. Li, and J. Zhang, “Output performance simulation and contact analysis of traveling wave rotary ultrasonic motor based on ADINA,” Comput. Struct. vol. 216, pp. 15-25, 2019.
[15] N. W. Hagoed and A. J. McFarland, “Modeling of a piezoelectric rotary ultrasonic motor,” IEEE Trans. Ultrason., Ferroelectr. Freq. Contr., vol. 42, no. 2, pp. 210-224, 1995.
[16] M. Aoyagi, Y. Tomikawa, and T. Takano, “Simplified equivalent circuit of an ultrasonic motor and its applications,” Ultrasonics, vol. 34, no. 2, pp. 275-278, 1996.
[17] Y. Lin, J. McPhee, and N. L. Azad, “Comparison of Deep Reinforcement Learning and Model Predictive Control for Adaptive Cruise Control,” IEEE Trans. Intell. Veh., vol. 6, no. 2, pp. 221-231, 2021.
[18] Z. Song, J. Yang, X. Mei, T. Tao, and M. Xu, “Deep reinforcement learning for permanent magnet synchronous motor speed control systems,” Neural. Comput. Appl., vol. 33, no. 10, pp. 5409-5418, 2021.
[19] M. Saeed, M. Nagdi, B. Rosman and H. S. M. Ali, “Deep Reinforcement Learning for Robotic Hand Manipulation,” in 2020 ICCEEE, 2021, pp. 1-5.
[20] G. Qi, Y. Luo, G. Wu, K. Boriboonsomsin, and M. Barth, “Deep reinforcement learning enabled self-learning control for energy efficient driving,” Transp. Res. Part C Emerg. Technol., vol. 99, pp. 67-81, 2019.
[21] Z. Zhou, S. Kearnes, L. Li, R. N. Zare, and P. Riley, “Optimization of Molecules via Deep Reinforcement Learning,” Sci. Rep., vol. 9, no. 1, Art. no. 1, 2020.
[22] B. Xu, Y. Xiao, and H. Lei, “An Adaptive Critic Design-Based Fuzzy Neural Controller for Hypersonic Vehicles: Predefined Behavioral Non-affine Control,” IEEE/ASME Trans. on Mechatron., vol. 24, no. 4, pp. 1871–1881, 2019.
[23] R. S. Sutton and A. G. Barto, “Finite Markov Decision Processes,” in Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: MIT Press, 2018, pp. 1-68.
[24] E. Denardo, “Contraction mappings in the theory underlying dynamic programming,” SIAM Rev., vol. 9, no. 2, pp. 165-177, 1967.
[25] X. Bu, “An improvement of single-network adaptive critic design for nonlinear systems with asymmetry constraints,” J. Franklin Inst., vol. 356, no. 16, pp. 9646-9664, 2019.
[26] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” Mach. Learn., vol. 8, no. 3, pp. 229–256, 1992.
[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing Atari with Deep Reinforcement Learning,” arXiv:1312.5002, 2013. [Online]. Available: http://arxiv.org/abs/1312.5002
[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms,” arXiv:1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347
[29] T. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” arXiv:1509.02971, 2015. [Online]. Available: http://arxiv.org/abs/1509.02971
[30] S. Fujimoto, H. van Hoof, and D. Meger, “Addressing Function Approximation Error in Actor-Critic Methods,” arXiv:1802.09477, 2018. [Online]. Available: http://arxiv.org/abs/1802.09477
[31] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor,” arXiv:1801.01290, 2018. [Online]. Available: http://arxiv.org/abs/1801.01290

FIGURE 15. Monte-Carlo evaluation of the stability of the learned policy under no-load [Simulation]. The trajectory is presented as a line plot, and the final state is shown with a star marker (a) Tracking three constant target speeds [100,200,300] rpm starting at different random driving frequencies. (b) Tracking different random target speeds starting at the two extreme frequencies [39,45] kHz.
ABDULLAH MUSTAFA received his B. Eng. degree in mechanical engineering with a concentration in mechatronics from the American University in Cairo in 2016. He joined the University of Tokyo as a research student in 2017 and completed his master’s degree in 2020. He is currently pursuing his Ph.D. degree at the graduate school of frontier sciences. His research interests include ultrasonic motors’ design and control.

TATSUKI SASAMURA received his B. Eng. degree in precision engineering from the University of Tokyo in 2020. He is currently pursuing his master’s degree at the Graduate School of Frontier Sciences, the University of Tokyo. His research interests include modeling, design, and control of ultrasonic motors.

TAKESHI MORITA received B. Eng., M. Eng. and Dr. Eng. degrees in precision machinery engineering from the University of Tokyo in 1994, 1996, and 1999, respectively. After being a postdoctoral researcher at RIKEN (the Institute of Physical and Chemical Research) and at EPFL (Swiss Federal Institute of Technology), he became a research associate at Tohoku University in 2002. He obtained a position at the University of Tokyo as an associate professor in 2005 and has been a full professor since 2018. His research interests include piezoelectric actuators and sensors, their fabrication processes, and control systems.

***

[32] A. Anand, K. Seel, V. Gjærum, A. Håkansson, H. Robinson, and A. Saad, “Safe Learning for Control using Control Lyapunov Functions and Control Barrier Functions: A Review,” Procedia Computer Science, vol. 192, pp. 3987–3997, 2021.

[33] M. Han, L. Zhang, J. Wang, and W. Pan, “Actor-Critic Reinforcement Learning for Control with Stability Guarantee,” IEEE Robot. Autom. Lett., vol. 5, no. 4, pp. 6217–6224, 2020.

[34] Y. Chang and S. Gao, “Stabilizing Neural Control Using Self-Learned Almost Lyapunov Critics,” Proc. of ICRA 21, 2021. [Online]. Available: http://arxiv.org/abs/2107.04989

[35] C. Dawson, Z. Qin, S. Gao, and C. Fan, “Safe Nonlinear Control Using Robust Neural Lyapunov-Barrier Functions,” Proc. of CoRL 21, 2021. [Online]. Available: http://arxiv.org/abs/2109.06697

[36] S. Di and H. Li, “Model-free adaptive speed control on travelling wave ultrasonic motor,” Journal of Electr. Eng., vol. 69, pp.14-23, 2018.

[37] J. Kennedy and R. Eberhart, “Particle swarm optimization,” Proc. of ICNN 95, 1995, vol. 4, pp. 1942-1948.