Article

Not peer-reviewed version

Neuromorphic Control of the Serv-Arm Robot Using Spiking Neural Networks

Alfredo Paulo Oliveira Barros [*,†], Reyner Carlos Silva Alegria [†], Gabriel Moriz da Silva [†], Pedro Victor dos Santos Matias [†], Carlos Alberto Oliveira de Freitas [†], Vicente Ferreira de Lucena Junior [†], Vandermi João da Silva [†]

Posted Date: 17 December 2025

doi: 10.20944/preprints202512.1045.v1

Keywords: neuromorphic engineering; spiking neural networks; robot control; embedded systems; energy measurement; Raspberry Pi

Article

Neuromorphic Control of the Serv-Arm Robot Using Spiking Neural Networks

Alfredo Paulo Oliveira Barros [1,*,†], Reyner Carlos Silva Alegria [1,†], Gabriel Moriz da Silva [1,†], Pedro Victor dos Santos Matias [2,†], Carlos Alberto Oliveira de Freitas [1,†], Vicente Ferreira de Lucena Junior [1,†] and Vandermi João da Silva [1,†]

[1] Institute of Exact and Technological Sciences (ICET), Federal University of Amazonas (UFAM), Itacoatiara, Brazil
[2] Motorola Mobility LLC, Brazil
* Correspondence: alfredo.barros@ufam.edu.br
† These authors contributed equally to this work.

Highlights

What are the main findings?

- A neuromorphic joint-space controller for a low-cost 4-DoF robotic manipulator.
- The algorithm achieves an 11.06° MAE via a 128-neuron LIF architecture.
- The real-world embedded evaluation reveals comparable power consumption between SNN and ANN controllers.

What are the implications of the main findings?

- This demonstrates that SNN-based robotic control is feasible for low-cost embedded hardware.
- A baseline for the energy behavior of SNNs on CPUs is established, enabling comparison with future neuromorphic hardware.
- The potential of neuromorphic methods for robotics should be reinforced even before specialized hardware is introduced.

Abstract

This work introduces a neuromorphic joint-space controller for the 4-DoF Serv-Arm robotic manipulator, implemented via a Spiking Neural Network (SNN) on low-cost embedded hardware. Unlike traditional methods based on inverse kinematics or Cartesian-space control, the proposed architecture operates directly on servo joint angles, receiving current and target configurations to predict the following joint-space action. A synthetic dataset with over 500,000 samples was generated to comprehensively cover the robot's workspace, including oversampling of mechanically challenging configurations. The SNN was trained with surrogate gradients (*snnTorch*), and a systematic grid search optimized the hidden layer size, learning rate, and membrane decay factor. The final model, featuring a 128-neuron LIF hidden layer, achieved a Mean Absolute Error (MAE) of 11.06° on the validation set and generated smooth, reproducible trajectories during real-robot execution. Additionally, this study includes an empirical energy comparison between the SNN controller and an ANN baseline on a Raspberry Pi under identical motion routines. The results indicate that both models consume similar average powers, with the SNN showing a modest increase due to the temporal simulation overhead typical of spiking models on standard CPUs.

Keywords: neuromorphic engineering; spiking neural networks; robot control; embedded systems; energy measurement; Raspberry Pi

 doi:10.20944/preprints202512.1045.v1

1. Introduction

Robotic manipulators are crucial in manufacturing, inspection, assistance, and autonomous systems. Designing controllers for these platforms requires balancing accuracy, robustness, computational speed, and real-time operation, a challenge that becomes even greater when using embedded or low-cost hardware. Traditional control methods, such as proportional–integral–derivative (PID) controllers, are popular because they are simple to implement. Nevertheless, they often depend on precise mathematical models and can struggle with uncertainties, nonlinear behaviors, or actuator issues. Additionally, computing inverse kinematics in real time is computationally intensive, which limits its use on embedded processors with restricted capacity [1].

Spiking Neural Networks (SNNs) have emerged as an alternative computing paradigm inspired by the event-driven communication of biological neurons. Unlike traditional artificial neural networks, SNNs process information through sparse spike-based signals, enabling low-latency computation and potentially higher energy efficiency—characteristics that make them especially attractive for real-time robotic control on constrained hardware platforms [2,3]. These features suggest that SNNs could serve as a promising foundation for embedded manipulation tasks, particularly when combined with lightweight robotic systems.

Despite these advantages, the use of SNN-based controllers in low-cost robotic manipulators is still limited. Current research efforts tend to focus on high-performance neuromorphic hardware or simulation environments, with little attention given to assessing SNN controllers on affordable, standard hardware in practical control tasks. Additionally, the energy consumption associated with deploying SNNs compared with traditional neural architectures in embedded robotic systems remains poorly understood.

To address these gaps, this work develops and evaluates a neuromorphic controller for a low-cost Serv-Arm manipulator implemented on a Raspberry Pi–based embedded platform. The proposed system covers the entire pipeline—from synthetic dataset creation to model training, hyperparameter tuning, and real-time inference. In addition to evaluating tracking accuracy and neural dynamics, this study offers an empirical analysis of energy consumption, enabling a direct comparison between SNN- and ANN-based controllers under identical execution conditions.

The main contributions of this work are as follows:

- Developing a fully reproducible neuromorphic control framework for the Serv-Arm robotic manipulator using low-cost embedded hardware;
- Systematically optimizing SNN hyperparameters through grid search applied to a large, synthetically generated training dataset;
- Experimentally evaluating tracking accuracy, prediction error distribution, and spiking activity dynamics;
- Empirically comparing the energy consumption of SNN and ANN controllers based on continuous power measurements.

The rest of this article is organized as follows. Section 2 reviews related work on robotic manipulation and neuromorphic control. Section 3 details the methods used for data generation, model design, and system implementation. Section 4 provides the experimental evaluation and results. Section 5 discusses the findings, and Section 6 concludes the paper while indicating directions for future work.

2. Related Work

2.1. Robotic Manipulators: Kinematics, Dynamics, and Classical Control

Robotic manipulators are complex systems composed of rigid links connected by actuated joints. Their control has traditionally depended on mathematical models involving forward kinematics, inverse kinematics, and dynamics. While forward kinematics are straightforward to compute, inverse kinematics (IK) are nonlinear, often ill-conditioned, and computationally intensive, particularly when executed on embedded systems with limited resources [1]. Conventional control methods, such as proportional–integral–derivative (PID) controllers, remain popular due to their simplicity and

 doi:10.20944/preprints202512.1045.v1

predictable behavior. However, their effectiveness heavily relies on accurate modeling and proper gain tuning, making them vulnerable to issues such as friction, payload changes, mechanical backlash, and structural uncertainties commonly found in low-cost manipulators such as the Serv-Arm.

The computational cost of real-time IK has led to the development of alternative methods that avoid explicit geometric modeling. Among these, joint-space control architectures—in which control signals are predicted directly in the servo-angle domain—have become especially relevant for lightweight and affordable robotic platforms.

2.2. The Spiking Neural Network (SNN) Paradigm and Neuromorphic Computation

Spiking neural networks (SNNs), regarded as the third generation of neural models, operate on discrete temporal events (spikes), enabling sparse and energy-efficient computation. The most common neural model is the leaky integrate-and-fire (LIF), in which the membrane potential accumulates incoming currents until it reaches a threshold and generates a spike. Its dynamics are described by:

$$\tau_m \frac{dV(t)}{dt} = -\big(V(t) - V_{\text{rest}}\big) + RI(t) \tag{1}$$

where τ_m is the membrane time constant, V_{rest} is the resting potential, R is the membrane resistance, and $I(t)$ is the synaptic input current. After the potential reaches the threshold V_{th}, a spike is emitted, and $V(t)$ is reset.

SNNs encode information through firing rates or precise spike timing [4]. Learning mechanisms such as spike-timing-dependent plasticity (STDP) provide biologically inspired adaptation rules. Neuromorphic processors such as Intel Loihi [5] and IBM TrueNorth [6] implement these neural dynamics directly in hardware, using asynchronous, event-driven computation that consumes extremely low power—an appealing feature for embedded robotic control.

2.3. Applications of SNNs in Robotic Control

Recent research has shown rapid growth in the use of SNNs for robotic control tasks. Fully spiking PID controllers have demonstrated competitive performance in multiaxis manipulators, often outperforming traditional controllers under dynamic disturbances [7]. For more complex systems, recurrent SNNs have been applied to manipulators with 6–7 degrees of freedom, learning motor coordination through spatiotemporal correlations [8]. Combined with reinforcement learning (RL), SNNs can acquire manipulation skills directly from interaction with the environment, without requiring explicit mechanical models [9]. Cerebellar-inspired SNN architectures have also shown the ability to predict and correct motion errors in real time, enhancing accuracy and reducing latency [10].

Neuromorphic hardware further expands the potential of spike-based controllers. Loihi achieves energy-delay products up to 100× lower than CPU implementations [11], while TrueNorth consumes only 65 mW during real-time inference and delivers more than 6,000 frames per watt [12]. Additional advances such as the ODIN chip achieve energy costs as low as 12.7 pJ per synaptic operation, reinforcing the feasibility of spike-based computation for embedded robotic platforms [13].

SNNs have also been applied to low-power embedded control of 4-DoF manipulators using spike-based learning [14], and to real-time manipulation tasks using event-driven sensory inputs [15]. Unsupervised SNNs can learn motor patterns solely from their temporal structure [16], while bioinspired spiking controllers demonstrate robustness to disturbances and model uncertainties [17]. Beyond manipulation, SNNs have also been explored in sensing, object detection [18], human–machine interfaces [19,20], and biological signal processing [21].

Despite these advances, most approaches rely on Cartesian-space control or implicit inverse kinematics. Few studies have investigated supervised SNNs for direct joint-space control, particularly for low-cost, limited-precision manipulators. This gap is directly addressed in the present work.

 doi:10.20944/preprints202512.1045.v1

2.4. Energy Efficiency and Embedded Execution of SNNs

Energy efficiency is a key motivation for adopting SNNs in robotic control. Neuromorphic processors leverage sparse, event-driven computation to achieve substantial improvements in energy consumption. Loihi consistently outperforms GPUs and CPUs across diverse workloads, achieving up to 100× better energy-delay products [11]. TrueNorth provides highly energy-efficient inference, operating at 25–275 mW while delivering high throughput [12]. The ODIN chip offers exceptionally low energy per synaptic operation (12.7 pJ), supporting compact and versatile spike-based computation [13].

However, accurately measuring energy consumption on embedded platforms remains challenging. Osolinskyi et al. examined traditional methods for current measurement in microcontrollers, highlighting limitations of shunt resistors, multimeters, and reactive components that may introduce distortions or instability [22]. Carroll and Heiser [23] presented detailed methodologies using sense resistors and high-speed data acquisition systems to obtain accurate power profiles in embedded devices. These studies emphasize the need for rigorous and reproducible energy benchmarking protocols.

Although neuromorphic chips offer excellent energy efficiency, relatively few studies have evaluated SNN performance on low-cost, general-purpose hardware such as Raspberry Pi or Arduino. This motivates the present work, which compares SNN and ANN power consumption under identical real-time robotic control workloads executed on a fully reproducible embedded platform.

2.5. Comparison Between PID, ANN-Based Controllers, and SNN Approaches

PID controllers remain standard in robotics due to their simplicity and well-understood behavior [1]. However, their dependence on accurate models and sensitivity to nonlinearities limit their applicability in inexpensive robotic systems. Artificial neural networks (ANNs) have become powerful tools for approximating nonlinear mappings and inverse kinematics [24]. Despite their capabilities, ANNs typically require large labeled datasets, rely on dense computation, and lack inherent temporal dynamics.

SNNs overcome several of these limitations by incorporating temporal processing, stateful neural dynamics, and event-driven execution [2]. Studies have shown that SNN-based controllers can outperform PID and ANN methods under noise, sudden disturbances, or dynamic conditions [7,17]. Their intrinsic temporal memory, low-latency response, and energy efficiency make them strong candidates for embedded robotic control [11,25].

Despite progress, most existing approaches rely on Cartesian-space control or SNN-based inverse kinematics. Few studies explore supervised SNN models that perform direct joint-space control by receiving the current and desired joint configurations and predicting the subsequent motor command. The present work addresses this gap by proposing and experimentally validating an embedded, energy-efficient SNN controller for a low-cost 4-DoF robotic arm based on direct joint-space inference.

3. Materials and Methods

3.1. Description and Assembly of the Serv-Arm Robot

The Serv-Arm is an open-source robotic platform that functions as an articulated manipulator arm, as shown in Figure 1. For educational, research, and experimental use, its design is universally accessible because most of its mechanical parts are 3D printed from affordable materials such as PLA, PETG, or ABS [26]. Its architecture draws inspiration from industrial robots, featuring a serial arrangement of links and joints that provide a wide range of movement and flexibility. The structure includes a fixed base, a shoulder link, an elbow link, and an end-effector, usually a gripper. In its standard setup, the Serv-Arm has four degrees of freedom (DoF) and is operated by servo motors, as described in the original project documentation [26].

 doi:10.20944/preprints202512.1045.v1

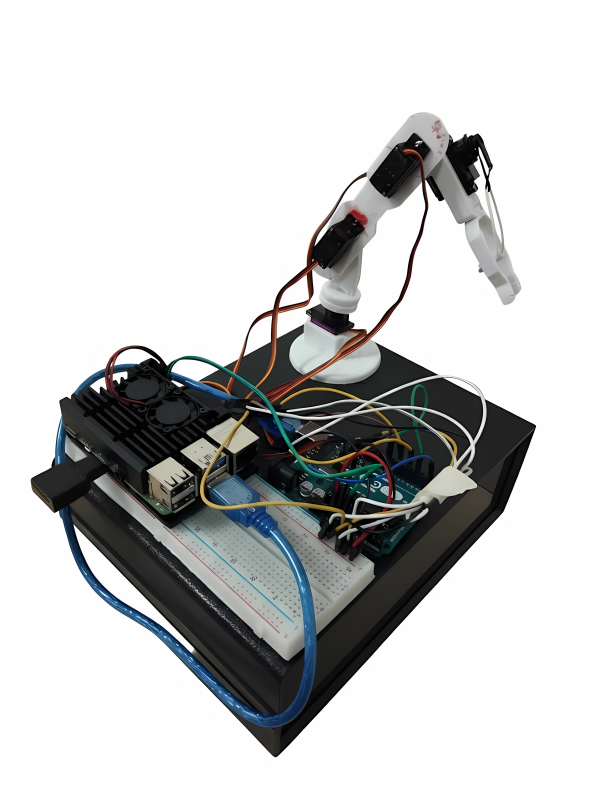

Figure 1. Physical prototype of the developed system, integrating the Serv-Arm manipulator with a Raspberry Pi 4B and an Arduino Uno R3 responsible for real-time control.

The core of the system comprises an Arduino Uno R3 microcontroller and a Raspberry Pi 4B, which are responsible for real-time control and the execution of SNN inference algorithms. The Arduino produces PWM signals for the servo motors, whereas the Raspberry Pi manages serial communication and neuromorphic model processing. Owing to its low cost and open-source design, the Serv-Arm–Arduino–Raspberry setup offers an excellent platform for experiments in robotic control and artificial intelligence [26].

The robot's physical assembly followed open-source project guidelines, including 3D printing structural parts, integrating servo motors (3× MG90S for main joints and 1× MG90S for the gripper), and mechanically calibrating the axes to ensure accurate motion and proper alignment between commands and the actual position of the manipulator.

For control, the main motion paths were defined, covering fundamental movements such as right, left, and upward, which served as the basis for SNN training. This definition considers the Serv-Arm's workspace and planned manipulation tasks, optimizing coverage of essential movements and ensuring representative trajectories.

3.2. Pick-and-Place Task and Motion Mapping

The main evaluation task for the system was a pick-and-place operation, in which the robot was instructed to pick up an object from a predefined initial position and move it to a predefined final position. This task demands precision in positioning and trajectory, making it an excellent benchmark for validating the neuromorphic control system. The trajectory was divided into subtasks (approach, grasp, lift, transport, release, and return), which simplified both training and performance analysis.

Unlike approaches that rely on geometric mapping algorithms to convert Cartesian coordinates into joint angles, this work trains the SNN directly to predict servo-motor angles. The model takes the current system states and desired commands as inputs and produces the normalized motor angles. Therefore, the network learns to determine the correct angles without relying on intermediate IK transformations, thereby reducing computational complexity and increasing efficiency.

 doi:10.20944/preprints202512.1045.v1

3.3. Dataset Construction and Preparation for Training

The performance of the neuromorphic controller relies directly on the quality and diversity of the dataset used to train the SNN. To ensure comprehensive coverage of the workspace, a fully synthetic dataset was created using a Python-based pipeline that simulates realistic manipulator configurations, deliberately focusing on mechanically challenging regions of the Serv-Arm. A fixed random seed was applied throughout the generation process to ensure reproducibility.

Dataset generation begins by creating a pose library. Several pick-and-place and point-to-point trajectories were simulated, and the corresponding joint angles were recorded. Additional CSV files containing the operational joint limits and a reference trajectory were combined into the pose library, resulting in a diverse set of unique, physically valid configurations. Each dataset entry includes three vectors: (i) the current joint configuration, (ii) a randomly chosen target configuration, and (iii) the next incremental step to be taken. To mimic real-world imprecision, Gaussian noise was independently added to each joint angle.

Motor 2 exhibited reduced noise variance because of its mechanical design. The shoulder joint handles vertical movement and bears most of the arm's load, making it especially sensitive to noise. Minor disturbances in this joint can cause significantly larger deviations in the end-effector's path, particularly in near-vertical positions. Lowering its injected noise helps produce more realistic samples and prevents implausible posture shifts. After noise was added, all the angles were clipped to stay within their physical limits.

To clarify the distribution, Figures 2 and 3 show histograms of the current, target, and next-step angles for all four motors. The multimodal patterns across joints highlight the focus on difficult workspace areas and support an intense training distribution.

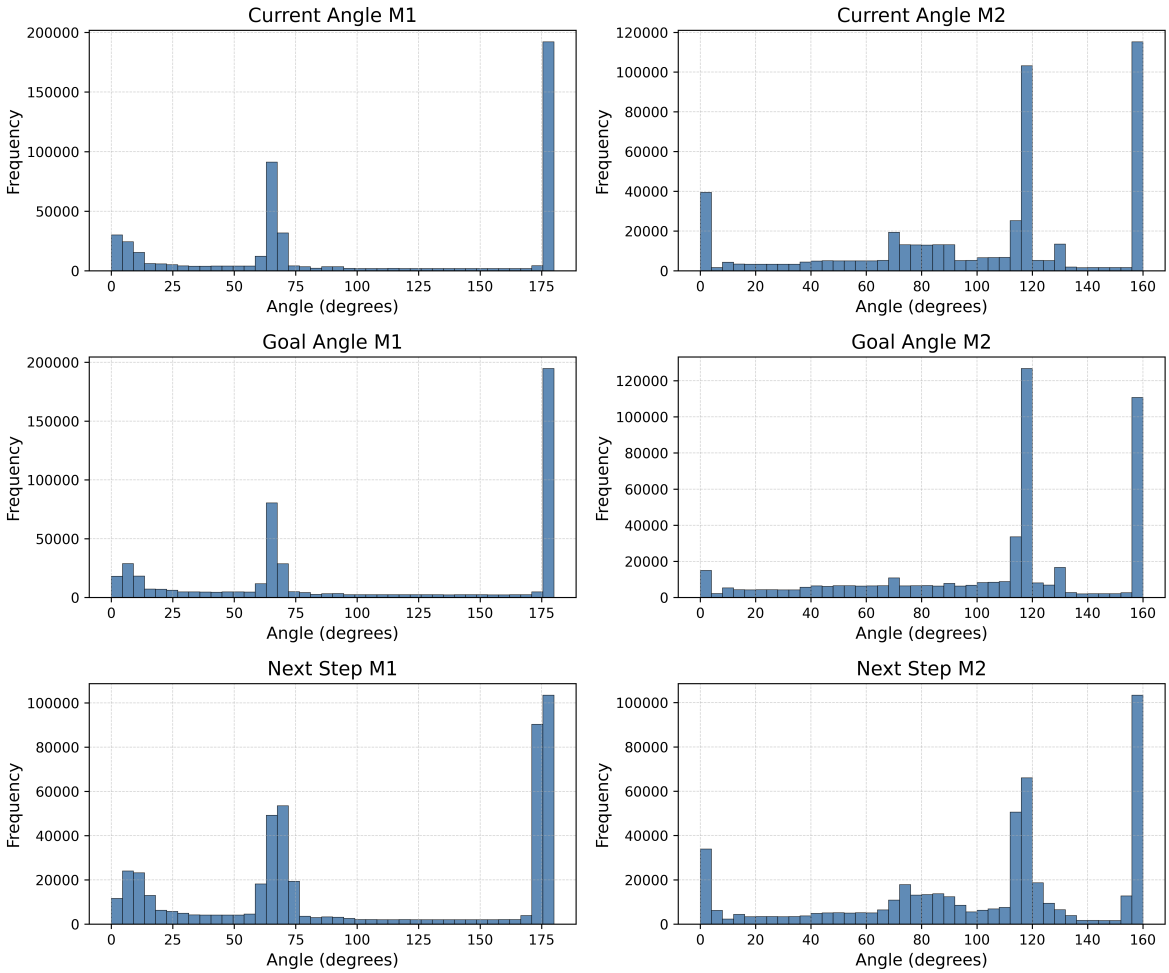

Figure 2. Distribution of current, goal, and next-step angles for motors M1 and M2 in the synthetic dataset.

 doi:10.20944/preprints202512.1045.v1

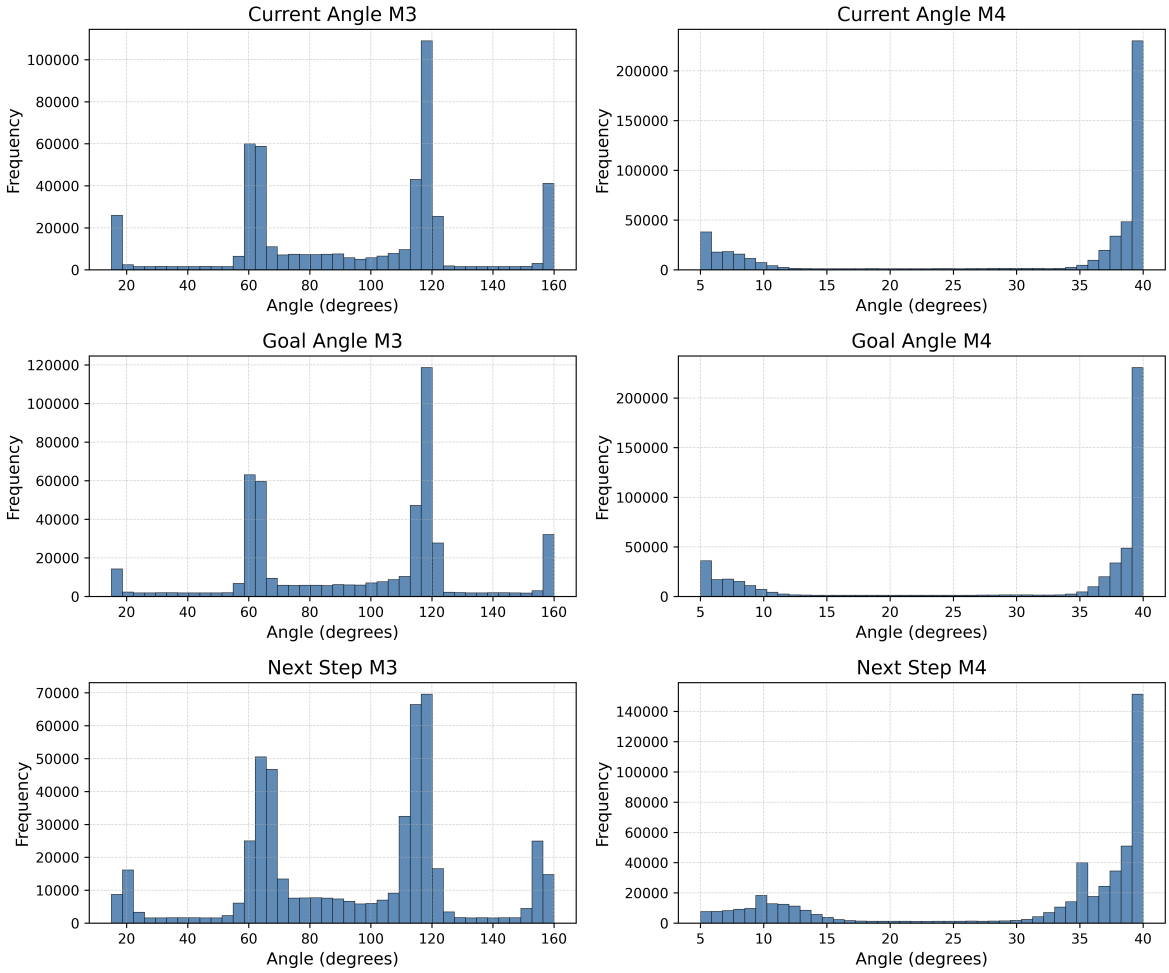

Figure 3. Distribution of current, goal, and next-step angles for motors M3 and M4.

A statistical summary of the dataset is provided in Table 1, which shows the mean and standard deviation for each motor across all three vector types. These statistics confirm that the dataset covers the entire operational workspace while maintaining physically consistent transitions.

Table 1. Summary statistics (mean and standard deviation) of the synthetic dataset used for SNN training. The minimum and maximum values align with the physical joint limits reported in Table 3.

Joint	Type	Mean (deg)	Std (deg)
M1 (Base)	Current	42.18	20.91
	Goal	42.18	20.91
	Next Step	42.20	20.90
M2 (Shoulder)	Current	49.65	14.48
	Goal	49.65	14.48
	Next Step	49.65	14.47
M3 (Elbow)	Current	100.28	6.18
	Goal	100.28	6.18
	Next Step	100.28	6.18
M4 (Gripper)	Current	17.51	7.10
	Goal	17.51	7.10
	Next Step	17.51	7.10

 doi:10.20944/preprints202512.1045.v1

The next-step vector for each sample is calculated via a proportional update rule, where each joint moves a small step toward the target configuration. The step size depends on the magnitude of the joint error but is capped by a maximum limit specific to each joint. Motor 2 again has a more restrictive cap to avoid sudden vertical movements that could destabilize the manipulator or cause unrealistic configurations.

To ensure that the SNN observes enough examples in the most error-prone regions, a curriculum-inspired balancing strategy was used. Approximately 10% of the samples focus on extreme values of motor 2, 5% represent workspace boundary cases, and additional subsets cover mid-range elevation movements, microadjustments, and edge interactions of motors 1 and 3. This ensures that the network encounters diverse scenarios and develops stable generalization behavior.

Finally, all values were normalized to the [0, 1] range and divided into 80% for training, 10% for validation, and 10% for testing [27]. For clarity and reproducibility, Algorithm 1 summarizes the pipeline used to generate the synthetic dataset.

Algorithm 1 Synthetic Dataset Generation (Simplified)

Load pose library $\mathcal{P}$ from simulated trajectories and CSV files.
Determine joint limits $\theta_i^{\min}$ and $\theta_i^{\max}$.
for $k = 1$ to 500,000 **do**
    Sample pair $(\Theta_t, \Theta^{goal})$ following balancing rules.
    Apply Gaussian noise: $\tilde{\Theta} = \Theta + \mathcal{N}(0, \sigma)$.
    Clip angles to joint limits.
    Compute next-step Θ_{t+1} with proportional update and step caps.
    Store $(\Theta_t, \Theta^{goal}, \Theta_{t+1})$.
end for
Normalize dataset to $[0, 1]$.
Partition into train/validation/test.

3.4. Integrated System Architecture

The complete architecture of the proposed neuromorphic control system is shown in Figure 4 and consists of five main stages. In the Data Acquisition stage, the Arduino microcontroller and Serv-Arm manipulator record joint movements and operational limits, stored in CSV files to ensure the SNN operates within physical constraints. The Preprocessing stage normalizes all collected data and performs hyperparameter optimization via a grid search, varying hidden layer size, learning rate, and membrane decay factor. The best configuration is saved in a JSON file.

During the Learning stage, the SNN is trained using Leaky Integrate-and-Fire (LIF) neurons, where inputs are encoded as spike trains representing current and target joint configurations. Synaptic weights are optimized using surrogate gradient descent, and the trained model is exported as a `.pth` file.

The Decoding stage converts spike-based outputs into continuous motor commands using spike rate averaging or first-spike latency decoding. These decoded values are denormalized to actual motor ranges (e.g., 0°–180°). Finally, in the Motion Execution stage, the Raspberry Pi sends the denormalized commands to the Arduino via serial communication, where PWM signals drive the servo motors to execute tasks such as pick-and-place.

 doi:10.20944/preprints202512.1045.v1

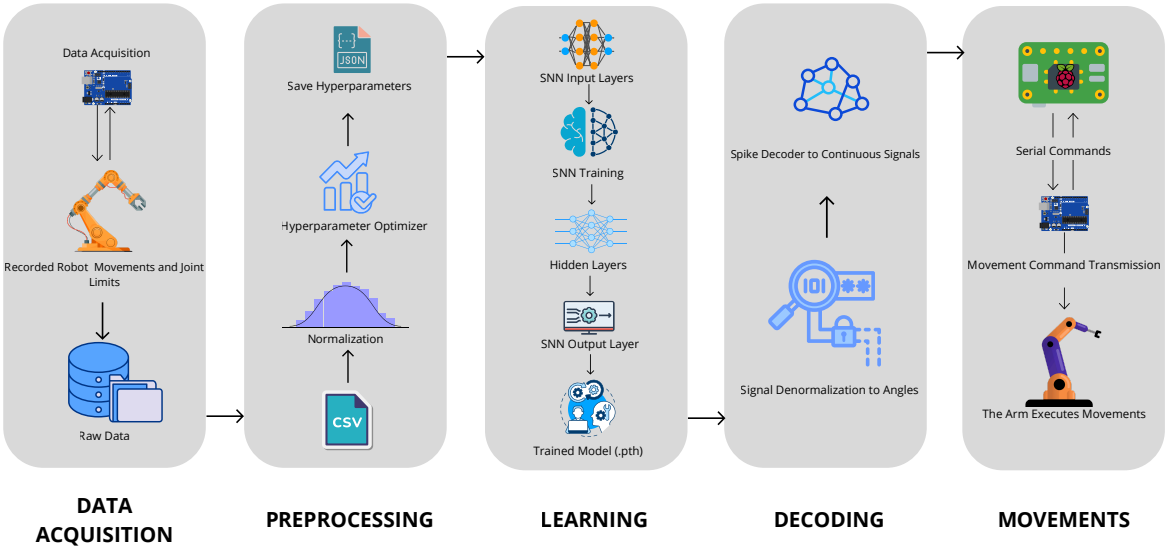

Figure 4. Integrated architecture of the proposed neuromorphic control system.

3.5. Energy Measurement

Assessing the energy consumption of neural controllers on embedded hardware is crucial for determining their suitability for real-time robotic tasks, especially on affordable platforms such as the Raspberry Pi. One of the main reasons for using Spiking Neural Networks (SNNs) is their potential energy efficiency compared with traditional artificial neural networks (ANNs); therefore, a specific and reproducible measurement process was included in the experimental setup. The goal was to measure, under the same operating conditions, the electrical demand of both controllers during the pick-and-place task.

To obtain accurate current measurements without altering the internal circuitry of the Raspberry Pi 4B, an external Hall-effect sensor (ACS712-20A) was connected in series with the power supply line. This method follows established low-cost measurement practices, where external current-sensing modules or shunt-based techniques are commonly used to capture real-time power traces in embedded systems. The ACS712 converts the load current into an analog voltage proportional to the instantaneous current, which an Arduino Uno R3 samples through its 10-bit analog-to-digital converter (ADC). The Arduino performs basic filtering and sends the processed values to the Raspberry Pi, where current, power, and energy logs are synchronized with the robot's motion data.

During operation, each batch of 150 ADC readings is averaged to reduce high-frequency noise, a recommended technique when using low-cost Hall-effect sensors. The instantaneous current $I(t)$ is estimated via the nominal sensitivity of the ACS712-20A (100 mV/A), and readings below 0.1 A are suppressed through a deadband filter to reduce offset noise. The instantaneous electrical power is then calculated assuming a constant 5 V supply according to:

$$P(t) = V \cdot I(t) \tag{2}$$

where $P(t)$ denotes the instantaneous power, V is the supply voltage (fixed at 5 V), and $I(t)$ is the measured current. The mean power over a trial is obtained by:

$$\bar{P} = \frac{1}{K} \sum_{k=1}^{K} P(t_k) \tag{3}$$

and the total energy consumed during each pick-and-place operation is given by:

$$E = \bar{P} \cdot t \tag{4}$$

where E is the total energy (in Joules) and t is the duration of the manipulation routine.

 doi:10.20944/preprints202512.1045.v1

This methodology aligns with established neuromorphic hardware evaluation practices, where energy consumption is calculated by integrating or averaging power over the execution interval [11,13]. To ensure reproducibility, all experiments were repeated multiple times for both controllers. Repetition reduces sensor noise, ADC quantization effects, and runtime variability on the Raspberry Pi 4B—factors known to affect low-cost embedded energy measurements [22]. All trials were conducted under identical electrical, mechanical, and software conditions to guarantee consistent comparisons.

A high-level overview of the acquisition and processing pipeline is given in Algorithm 2, which describes the entire process from sensor calibration to real-time sampling and final task energy calculation.

Algorithm 2 Energy Measurement Procedure Using the ACS712-20A Sensor

Initialization:
Set sensor parameters ($V_{CC} = 5\,\text{V}$; ADC resolution 1023; sensitivity $S = 100\,\text{mV/A}$).
Initialize serial communication at 9600 baud.

Offset Calibration (no load):
Collect $N = 500$ ADC samples.
Compute zero-current offset V_{offset}.

Real-Time Sampling:
for each sampling interval **do**
    Read 150 ADC samples and compute moving average V_{out}.
    Estimate current $I(t)$ using V_{offset} and sensitivity S.
    Apply deadband: if $|I(t)| < 0.1\,\text{A}$, set $I(t) = 0$.
    Compute power $P(t) = 5.0 \cdot |I(t)|$.
    Send $(I(t), P(t))$ to the Raspberry Pi.
end for

Energy Computation (Raspberry Pi):
Compute mean power $\bar{P}$.
Measure task duration t.
Compute energy $E = \bar{P} \cdot t$.

3.6. Spiking Neural Network (SNN) Architecture and Optimization

The spiking neural network (SNN) developed in this work was designed to process trajectory data and generate control commands for the Serv-Arm robot. Its architecture follows a feedforward model with input, hidden, and output layers, where neurons utilize the leaky integrate-and-fire (LIF) model with parameters optimized for the task.

The input layer receives encoded data that represent the robot's current position and, optionally, the target position. These data are transformed into spike trains via temporal encoding methods such as latency or rate coding, enabling the network to capture the spatiotemporal dynamics of movements. After training, the SNN operates in an event-driven manner: neurons integrate incoming signals and fire spikes, transmitting activity across layers. The output layer produces spike trains that are decoded into continuous values representing the adjustment commands for the Serv-Arm servo motors.

The optimal number of neurons and other hyperparameters were selected via a grid search, and variables such as the hidden layer size, learning rate, and membrane potential decay factor (β) were evaluated. Each setup was trained and evaluated on the basis of validation loss and mean absolute error (MAE).

4. Results

4.1. Contributions and Approach Differentials

Several previous studies have examined the use of machine learning and neuromorphic control techniques for robotic manipulators, focusing on energy efficiency and generalization to embedded

 doi:10.20944/preprints202512.1045.v1

systems. Classical controllers, such as PID and adaptive types, continue to be the industry standard for manipulator control. However, these methods have well-known limitations, especially in situations with dynamic uncertainties and incomplete system models, where slight changes in load or friction can significantly reduce performance [1].

Recent research has introduced artificial neural networks (ANNs) to approximate nonlinear control functions in robotics, achieving greater adaptability but at a high computational cost. Other studies expand on this by emphasizing the potential of Spiking Neural Networks (SNNs) as natural successors to ANNs, given their event-driven design and low energy consumption. These works demonstrate that SNNs can produce results comparable to those of traditional methods in trajectory-tracking tasks, although most approaches remain limited to simulations or proprietary neuromorphic hardware platforms, such as Intel Loihi or SpiNNaker, which limits reproducibility and real-world application [3,14,28].

In [15], a neuromorphic control architecture based on SNNs was proposed for a four-degree-of-freedom robotic arm, showing superior energy efficiency and good temporal stability. However, the study concentrates on computational performance analysis rather than physical validation on accessible robots, and relies on specialized hardware for real-time execution. Other works explore the training of spiking networks but depend on limited datasets and lack a systematic methodology for sample generation and balancing [14].

This work differs by proposing a neuromorphic control system built entirely from low-cost, widely available hardware, combining a Raspberry Pi 4B, an Arduino Uno R3, and a Serv-Arm manipulator, without requiring dedicated neuromorphic platforms. Additionally, the synthetic data generation method developed here produces a dataset with over 500,000 balanced samples spanning the robot's entire workspace, enabling the SNN to learn spatiotemporal relationships in a broader and more robust manner [26,27].

Another distinctive feature is systematic hyperparameter optimization via grid search, which helps identify the best configurations for the architecture, learning rate, and membrane decay factor. This step, absent from many prior studies, enhances transparency and scientific reproducibility. Finally, while previous work has often been limited to simulation analysis, this paper experimentally verifies the approach on a complete pick-and-place task, demonstrating the practical feasibility of a neuromorphic controller operating in real time with accuracy comparable to that of traditional techniques [3,28].

Therefore, this study advances the field by demonstrating that satisfactory performance and computational efficiency can be achieved in a fully reproducible neuromorphic robotic system without relying on proprietary hardware. The combination of synthetic data generation, hyperparameter optimization, and physical execution of the model positions this work as an accessible and effective alternative for intelligent control of robotic manipulators [3,14,15].

4.2. SNN Training and Results

Training the spiking neural network (SNN) was preceded by a hyperparameter optimization stage that used a grid search with 54 different configurations. The variables examined included the hidden layer size, learning rate (LR), and membrane potential decay factor (β). Model performance was assessed via validation loss and the mean absolute error (MAE), with early stopping employed to prevent overfitting. Table 2 shows the best results achieved.

 doi:10.20944/preprints202512.1045.v1

Table 2. Summary of the top-performing models from the hyperparameter search.

ID	Hidden	LR	β	Epochs	Val Loss	MAE	Time (min)
1	128	1×10^{-3}	0.7	94	0.0291	11.06	11.68
2	512	1×10^{-3}	0.7	76	0.0428	17.34	10.26
3	1024	1×10^{-3}	0.7	103	0.0341	14.57	23.94
4	2048	1×10^{-4}	0.7	94	0.0432	17.09	49.94
5	512	1×10^{-4}	0.9	147	0.0487	18.94	20.03

The final chosen model features a 128-neuron hidden layer, a learning rate of 1×10^{-3}, and $\beta = 0.7$, achieving an MAE of 11.06° and a validation loss of 0.0291. The Adam optimizer and a weighted mean squared error (MSE) loss were used for training to address dataset imbalance. Synaptic weight updates were performed via surrogate gradient descent, a technique that addresses the nondifferentiability of the neuron firing function.

Model regularization employs early stopping, halting training after 15 epochs without improvement in the validation MAE. To conserve computational resources, the model was compiled, and the batch size was dynamically adjusted based on GPU capacity. Table 3 summarizes the implementation specifications for both training and real-time execution of the SNN model on the robotic platform.

Table 3. Specifications for training, inference, and robotic execution.

Category	Specification	Details
Training (PC)		
Operating System	Ubuntu (x86_64)	—
CPU	Intel Core i7-14650HX	16C/24T, up to 5.2 GHz
GPU	NVIDIA RTX 4060 Laptop	8 GB VRAM
CUDA / Driver	CUDA 12.4	Driver 550.163.01
Training Time	11.68 min	—
Inference / Execution (Robot)		
Inference Platform	Raspberry Pi 4B	Linux
Controller	Arduino Uno R3	—
Communication	UART (RX/TX)	9600 baud
Commands Sent	4 motor angles	Integer degrees
Real Limits per Motor (Degrees)		
Motor 1 (Base)	0° to 90°	
Motor 2 (Shoulder)	10° to 90°	
Motor 3 (Elbow)	90° to 110°	
Motor 4 (Gripper)	5° to 30°	

4.3. Performance Analysis

System performance was assessed by examining both prediction behavior and internal SNN activity. The evolution of the membrane potential in the output neurons is displayed in Figure 5, enabling analysis of the firing dynamics and the stability of the network's responses throughout the temporal encoding window.

The variation among the four output neurons shows that each responds differently to the input pattern, which corresponds to the mapping from the predicted angles to the robot's different servo motors. Membrane potential analysis verified network stability during task execution and motor command consistency, demonstrating that the SNN maintains dynamics consistent with the manipulator's control requirements.

 doi:10.20944/preprints202512.1045.v1

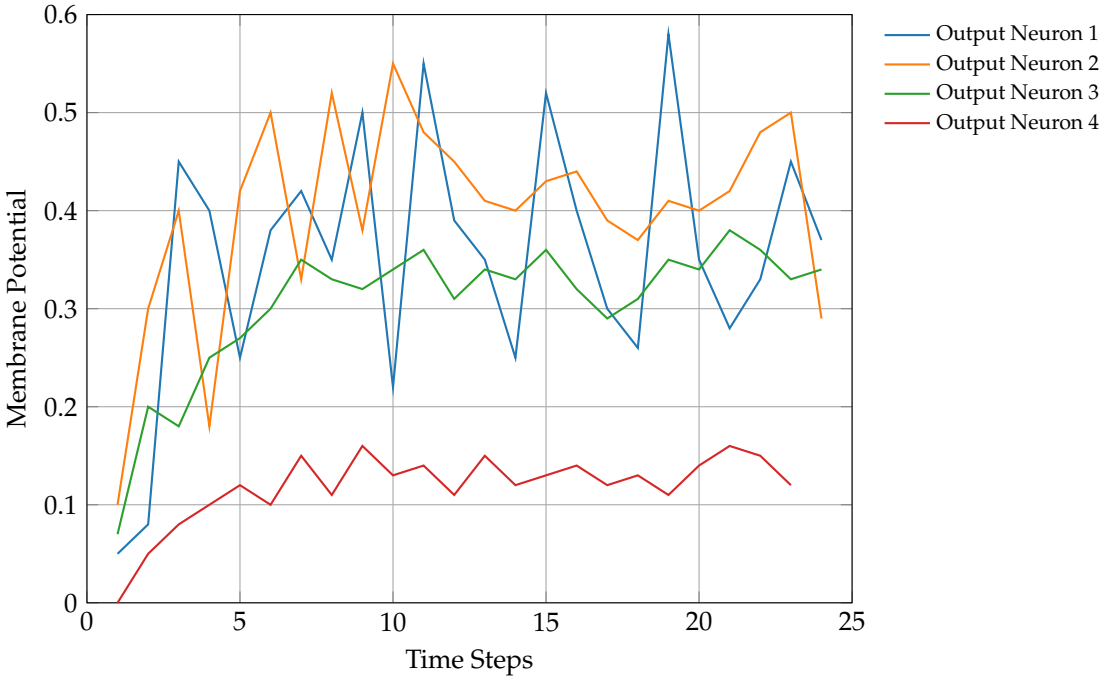

Figure 5. Membrane potential changes of the four output neurons over 25 time steps.

4.4. Learning Curves

During SNN training, learning dynamics were tracked through the training and validation loss curves (Figure 6). Both curves show a steady decrease during the initial epochs, indicating stable optimization and effective convergence of the surrogate-gradient learning process. The close match between training and validation losses throughout the entire training process suggests strong generalization and shows that the model did not overfit the training data. After approximately 60 epochs, minor fluctuations in the validation loss appear, as expected, due to the stochastic nature of the synthetic dataset and the noise added during sample creation. These oscillations remain small and stable, not affecting the overall convergence trend, confirming that the chosen hyperparameters (hidden layer with 128 neurons, learning rate of 10^{-3}, and decay factor $\beta = 0.7$) produce a well-behaved model with consistent learning stability.

 doi:10.20944/preprints202512.1045.v1

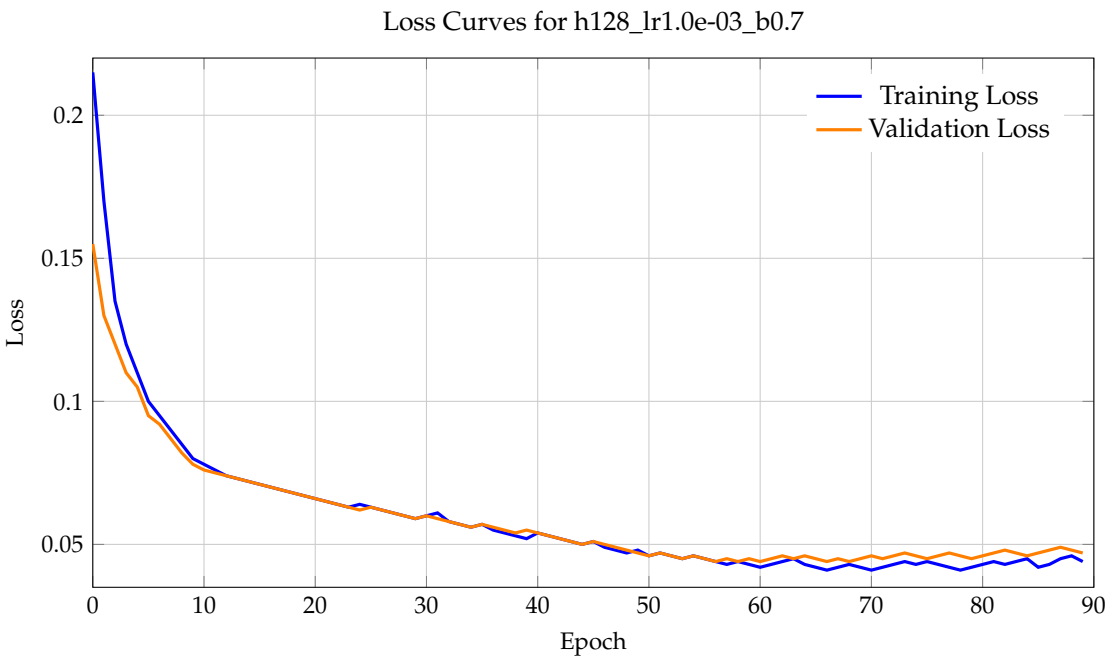

Figure 6. Training and validation loss curves for the optimal SNN setup (hidden layer of 128 neurons, learning rate of 1×10^{-3}, and membrane decay factor $\beta = 0.7$).

4.5. Per-Joint Quantitative Evaluation

Figure 7 shows the distribution of the mean absolute error (MAE) per joint on the test set. The base and elbow joints have low median errors, but some outliers reach higher values, indicating isolated cases with more complex trajectories. The shoulder joint shows the best dispersion, confirming that vertical arm movement is the most challenging movement for the network. The gripper joint has the lowest variability and the smallest absolute errors, suggesting that the SNN controls this actuator with high consistency.

Although some outliers with high absolute errors are observed, they occur at boundary configurations of the robot's workspace where kinematic nonlinearities are more significant. In the central operating areas, the network shows low and consistent errors across all joints. This indicates that the SNN generalizes well under normal conditions, whereas extreme configurations lead to larger deviations due to the mechanical and geometric constraints of the Serv-Arm.

 doi:10.20944/preprints202512.1045.v1

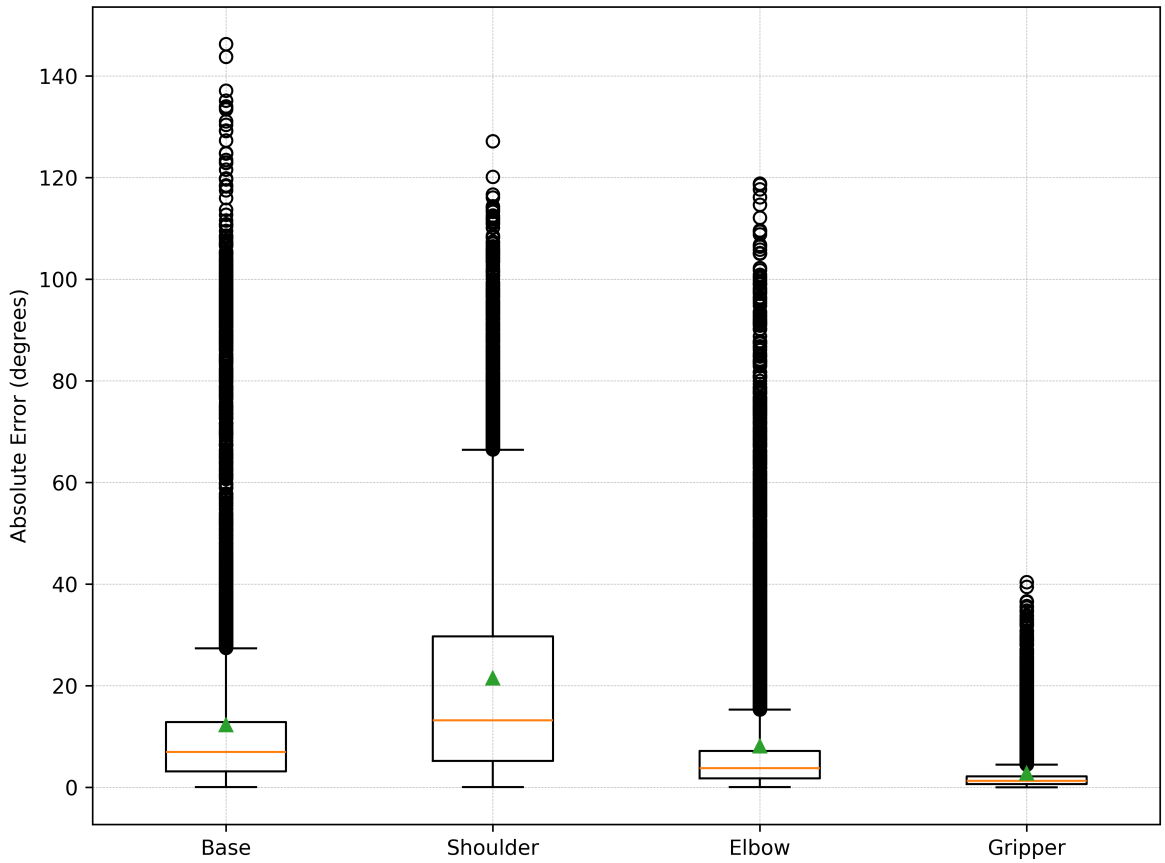

Figure 7. Distribution of the mean absolute error (MAE) for each joint in the test set.

4.6. Energy Consumption Evaluation

The energy efficiency of both controllers was evaluated during the actual execution of the pick-and-place routine using a Raspberry Pi 4B as the inference platform. Unlike many previous studies that rely on smart sockets or software estimators, this work uses a dedicated measurement system based on the ACS712-20A Hall-effect current sensor connected to an Arduino Uno R3, enabling continuous recording of instantaneous voltage, current, and electrical power throughout each experiment.

ACS712 measures the current magnetically, providing galvanic isolation and stable readings even during rapid power fluctuations. To ensure reproducibility, the sensor was calibrated with a known resistive load and offset correction. During each run, the Arduino streamed the formatted measurement packets containing *Vout*, the estimated current, and the instantaneous power over the serial interface to a Python script, which recorded all samples in CSV format for statistical analysis.

Three independent trials were conducted for each controller (SNN and ANN) under precisely identical electrical, mechanical, and software conditions. Each trial involved booting the inference platform, launching the respective controller, and recording power consumption until the pick-and-place task was finished. All measurements were then processed to calculate the average power for each run.

Figure 8 displays the distribution of average power consumption over the three repetitions for each model. The ANN baseline showed an average power consumption of approximately 4.36 W, with minimal variation across trials. In contrast, the SNN used approximately 4.42 W on average. Although the difference is slight—just a few tens of milliwatts—it remains consistent across all runs.

 doi:10.20944/preprints202512.1045.v1

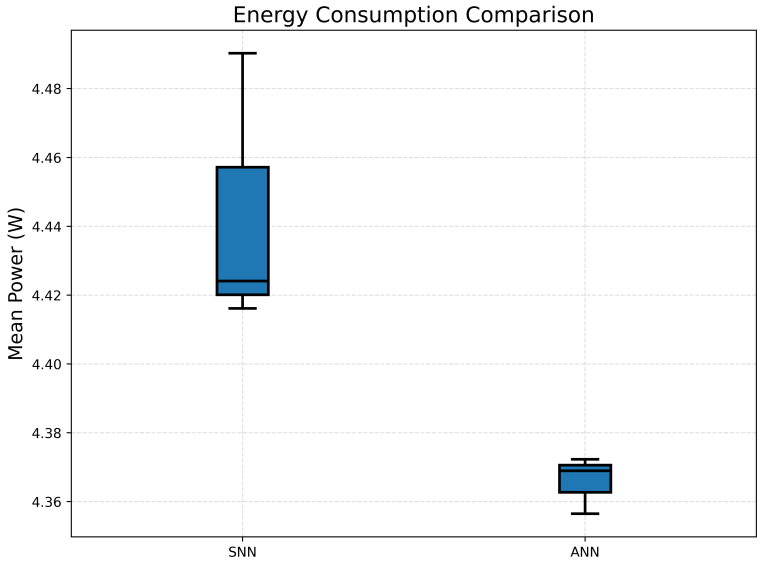

Figure 8. Average power consumption across three experimental trials for each controller. The ANN baseline results in slightly lower, more consistent energy usage, whereas the SNN results in a modest increase due to temporal simulation overhead.

This behavior aligns with previous studies assessing SNN performance on traditional von Neumann hardware [22,23]. Unlike dense ANNs, which perform a single forward pass per control cycle, the SNN in this work uses temporal unrolling over 25 internal simulation steps to mimic membrane dynamics and spike-based information flow. This increases the number of arithmetic operations and memory accesses per prediction, resulting in a slightly higher CPU workload and, consequently, greater power consumption.

Notably, this overhead is not inherent to SNNs themselves but results from simulating spike dynamics on hardware primarily designed for synchronous computation. When used on neuromorphic processors such as Intel Loihi or ODIN, SNNs take advantage of sparse, event-driven communication and localized memory access, leading to energy reductions of one to three orders of magnitude compared with CPU/GPU execution [11,13]. Therefore, the results presented here serve as a software-based execution baseline rather than indicating the actual energy efficiency potential of the neuromorphic model.

Although the Raspberry Pi 4B is not optimized for spike-based computation, the measurements obtained here are still valuable for two reasons. First, they quantify the computational overhead of running an SNN controller on a widely accessible embedded platform. Second, they provide a reproducible reference point for future evaluations of dedicated neuromorphic hardware, where significant energy savings are expected due to event-driven computation and massively parallel spike processing.

Overall, although the ANN used slightly less energy during CPU execution, the differences are minor and due to the extra temporal simulation needed by the SNN. These results align with expectations for software-based SNN inference on traditional processors and emphasize the need to deploy optimized neuromorphic architectures to fully harness spike-based energy savings.

4.7. Joint-Level Error Distribution

System performance was assessed on the basis of trajectory accuracy and final positioning. To gain a clearer understanding of the model's behavior, the distribution of prediction errors across all joints was examined. Figure 9 shows the histogram of errors calculated from 10,000 samples in the test set.

 doi:10.20944/preprints202512.1045.v1

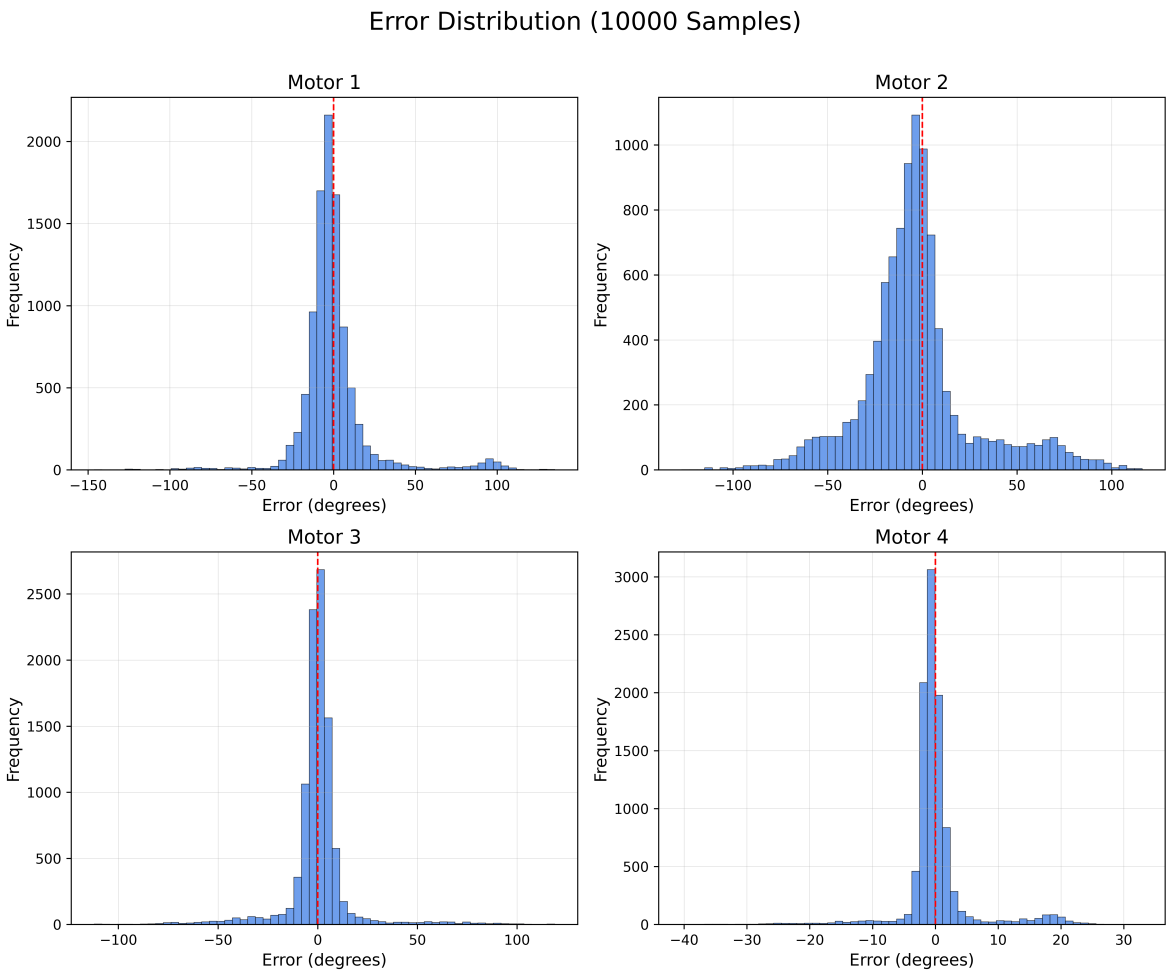

Figure 9. Distribution of prediction errors for the four joints on the basis of 10,000 test set samples. The dashed line indicates a zero-error reference.

The error distribution across 10,000 test samples shows that all joints have predictions concentrated near zero, indicating that the SNN learns a stable and unbiased mapping from inputs to motor commands. The Base (M1) and Elbow (M3) joints exhibit narrow distributions, confirming the accuracy of mid-range lateral movement modeling. The Shoulder joint (M2) presents the greatest variability, which is consistent with its mechanical role in vertical elevation, where small angular variations result in larger end-effector deviations due to nonlinear kinematics. The Gripper (M4) displays the tightest error distribution, reflecting its limited range of motion and simpler kinematic structure. Overall, these results confirm the SNN's ability to generalize across most of the manipulator's workspace, with larger deviations occurring only in mechanically challenging configurations.

5. Discussion

The results of this study show that a neuromorphic controller based on spiking neural networks (SNNs) can provide stable, consistent joint-angle predictions for a low-cost robotic manipulator. The optimized model reached a mean absolute error (MAE) of approximately 11°. While this accuracy is not on par with that of high-precision industrial manipulators, it aligns well with the mechanical limits of the service arm platform, the embedded hardware used, and the open-loop control strategy in this work.

A jointwise analysis shows a clear mechanical link between the model's prediction errors and the robot's structure. The shoulder joint (M2), which controls vertical movement, displayed the highest variability. This is expected: minor angular deviations at this joint lead to larger displacements at the end effector owing to geometric nonlinearities and lever–arm effects. Conversely, the gripper joint

 doi:10.20944/preprints202512.1045.v1

(M4) showed the least variation, reflecting its limited mobility and minimal impact on the overall kinematics. These results suggest that part of the error comes not from learning limitations but from the robot's mechanical constraints.

SNNs provide structural benefits over traditional artificial neural networks (ANNs). Although ANN baselines may achieve higher numerical accuracy, they often rely on dense activation patterns and continuous computations, which can increase computational costs on embedded platforms. SNNs, however, leverage temporal dynamics, sparse event-driven communication, and biologically inspired processing methods that are theoretically better suited to low-power execution. The use of Gaussian noise and dataset balancing during synthetic sample creation may have further enhanced the robustness of the SNN controller.

Importantly, the SNN trained only on synthetic data was successfully generalized to a real robotic system. Despite differences between the simulation and physical conditions—such as servo backlash, asymmetric friction, mechanical tolerances, and UART communication delays—the model successfully executed the whole pick-and-place routine. This domain-transfer ability shows that the synthetic data generation method was sufficiently expressive to capture the key kinematic relationships of the Serv-Arm and that the neuromorphic controller is robust to real-world noise and imperfections.

An energy consumption analysis provides further insight into the practical performance of the neuromorphic controller. Although SNNs are generally linked to energy efficiency, this trait is fully realized only on neuromorphic processors that enable sparse, event-driven computation. On a traditional von Neumann processor such as the Raspberry Pi 4B, the SNN must be simulated via temporal unrolling—25 internal simulation steps per inference—resulting in increased arithmetic operations and memory access. As a result, the SNN uses slightly more power (≈ 4.42 W) than the ANN baseline (≈ 4.36 W). This slight but consistent difference aligns with expectations for software-based SNN execution. This reflects not the intrinsic efficiency of the spiking model but the cost of simulating its dynamics on general-purpose hardware.

Despite these limitations, the results demonstrate the feasibility of deploying SNN-based controllers on readily available embedded platforms and highlight the need for future work to integrate the system with dedicated neuromorphic processors such as Intel Loihi or ODIN. Compared with CPU execution, such hardware can leverage sparse spike-based communication, local memory, and asynchronous processing, enabling reductions of one to three orders of magnitude in energy consumption.

In summary, the proposed neuromorphic controller shows that biologically inspired models can be a practical, robust, and computationally efficient option for low-cost robotic manipulators. The approach eliminates the need for complex analytical models or high-end hardware, offers reproducible learning dynamics, and sets the stage for future applications on neuromorphic architectures where the full benefits of spike-based computation can be utilized.

6. Conclusions

This work introduces a complete neuromorphic control system for a four-degree-of-freedom robotic arm, implemented via a spiking neural network (SNN) and fully deployed on affordable, widely available hardware. By integrating large-scale synthetic data generation, systematic hyperparameter tuning, and physical testing on the Serv-Arm platform, the proposed controller achieved stable, consistent joint-angle predictions across most of the robot's workspace.

The optimized SNN, which includes a 128-neuron hidden layer, a learning rate of 1×10^{-3}, and a membrane decay factor $\beta = 0.7$, achieved an average prediction error of approximately 11°. Jointwise evaluation revealed that most errors stem from the manipulator's mechanical properties, especially at the shoulder joint (M2), where minor angular deviations amplify end-effector motion owing to geometric nonlinearities. These findings suggest that the SNN learned the robot's underlying kinematic structure even when trained solely on synthetic samples.

A key contribution of this study is that neuromorphic controllers can operate reliably on embedded platforms such as the Raspberry Pi 4B and Arduino Uno R3 without specialized neuromorphic

 doi:10.20944/preprints202512.1045.v1

hardware. The energy consumption analysis also revealed that when run as software simulations on a standard CPU, the SNN uses slightly more power than the ANN baseline does. This slight increase is fully accounted for by the temporal unrolling required to simulate spike-based dynamics and does not reflect the inherent efficiency of neuromorphic computation. Instead, these results provide a practical performance baseline for future deployment on dedicated neuromorphic processors, where sparse, event-driven computations can yield significant energy savings.

Future work will focus on increasing task complexity, exploring deeper or multilayer SNN architectures, adopting biologically inspired or population-based encoding schemes, and integrating closed-loop control to reduce accumulated drift in mechanically challenging regions of the workspace. Additionally, evaluating the approach on an industrial-grade robotic manipulator will enable testing under more demanding precision and robustness requirements.

Overall, the findings confirm the viability of neuromorphic control for low-cost robotic manipulators and emphasize the potential of SNNs as a biologically inspired and embedded-friendly alternative to traditional ANN- or PID-based controllers. The approach presented here, from synthetic dataset creation to real-world testing, offers a reproducible and accessible framework for advancing neuromorphic robotics research.

Author Contributions: Conceptualization, A.P.O.B. and V.J.S.; Methodology, A.P.O.B.; Software, A.P.O.B.; Validation, A.P.O.B., R.C.S.A. and G.M.S.; Formal analysis, A.P.O.B.; Investigation, A.P.O.B.; Resources, C.A.O.F. and V.J.S.; Data curation, A.P.O.B.; Writing—original draft preparation, A.P.O.B.; Writing—review and editing, V.F.L.J. and V.J.S.; Visualization, A.P.O.B.; Supervision, V.F.L.J. and V.J.S.; Project administration, V.J.S.; Funding acquisition, P.V.S.M.

Funding: As provided for in Arts. 21 and 22 of Decree No. 10.521/2020, this work was partially financed by Motorola Mobility Comércio de Produtos Eletrônicos Ltda and Flextronics da Amazônia Ltda, under Federal Law No. 8.387/1991 and Agreement No. 008/2024, through the Research, Development, and Technological Innovation Project (PD&I) "Mob4AI – Advanced Artificial Intelligence for Neuromorphic Mobile System."

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Acknowledgments: The authors thank the Motorola Mobility/UFAM R&D project (MOB4AI) and the NAVIR Research Center (Center for Automation, Computer Vision, Artificial Intelligence, and Robotics) for providing infrastructure and support.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

SNN	Spiking Neural Network
LIF	Leaky Integrate-and-Fire
DoF	Degrees of Freedom
PID	Proportional–Integral–Derivative
MAE	Mean Absolute Error
MSE	Mean Squared Error
STDP	Spike-Timing-Dependent Plasticity
PWM	Pulse Width Modulation

References

1. Siciliano, B.; Sciavicco, L.; Villani, L.; Oriolo, G. *Robotics: Modelling, Planning and Control*; Springer, 2009. https://doi.org/10.1007/978-1-84628-642-1.
2. Maass, W. Networks of spiking neurons: The third generation of neural network models. *Neural Netw.* **1997**, *10*, 1659–1671. https://doi.org/10.1016/S0893-6080(97)00011-7.

 doi:10.20944/preprints202512.1045.v1

3. Schuman, C.D.; Potok, T.E.; Patton, R.M.; Birdwell, J.D.; Dean, M.E.; Rose, G.S.; Plank, J.S. A survey of neuromorphic computing and neural networks in hardware. *arXiv preprint* **2017**, [1705.06963].
4. Tavanaei, A.; Ghodrati, M.; Kheradpisheh, S.R.; Masquelier, T.; Maida, A. Deep learning in spiking neural networks. *Neural Netw.* **2019**, *111*, 47–63. https://doi.org/10.1016/j.neunet.2018.12.002.
5. Davies, M.; Srinivasa, N.; Lin, T.H.; Chinya, G.; Cao, Y.; Choday, S.H.; Dimou, G.; Joshi, P.; Imam, N.; Jain, S.; et al. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro* **2018**, *38*, 82–99. https://doi.org/10.1109/MM.2018.112130359.
6. Akopyan, F.; Sawada, J.; Cassidy, A.; Alvarez-Icaza, R.; Arthur, J.; Merolla, P.; Imam, N.; Nakamura, Y.; Datta, P.; Nam, G.J.; et al. TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **2015**, *34*, 1537–1557. https://doi.org/10.1109/TCAD.2015.2474396.
7. Trujillo, D.; Morales, L.A.; Chávez, D.; Pozo, D.F. Trajectory tracking control of a mobile robot using neural networks. *Emerg. Sci. J.* **2023**, *7*, 1843–1862. https://doi.org/10.28991/ESJ-2023-07-06-01.
8. Wang, Z.; Zhao, J.; Yang, J.; Wang, Y.; Xiao, X.; Li, Y.; Xiao, C.; Wang, L. Embodied neuromorphic control applied on a 7-DOF robotic manipulator. *arXiv preprint* **2025**, [2504.12702].
9. Park, Y.; Lee, J.; Sim, D.; Cho, Y.; Park, C. Designing spiking neural network-based reinforcement learning for 3D robotic arm applications. *Electronics (Basel)* **2025**, *14*, 578. https://doi.org/10.3390/electronics14030578.
10. Zahra, O.; Navarro-Alarcon, D.; Tolu, S. A fully spiking neural control system based on cerebellar predictive learning for sensor-guided robots. In Proceedings of the Proc. IEEE Int. Conf. Robotics and Automation (ICRA), 2021, pp. 4423–4429. https://doi.org/10.1109/ICRA48506.2021.9561127.
11. Davies, M.; Wild, A.; Orchard, G.; Sandamirskaya, Y.; Fonseca Guerra, G.; Joshi, P.; Plank, P.; Risbud, S. Advancing Neuromorphic Computing With Loihi: A Survey of Results and Outlook. *Proceedings of the IEEE* **2021**, *PP*, 1–24. https://doi.org/10.1109/JPROC.2021.3067593.
12. Esser, S.K.; Merolla, P.A.; Arthur, J.V.; Cassidy, A.S.; Appuswamy, R.; Andreopoulos, A.; Berg, D.J.; McKinstry, J.L.; Melano, T.; Barch, D.R.; et al. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences* **2016**, *113*, 11441–11446, [https://www.pnas.org/doi/pdf/10.1073/pnas.1604850113]. https://doi.org/10.1073/pnas.1604850113.
13. Frenkel, C.; Lefebvre, M.; Legat, J.D.; Bol, D. A 0.086-mm^2 12.7-pJ/SOP 64k-Synapse 256-Neuron Online-Learning Digital Spiking Neuromorphic Processor in 28-nm CMOS. *IEEE Transactions on Biomedical Circuits and Systems* **2019**, *13*, 145–158. https://doi.org/10.1109/TBCAS.2018.2880425.
14. Bouganis, A.; Shanahan, M. Training a spiking neural network to control a 4-DOF robotic arm based on spike timing-dependent plasticity. In Proceedings of the Proc. Int. Joint Conf. Neural Netw. (IJCNN), 2010, pp. 1–8. https://doi.org/10.1109/IJCNN.2010.5596525.
15. Cheng, R.; Mirza, K.B.; Nikolic, K. Neuromorphic robotic platform with visual input, processor and actuator, based on spiking neural networks. *Appl. Syst. Innov.* **2020**, *3*, 28. https://doi.org/10.3390/asi3020028.
16. Yang, G.; Lee, W.; Seo, Y.; Lee, C.; Seok, W.; Park, J.; Sim, D.; Park, C. Unsupervised spiking neural network with dynamic learning of inhibitory neurons. *Sensors (Basel)* **2023**, *23*, 7232. https://doi.org/10.3390/s23167232.
17. Quintanar-Guzmán, S.; Kannan, S.; Aguilera-González, A.; Olivares-Mendez, M.A.; Voos, H. Operational space control of a lightweight robotic arm actuated by shape memory alloy wires: A comparative study. *J. Intell. Mater. Syst. Struct.* **2019**, *30*, 1368–1384. https://doi.org/10.1177/1045389X17721050.
18. Kim, S.; Park, S.; Na, B.; Yoon, S. Spiking-YOLO: Spiking neural network for energy-efficient object detection. In Proceedings of the Proc. AAAI Conf. Artif. Intell., 2020, Vol. 34, pp. 11270–11277.
19. Leroux, N.; Finkbeiner, J.; Neftci, E. Online transformers with spiking neurons for fast prosthetic hand control. In Proceedings of the Proc. IEEE Biomedical Circuits and Systems Conf. (BioCAS), 2023, pp. 1–6. https://doi.org/10.1109/BioCAS58349.2023.10388996.
20. Song, Y.; Cai, S.; Yang, L.; Li, G.; Wu, W.; Xie, L. A practical EEG-based human-machine interface to online control an upper-limb assist robot. *Front. Neurorobot.* **2020**, *14*, 32. https://doi.org/10.3389/fnbot.2020.00032.
21. Scrugli, M.A.; Leone, G.; Busia, P.; Raffo, L.; Meloni, P. Real-time sEMG processing with spiking neural networks on a low-power 5K-LUT FPGA. *IEEE Trans. Biomed. Circuits Syst.* **2024**.
22. Osolinskyi, O.; Sachenko, A.; Kochan, V.; Kolodiichuk, L. Measurement and Optimization Methods of Energy Consumption for Microcontroller Systems Within IoT. In Proceedings of the 2022 12th International Conference on Dependable Systems, Services and Technologies (DESSERT), Dec 2022, pp. 1–7. https://doi.org/10.1109/DESSERT58054.2022.10018631.

 doi:10.20944/preprints202512.1045.v1

23. Carroll, A.; Heiser, G. An analysis of power consumption in a smartphone. In Proceedings of the Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference, USA, 2010; USENIXATC'10, p. 21. https://doi.org/10.5555/1855840.1855861.
24. de Moraes Cruz, C.A.; de Lima Monteiro, D.W.; Cotta, E.A.; de Lucena, V.F.; Souza, A.K.P. FPN Attenuation by Reset-Drain Actuation in the Linear-Logarithmic Active Pixel Sensor. *IEEE Transactions on Circuits and Systems I: Regular Papers* **2014**, *61*, 2825–2833. https://doi.org/10.1109/TCSI.2014.2327284.
25. Schuman, C.D.; Kulkarni, S.R.; Parsa, M.; Mitchell, J.P.; Date, P.; Kay, B. Opportunities for neuromorphic computing algorithms and applications. *Nat. Comput. Sci.* **2022**, *2*, 10–19. See also correction: doi:10.1038/s43588-022-00223-2, https://doi.org/10.1038/s43588-021-00184-y.
26. dawars. Serv-Arm. https://www.thingiverse.com/thing:1684471, 2016. [Online; accessed 2025-10-21].
27. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press, 2016.
28. Bing, Z.; Meschede, C.; Röhrbein, F.; Huang, K.; Knoll, A.C. A survey of robotics control based on learning-inspired spiking neural networks. *Front. Neurorobot.* **2018**, *12*, 35. https://doi.org/10.3389/fnbot.2018.00035.