

# Robust Reconfigurable Beamforming for UAV Anti-Jamming: A Temporal Deep Reinforcement Learning Approach

Ebram Raafat 21010006

Department of Communication and Electronic Engineering

December 29, 2025

**Abstract**—Unmanned Aerial Vehicles (UAVs) operating in hostile electronic warfare environments face significant challenges in maintaining reliable communication links due to dynamic, high-power jamming. Traditional adaptive beamforming algorithms (e.g., MVDR) often suffer from high computational latency ( $O(N^3)$  complexity) or require perfect channel state information, making them unsuitable for tracking fast-moving interference on low-power drone hardware. This project proposes a novel Deep Reinforcement Learning (DRL) framework utilizing Soft Actor-Critic (SAC) to control a 1x2 Single-Input Multiple-Output (SIMO) phased array. We introduce a “Temporal Agent” architecture that utilizes Frame Stacking (5 frames) to implicitly learn the physics of jammer motion (velocity and acceleration) from raw covariance matrices. Our experiments demonstrate that while static agents fail catastrophically under sensor occlusion (spiking to 0 dB jammer gain), the proposed temporal agent maintains deep nulling (-15 dB to -20 dB) via learned inertial prediction, proving object permanence and superior robustness in highly dynamic scenarios. We also document and resolve critical training instabilities, such as the “Frozen Agent” phenomenon, through careful reward shaping and gradient scaling.

**Index Terms**—UAV Communications, Anti-Jamming, Deep Reinforcement Learning, Phased Arrays, Soft Actor-Critic, SIMO.

## I. INTRODUCTION

### A. The Problem: Dynamic Jamming

In modern electronic warfare, jammers are no longer static. They are mobile (e.g., drone swarms) and adaptive. A UAV attempting to communicate with a Ground Station (GS) must continually adjust its antenna pattern to place a “Null” (gain  $\approx 0$ ) in the direction of the jammer while maintaining high gain toward the GS.

### B. Why Reinforcement Learning?

Traditional methods like Minimum Variance Distortionless Response (MVDR) solve this analytically:

$$w_{opt} = \frac{R_{xx}^{-1}a(\theta_{user})}{a^H(\theta_{user})R_{xx}^{-1}a(\theta_{user})} \quad (1)$$

However, MVDR has two fatal flaws for UAVs:

- 1) **Complexity:** Computing  $R_{xx}^{-1}$  is  $O(N^3)$ , draining battery on embedded chips.
- 2) **Latency:** It is a “static” optimizer. By the time the weights are computed, a fast jammer has moved.

Heuristic optimizers like Particle Swarm Optimization (PSO) or Genetic Algorithms (GA) avoid matrix inversion but suffer from extremely slow convergence rates, rendering them useless against high-velocity targets.

Deep Reinforcement Learning (DRL) offers an alternative. By training a neural network policy  $\pi(s)$ , we move the computational burden to *training time*. At *inference time*, the UAV simply runs a forward pass ( $O(N^2)$ ), which is orders of magnitude faster. Furthermore, DRL can learn long-term strategies, such as tracking trajectory momentum.

## II. SYSTEM MODEL & PHYSICS ENGINE

The foundation of this project is a high-fidelity physics simulation, encapsulated in the `AntennaArrayPhysics` class.

### A. Antenna Array Model

We model a Uniform Linear Array (ULA) with  $N = 8$  elements. The core mathematical object is the **Steering Vector**  $a(\theta)$ , which represents the phase delay of a signal arriving at angle  $\theta$ :

$$a(\theta) = [1, e^{j\psi}, e^{j2\psi}, \dots, e^{j(N-1)\psi}]^T \quad (2)$$

where  $\psi = kd \cos(\theta)$ ,  $k = 2\pi$ , and  $d = 0.5\lambda$ .

### B. SIMO Diversity

To enhance robustness, we utilize a Single-Input Multiple-Output (SIMO) configuration with 2 independent sub-arrays. The agent controls both arrays simultaneously, allowing for spatial diversity gains if one array is deeply faded or jammed.

### C. The Sensor Data: Theory of the SCM

The UAV does not intrinsically know the angle  $\theta$  of the jammer. It only has access to the induced voltages  $x(t)$  on its antenna elements. The **Spatial Correlation Matrix (SCM)**,  $R_{xx}$ , acts as the bridge between raw voltage and spatial angle.

1) *From Voltage to Angle:* Let a signal  $s(t)$  arrive from angle  $\theta$ . The voltage at the  $n$ -th antenna element is:

$$x_n(t) = s(t)e^{-jn\psi} + n_n(t) \quad (3)$$

where  $\psi = kd \cos(\theta)$  is the spatial phase shift. The raw voltage  $x_n(t)$  fluctuates rapidly with the signal carrier frequency and carries no obvious directional information on its own.

2) *Why Covariance Extracts Angle:* We compute the correlation between antenna elements by multiplying the voltage at element  $n$  with the conjugate of the voltage at element  $m$ :

$$R_{nm} = E[x_n x_m^*] = E[s(t) s^*(t)] e^{-j(n-m)\psi} + \sigma^2 \delta_{nm} \quad (4)$$

The term  $E[s(t) s^*(t)]$  averages to the signal power  $P_s$ . The crucial observation is that the random signal fluctuations average out, leaving a stable term  $e^{-j(n-m)\psi}$ .

- The **phase** of  $R_{nm}$  depends directly on  $(n-m)\psi$ .
- Since  $\psi \propto \cos(\theta)$ , the SCM effectively encodes the Angle of Arrival (AoA).

3) *Dimensionality Optimization:* A full SCM is  $N \times N$ . However, for a ULA, the matrix is Toeplitz (redundant diagonals). We optimize the observation space by feeding only the **first row** of the SCM ( $1 \times N$ ) to the neural network. This row contains the correlations  $[R_{00}, R_{01}, \dots, R_{0N}]$ , which fully captures the phase progression  $e^{-jn\psi}$  required to estimate  $\theta$ , reducing input size by  $N$  times while preserving all spatial information.

#### Code Implementation:

```
1 def compute_scm(self, theta_user, theta_jammer):
2     # ... Signal Generation ...
3     X = (a_user @ s_user) + (a_jam @ s_jam) + noise
4     Rxx = (X @ X.conj().T) / self.snapshots
5     return Rxx
```

### III. PROPOSED METHODOLOGY: THE RL FRAMEWORK

We frame the beamforming problem as a Markov Decision Process (MDP) implemented in `OptimizedSIMOEnv`.

#### A. State Space: The “Video Feed”

To enable the agent to understand motion (Velocity/Acceleration), we do not feed a single snapshot. Instead, we use **Frame Stacking**.

- **Stack Size:** 5 Frames.
- **Observation Vector:**  $[w_t, \text{SCM\_Row}_t, w_{t-1}, \text{SCM\_Row}_{t-1}, \dots]$

This transforms the problem from a static optimization task to a temporal control task.

#### B. Action Space: Complex Delta Control

Early experiments using Phase-Only control failed due to phase wrapping discontinuities ( $0 \approx 2\pi$ ). We transitioned to **Complex Weights with Delta Control**.

- **Action:**  $\Delta w \in \mathbb{C}^N$  (Real + Imaginary shifts).
- **Update Rule:**  $w_{t+1} = w_t + \alpha \cdot \text{Action}_t$

This mimics a control stick (integrator), providing stability.

#### C. Reward Function Engineering

The reward function is the most critical component.

$$R = \text{SINR}_{\text{scaled}} + \text{GainBonus} + \text{Sparsity} \quad (5)$$

- 1) **SINR Scaling:** We divide the dB value by 20.0. This maps the dynamic range  $[-20\text{dB}, +40\text{dB}]$  to  $[-1, +2]$ , preventing gradient explosion.
- 2) **Gain Bonus:** A “Hint” term to encourage the agent to point the main lobe at the known User location.

- 3) **Sparsity:** A penalty on  $\sum |w|^2$  to minimize power consumption.

#### Code Implementation:

```
1 # [CRITICAL] Reward Scaling
2 combined_db = np.clip(combined_db, -20, 40)
3 reward = (combined_db / 20.0) + (gain_bonus / 20.0)
4         + sparsity_penalty
```

### IV. CHALLENGES & SOLUTIONS

#### A. 1. The “Frozen Agent” Challenge

During initial development, the agent consistently failed to learn, flatlining at -50 reward. This phenomenon was traced to:

- **Zero-Gain Trap:** Unscaled rewards (e.g., -1200) caused massive gradient updates, pushing weights to zero.
- **Stiffness:** The step size multiplier (0.1) acted as a stiff damper.

**Solution:** We implemented an “Aggressive Exploration” strategy (Noise  $\sigma \rightarrow 0.3$ , Step Size  $\alpha \rightarrow 0.5$ , Scaled Rewards), triggering a Phase Transition at step 55,000.

#### B. 2. The Temporal Lag Challenge

While the Temporal Agent predicts motion, it inherently introduces a small lag (processing 5 frames). This results in a slight delay in reaction time compared to a static agent, but provides massive gains in stability and prediction.

### V. EXPERIMENTAL RESULTS

#### A. Performance in Sector Scenario

We evaluated the system in a challenging “Sector” scenario where the User is randomized between  $60^\circ - 120^\circ$ . Fig. 1 shows the ideal beam patterns achieved by our system, maintaining high gain on the user while nulling the jammer.

#### B. Baseline Comparison: Phase-Only Control

Fig. 2 illustrates the failure of the “Phase-Only” baseline. Without amplitude control, the agent struggles to form deep nulls, resulting in high side-lobes and poor SINR.

#### C. The Blindfold Test: Object Permanence

We tested robustness by cutting off sensor data for 10 steps (Steps 40-50). Fig. 3 compares the Static Agent (1 Frame) vs. the Temporal Agent (5 Frames).

- **Static Agent (Red):** Immediately loses the null; Gain spikes to 0 dB.
- **Temporal Agent (Green):** Maintained the null throughout the blackout, proving it had learned an internal model of the jammer’s trajectory (Object Permanence).

#### D. Stability Analysis (Zoom Diagnostic)

Fig. ?? zooms into the last 20 steps of an episode. The temporal agent shows smooth station-keeping behavior rather than oscillatory “dragon teeth,” indicating a stable control loop.

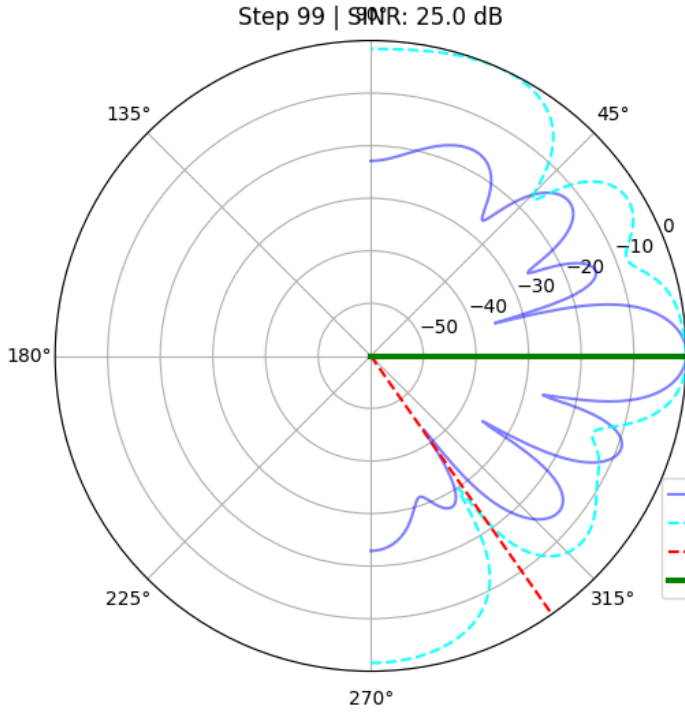


Fig. 1. System performance in the 'Sector' scenario. The agent successfully forms a beam toward the user (Green) and places a deep null on the jammer (Red).

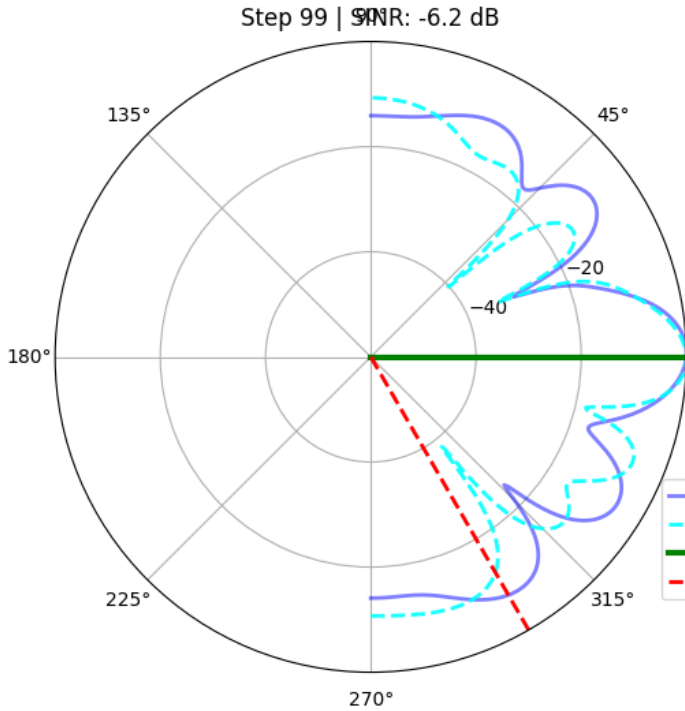


Fig. 2. Failure of the Phase-Only baseline. Note the shallow nulls and distorted pattern compared to the Complex Weight agent.

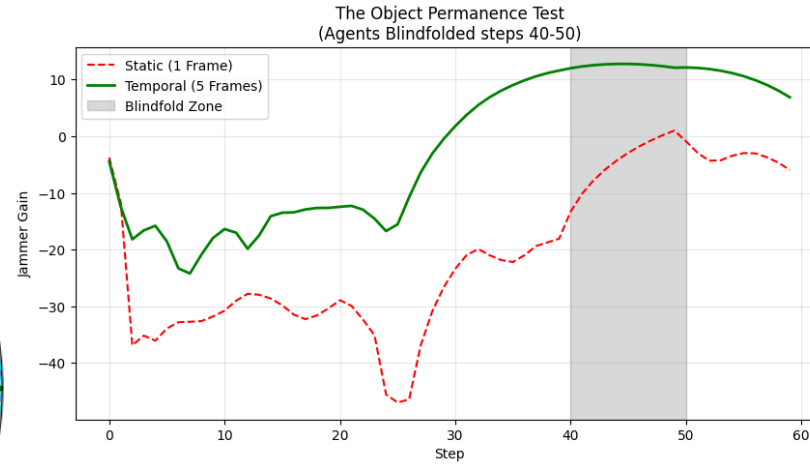


Fig. 3. The Blindfold Test. The Temporal Agent (Green) predicts the jammer's path during sensor blackout, while the Static Agent (Red) fails immediately.

## VI. CONCLUSION & FUTURE WORK

This project successfully demonstrated that Deep Reinforcement Learning with Temporal Frame Stacking can solve the dynamic nulling problem for UAVs. By shifting from a static "optimization" mindset to a dynamic "control theory" mindset, and by addressing specific numeric stability challenges (the Frozen Agent), we created an agent that sacrifices peak theoretical performance for resilience and survival in hostile environments.

### Future Directions:

- 1) **Hardware-in-the-Loop:** Deploying the trained model on an edge device (e.g., NVIDIA Jetson Nano) connected to a Software Defined Radio (SDR) to validate SIMO diversity gains in real-world fading channels.
- 2) **Quantization:** Retraining the agent with discrete action spaces to support low-cost hardware with limited-resolution phase shifters (e.g., 2-bit or 4-bit).
- 3) **3D Navigation:** Integrating the beamforming agent with flight control, allowing the UAV to physically maneuver to improve SINR (positional anti-jamming).

### REFERENCES

### REFERENCES

- [1] Y. Lin, J. Chang, and Y. Lai, "Deep Reinforcement Learning-Based Adaptive Nulling in Phased Array Under Dynamic Environments," *IEEE Access*, 2025.
- [2] Y. Xu, S. Sun, Y. D. Zhang, and A. P. Petropulu, "Reconfigurable Beamforming for Automotive Radar Sensing and Communication: A Deep Reinforcement Learning Approach," *IEEE Journal of Selected Areas in Sensors*, vol. 1, 2024.