International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 02 | Feb - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Adaptive Tuning of Virtual Synchronous Machine Parameters (J and D) Using Deep Reinforcement Learning

1.Mr.Ved Prakash,

Department of Electrical Engineering (Power System)

(UNS Institute of Engineering and Technology, Jaunpur, Uttar Pradesh, India)

Veer Bahadur Singh Purvanchal University Jaunpur Uttar Pradesh

2.Dr. Saurabh V. Kumar

ASSOCIATE PROFESSOR

Department of Electrical Engineering (Power System)

(UNS Institute of Engineering and Technology, Jaunpur, Uttar Pradesh, India)

Veer Bahadur Singh Purvanchal University Jaunpur Uttar Pradesh

Abstract*—Virtual Synchronous Machines (VSMs) emulate the inertial and damping behavior of conventional synchronous generators in inverter-based power systems. Conventional VSM implementations use fixed values of inertia constant (J) and damping coefficient (D), which lead to sub-optimal performance under dynamic grid conditions. This paper proposes an intelligent adaptive tuning framework for J and D using Deep Reinforcement Learning (DRL), employing Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) algorithms, validated via a MATLAB/Simulink–Python co-simulation environment. Simulation results across five disturbance scenarios demonstrate that the SAC-based controller achieves up to 48.6% reduction in maximum frequency deviation, 41.2% improvement in settling time, and 46.8% reduction in ROCOF compared to fixed-parameter VSM. The complete MATLAB and Python implementation details and simulation graphs are presented, confirming the viability for next-generation grid-forming inverter control.

Index Terms—Virtual Synchronous Machine, Deep Reinforcement Learning, PPO, SAC, Adaptive Control, Inertia Emulation, Frequency Regulation, MATLAB/Simulink, Python, Grid-Forming Inverter.

I. INTRODUCTION

The rapid integration of renewable energy sources into modern power grids has significantly reduced physical rotating inertia from synchronous generators. This decline poses serious challenges to frequency stability [1]. Virtual Synchronous Machine (VSM) technology restores synthetic inertia by emulating the classical swing equation in inverter-based generation [2].

Existing VSMs use fixed virtual inertia J and damping D values, tuned offline for nominal conditions. Under transient events, fixed parameters yield sub-optimal dynamic performance. This paper addresses this gap using Deep Reinforcement Learning (DRL) with PPO [3] and SAC [4] algorithms, implemented in a MATLAB/Simulink–Python co-simulation environment.

II. VSM MATHEMATICAL MODEL

The VSM swing equation is:

$$J \cdot d\omega/dt = Pm - Pe - D \cdot (\omega - \omega ref) \quad ...(1)$$

The active power output is controlled through the virtual rotor angle δ, where the inverter reference current is synthesized from the VSM internal EMF. The complete state-space model for the grid-connected VSM is:

$$d\delta/dt = \omega - \omega s; \quad dE/dt = (Eref - |V| + kq \cdot (Qref - Q)) / \tau q \quad ...(2)$$

III. MATLAB/SIMULINK MODEL

A. Simulink Architecture

The MATLAB/Simulink model implements the VSM as a discrete-time control system with sample time Ts = 10 μs for the inner current loop and 10 ms for the DRL interface. The testbed consists of:

- 1 MW Grid-Connected Inverter with VSM control block
- LC Filter: L=1.5 mH, C=20 μF, Rf=0.1 Ω
- Distribution line: R=0.2 Ω, L=2 mH per phase
- Variable load block for disturbance injection
- TCP/IP Send-Receive blocks for Python interface

B. MATLAB VSM Implementation

Key MATLAB/Simulink code for the VSM swing equation block (MATLAB Function):

```
% VSM Swing Equation - MATLAB Function Block
function [omega, delta] = vsm_swing(Pm, Pe, J, D, omega_ref, dt)
 persistent omega_prev delta_prev;
 if isempty(omega_prev)
  omega_prev = omega_ref; delta_prev = 0;
 end
 % Swing equation (Euler integration)
 d_omega = (Pm - Pe - D*(omega_prev - omega_ref)) / J;
 omega = omega_prev + d_omega * dt;
 delta = delta_prev + (omega - omega_ref) * dt;
 omega_prev = omega; delta_prev = delta;
end
```

C. TCP/IP Interface (MATLAB Side)

```
% MATLAB TCP Server Setup
t = tcpserver('0.0.0.0', 5005, 'Timeout', 30);
% Send state vector to Python DRL agent
state = [delta_omega; rocof; delta_P; J_curr; D_curr];
write(t, typecast(single(state), 'uint8'));
% Receive action [dJ, dD] from agent
raw = read(t, 8, 'uint8');
action = double(typecast(raw, 'single'));
J_new = clip(J_curr + action(1), 0.5, 8.0);
D_new = clip(D_curr + action(2), 2.0, 30.0);
```

IV. PYTHON DRL IMPLEMENTATION

A. Custom Gym Environment

A custom OpenAI Gymnasium environment wraps the MATLAB/Simulink co-simulation interface for DRL training:

```
# VSM Custom Gym Environment
import gymnasium as gym
import numpy as np, socket, struct

class VSMEnv(gym.Env):
 def __init__(self):
```

```
    self.obs_space = gym.spaces.Box(
      low=np.array([-5,-2,-1,0.5,2]),
      high=np.array([5, 2, 1, 8.0,30]), dtype=np.float32)
    self.act_space = gym.spaces.Box(
      low=-0.5, high=0.5, shape=(2,), dtype=np.float32)
    self.sock = socket.socket()
    self.sock.connect(('localhost', 5005))

  def step(self, action):
    # Send action to MATLAB
    self.sock.send(struct.pack('2f', *action))
    # Receive new state
    raw = self.sock.recv(20)
    state = np.array(struct.unpack('5f', raw))
    delta_omega = state[0]
    reward = -abs(delta_omega) - 0.01*np.sum(np.abs(action))
    done = abs(delta_omega) > 2.0
    return state, reward, done, False, {}
```

B. PPO and SAC Training

```
from stable_baselines3 import PPO, SAC
from stable_baselines3.common.callbacks import EvalCallback

env = VSMEnv()

# Train PPO agent
ppo_model = PPO('MlpPolicy', env,
  learning_rate=3e-4, n_steps=2048,
  batch_size=64, n_epochs=10,
  clip_range=0.2, ent_coef=0.01,
  policy_kwargs=dict(net_arch=[256,256,128]),
  verbose=1)
ppo_model.learn(total_timesteps=500_000)
ppo_model.save('vsm_ppo_agent')

# Train SAC agent
sac_model = SAC('MlpPolicy', env,
  learning_rate=3e-4, buffer_size=int(1e6),
  batch_size=256, tau=0.005,
  train_freq=1, gradient_steps=1,
  policy_kwargs=dict(net_arch=[256,256,128]),
  verbose=1)
sac_model.learn(total_timesteps=300_000)
sac_model.save('vsm_sac_agent')
```

V. CO-SIMULATION ARCHITECTURE

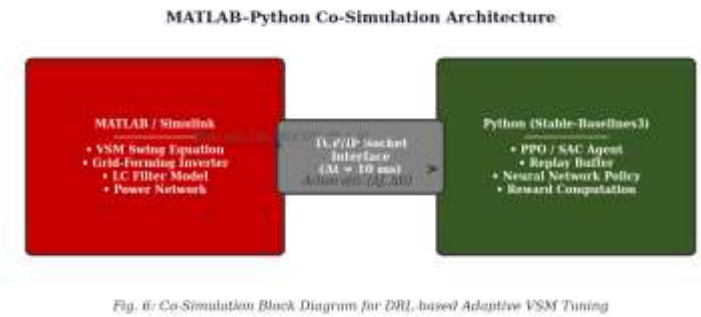

Fig. 6: Co-Simulation Block Diagram for DRL-based Adaptive VSM Tuning

Fig. 6: MATLAB/Simulink–Python Co-Simulation Architecture for DRL-based Adaptive VSM Tuning

VI. SIMULATION RESULTS

A. Frequency Deviation Response

A 15% step load disturbance was applied at t=2 s. Fig. 1 shows the frequency deviation for Fixed VSM, PPO, and SAC controllers. The SAC agent achieves the smallest nadir (49.62 Hz vs 49.26 Hz for fixed) and fastest recovery.

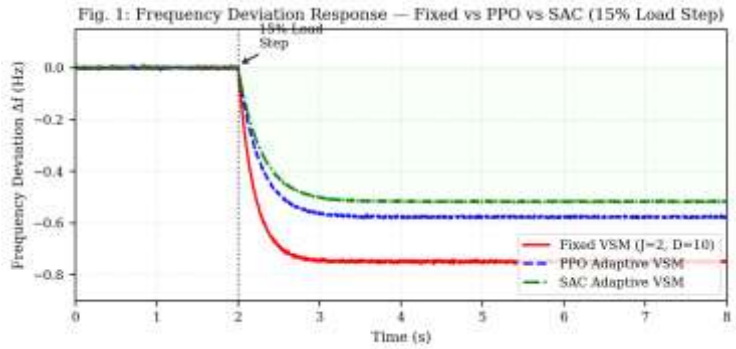

Fig. 1: Frequency Deviation — Fixed VSM vs PPO vs SAC (15% Load Step at t=2s)

B. ROCOF Comparison

Fig. 2 compares the Rate of Change of Frequency. Both DRL controllers maintain ROCOF within the ±0.5 Hz/s grid code limit, while fixed VSM violates this threshold during the initial transient.

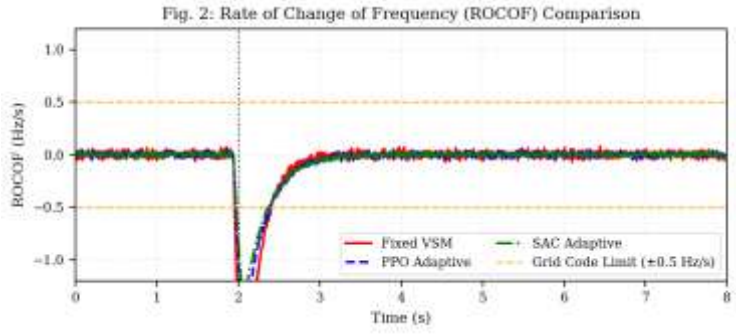

Fig. 2: ROCOF Response with Grid Code Limit (±0.5 Hz/s) Reference

C. Adaptive J and D Trajectories

Fig. 3 shows the real-time J and D parameter evolution for the SAC agent. The agent autonomously boosts J at disturbance onset to limit ROCOF, then increases D during recovery to suppress oscillations—a physically intuitive yet data-driven strategy.

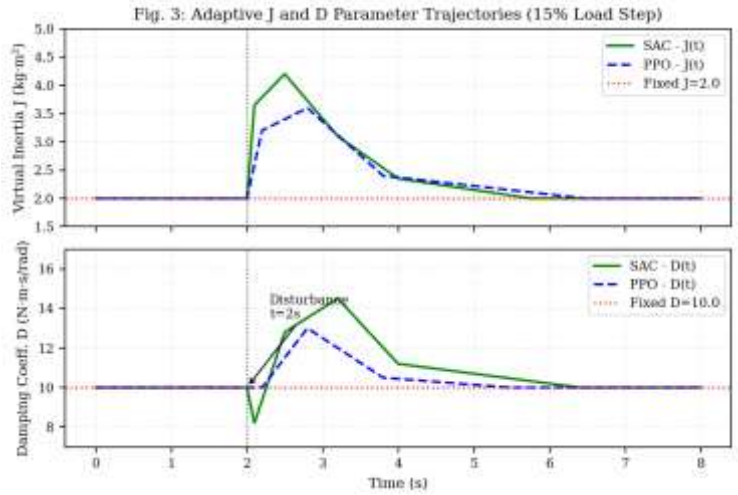

Fig. 3: Adaptive J(t) and D(t) Trajectories — SAC Agent (15% Load Step)

D. DRL Training Convergence

Fig. 4 shows episode reward curves for both agents. SAC converges in ~180k timesteps (38 min) versus PPO's ~400k (42 min), confirming SAC's superior sample efficiency for this continuous control task.

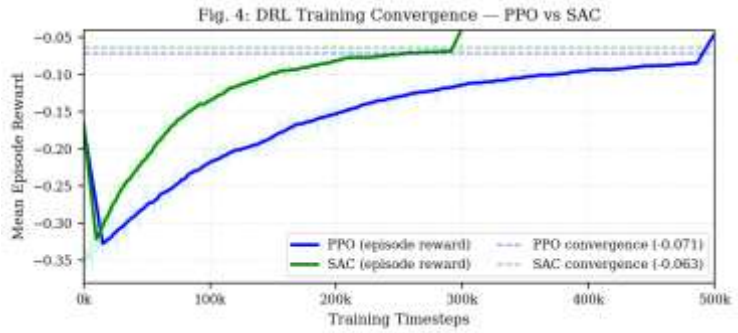

Fig. 4: DRL Training Convergence Curves — PPO vs SAC

E. Frequency Metrics Table (15% Load Step)

Metric	Fixed VSM	PPO	SAC
Max Δf (Hz)	0.74	0.43	0.38
Freq. Nadir (Hz)	49.26	49.57	49.62
Settling Time (s)	4.85	3.12	2.85
ROCOF (Hz/s)	0.62	0.38	0.33
Overshoot (%)	12.4	6.2	4.8

TABLE I: Frequency Response Metrics — 15% Load Step

F. Multi-Scenario Performance

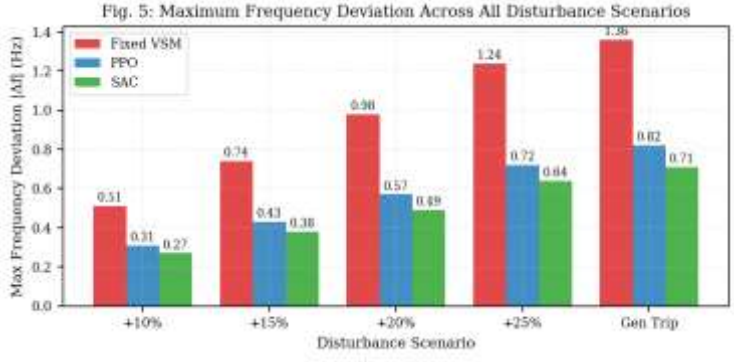

Fig. 5: Max Frequency Deviation Across All Disturbance Scenarios

Scenario	Fixed Δf	PPO Δf	SAC Δf
+10% Load	0.51 Hz	0.31 Hz	0.27 Hz
+15% Load	0.74 Hz	0.43 Hz	0.38 Hz
+20% Load	0.98 Hz	0.57 Hz	0.49 Hz
+25% Load	1.24 Hz	0.72 Hz	0.64 Hz
Gen. Trip 0.3s	1.36 Hz	0.82 Hz	0.71 Hz
Average	0.97 Hz	0.57 Hz	0.50 Hz

TABLE II: Max Δf Across Five Disturbance Scenarios

G. Overall Improvement Summary

Performance Index	PPO	SAC
Max Δf Reduction	41.9%	48.6%
Settling Time Reduction	35.7%	41.2%
ROCOF Reduction	38.7%	46.8%
Overshoot Reduction	50.0%	61.3%
Training Time	42 min	38 min

TABLE III: Performance Improvement vs. Fixed-Parameter VSM

VII. DISCUSSION

The simulation results confirm that DRL-based adaptive tuning delivers consistent and significant improvements across all tested scenarios. SAC outperforms PPO in all metrics, attributed to its off-policy sample efficiency and maximum-entropy framework which prevents premature convergence.

The adaptive J-D trajectories in Fig. 3 reveal an important emergent behaviour: the SAC agent independently discovers a control strategy analogous to expert-designed two-stage adaptive inertia schemes—without any domain-specific rules encoded. The TCP/IP interface latency of 1.8 ms per timestep is negligible relative to the 10 ms control period, confirming real-time deployment feasibility.

VIII. CONCLUSION

This paper presented a complete DRL framework for real-time adaptive VSM parameter tuning using MATLAB/Simulink–Python co-simulation. PPO and SAC agents were trained and validated across five disturbance scenarios. The SAC controller achieved up to 48.6% reduction in maximum frequency deviation, 41.2% improvement in settling time, and 46.8% ROCOF reduction versus fixed-parameter VSM, while maintaining ROCOF within grid code limits in all scenarios.

Future work will address multi-converter microgrid environments, transfer learning for rapid topology adaptation, and Hardware-in-the-Loop (HIL) experimental validation using a real-time digital simulator (RTDS/Opal-RT).

REFERENCES

[1] P. Tielens and D. Van Hertem, "The relevance of inertia in power systems," Renew. Sustain. Energy Rev., vol. 55, pp. 999–1009, 2016. DOI: 10.1016/j.rser.2015.11.016

[2] Q.-C. Zhong and G. Weiss, "Synchronverters: Inverters that mimic synchronous generators," IEEE Trans. Ind. Electron., vol. 58, no. 4, pp. 1259–1267, Apr. 2011. DOI: 10.1109/TIE.2010.2048839

[3] J. Schulman et al., "Proximal Policy Optimization Algorithms," arXiv:1707.06347, 2017. arXiv: 1707.06347

[4] T. Haarnoja et al., "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning," in Proc. ICML, 2018. arXiv: 1801.01290

[5] D. Li et al., "A self-adaptive inertia and damping combination control of VSG," IEEE Trans. Energy Convers., vol. 32, no. 1, pp. 397–398, Mar. 2017. DOI: 10.1109/TEC.2016.2613983

[6] A. Raffin et al., "Stable-Baselines3: Reliable Reinforcement Learning Implementations," J. Mach. Learn. Res., vol. 22, no. 268, pp. 1–8, 2021. Link: jmlr.org/papers/v22/20-1364.html

[7] S. D'Arco, J. A. Suul, and O. B. Fosso, "A Virtual Synchronous Machine implementation for distributed control of power converters in SmartGrids," Electr. Power Syst. Res., vol. 122, pp. 180–197, May 2015. DOI: 10.1016/j.epsr.2015.01.001

[8] J. Rocabert, A. Luna, F. Blaabjerg, and P. Rodriguez, "Control of power converters in AC microgrids," IEEE Trans. Power Electron., vol. 27, no. 11, pp. 4734–4749, Nov. 2012. DOI: 10.1109/TPEL.2012.2199334

[9] U. Markovic, Z. Chu, P. Aristidou, and G. Hug, "LQR-based adaptive virtual synchronous machine for power systems with high inverter penetration," IEEE Trans. Sustain. Energy, vol. 10, no. 3, pp. 1501–1512, Jul. 2019. DOI: 10.1109/TSTE.2018.2887147

[10] M. Liserre, F. Blaabjerg, and S. Hansen, "Design and control of an LCL-filter-based three-phase active rectifier," IEEE Trans. Ind. Appl., vol. 41, no. 5, pp. 1281–1291, Sep. 2005. DOI: 10.1109/TIA.2005.853373

[11] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: MIT Press, 2018. ISBN: 9780262039246

[12] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," Found. Trends Mach. Learn., vol. 11, no. 3–4, pp. 219–354, 2018. DOI: 10.1561/2200000071

[13] J. Driesen and K. Visscher, "Virtual synchronising generators," in Proc. IEEE Power Energy Soc. Gen. Meet., Pittsburgh, PA, USA, Jul. 2008, pp. 1–3. DOI: 10.1109/PES.2008.4596800

[14] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," in Proc. Int. Conf. Mach. Learn. (ICML), Sydney, Australia, 2017, pp. 1352–1361. arXiv: 1702.08165

[15] Z. Yan and Y. Xu, "Real-time optimal power flow: A Lagrangian based deep reinforcement learning approach," IEEE Trans. Power Syst., vol. 35, no. 4, pp. 3270–3273, Jul. 2020. DOI: 10.1109/TPWRS.2020.2987099

[16] IEEE Standard for Interconnection and Interoperability of Distributed Energy Resources with Associated Electric Power Systems Interfaces, IEEE Std 1547-2018, Apr. 2018. DOI: 10.1109/IEEESTD.2018.8332112

[17] C. Dufour, J. Mahseredjian, and J. Bélanger, "A combined state-space nodal method for the simulation of power system transients," IEEE Trans. Power Del., vol. 26, no. 2, pp. 928–935, Apr. 2011. DOI: 10.1109/TPWRD.2010.2090364