International Journal of Combinatorial Optimization Problems and Informatics, 13(4), Dec 2022, 36-46. ISSN: 2007-1558.

www.editada.org

Spiking Neural Network implementation of LQR control on underactuated system

J. A. Juárez-Lora[1], Humberto Sossa[1], Victor H. Ponce-Ponce[1], Elsa Rubio-Espino[1], Ricardo Barrón Fernández[1]

[1] Instituto Politecnico Nacional, Centro de Investigación en Computación, Av. Juan de Dios Bátiz, Esq. Miguel Othón de Mendizábal | Col. Nueva Industrial Vallejo, Delegación Gustavo A. Madero | C.P. 07738 | Ciudad de México, México
jjuarezl2020@cic.ipn.mx, hsossa@cic.ipn.mx, vponce@cic.ipn.mx, erubio@cic.ipn.mx, rbarron@cic.ipn.mx

Abstract. Adaptability, learning capabilities, and space-energy efficient hardware are required in robotic architectures, which must deal with changing dynamic environments. Nowadays, learning algorithms are implemented in Von Neumann Architectures, which separate storage from processing units, making them not appropriate for artificial neural networks (ANN), resulting in inefficient implementations. This writing presents a neural architecture proposal designed to implement a control loop in a mobile wheeled under-actuated inverted pendulum system, using spiking neural networks, linear quadratic regulator control technique, and a neural framework that allows us to define the neuron ensembles specification to represent specific control signals. The intention is to study how typical control theory algorithms can be translated into neural structures, aiming for neuromorphic implementation.

Keywords: Robotics, Neural Networks, Spiking Neural Networks, Machine Learning, Neurorobotics, Neurocomputing.

Article Info
Received March 24, 2022
Accepted August 12, 2022

1 Introduction

4.0 Industry has brought a massive proliferation in the usage of sensors and data acquisition devices for monitoring and analysis purposes, leading to an overpass of data recollection. This challenges current computation and storage capacities to provide Big Data process techniques and solutions for intelligent decision making. New analysis and processing techniques, such as artificial neural networks, have proven to be very useful in online learning scenarios [1].

Spiking neural networks (SNN), also known as artificial third-generation neural networks, intend to emulate physical, chemical, and biological mechanisms that enable computation and Hebbian learning ocurring in biological living systems. These models have optimal characteristics for hardware implementation [1,2] as it promises huge parallel computing capacity, in addition to low energetic usage, setting a path towards online learning platforms implementation with size and power restriction applications, such as robotics.

Neurorobotics [3,4] is a discipline that takes as a challenge to design control mechanisms, hardware, and implementation techniques for robotic applications, which must learn to adapt their behavior to changes in themselves or the environment dynamics. This situation can be seen in biological systems with growing limbs, holding a heavy object, alien to its composition, or perhaps aging, which modifies friction in the arm or leg joints.

Due to its analog nature, there is a lack of ease-of-access platforms for neuromorphic computation. At [31], the authors propose hardware architectures, including neurons and memristors, and how these components can be interconnected as an array scheme to achieve large-scale spiking systems using synaptic time-dependant plasticity (STDP). Multiple research architectures proposals have emerged from this article, both analog and digital. At [32], a synaptic array of 4096 memristive devices is proposed. A memristor analog crossbar circuit is used to emulate a single layer perception for the MNIST image classification. In the analog domain, digital efforts such as Loihi [15] or Spinnaker AI hardware accelerators have been created in private research initiatives. As the industry focuses on typical von Neumann Architectures, these platforms are not widely available for acquisition or commercialization.

However, simulation platforms allow academic researchers to design and study neural structures. The Human Brain Project [5] is an organization of researchers who have created a neurorobotics platform [6] that connects a simulated brain to a simulated or physical body, allowing researchers to explore how to control movement, stimuli reaction, and learning from a virtual or natural environment. Another platform primarily focused on SNN implementation is called Nengo [7-9], a tool that allows to build and design SNNs architectures. The user can define neural models, its own learning rules, optimization methods, reuse of subnetworks, and data input, and even has libraries for exporting these models to neuromorphic hardware or FPGA implementations.

A small tour in neurorobotics literature points towards the implementation of primitive tasks. In [10], an adaptive control method proposed in [11] is used, allowing them to control a three-link arm in simulation, using a spiking neural network structure designed to estimate the inverse jacobian dynamics. Here, the authors name part of their proposed neural network as a biological name part in order to match specific tasks made by biological brains. In [12], control of a simulated robotic arm, without path planning, is achieved using SNNs and motor primitives. In [13], a biologically inspired spiking neural network (SNN) for soft-grasping to control a robotic hand, used for robots interacting with objects shaped for humans, is presented. For the last example, in [14], a hardware adaptive control implementation of a Kinova Jaco robotic arm using the Loihi platform [15] is shown. These three examples have something in common; these are complete actuated systems.

In this work, an implementation of a linear quadratic regulator (LQR) control strategy using SNN is presented as an introductory example of how SNNs structures can be used to process control signals, specifically, for under-actuated systems, in order to show which obstacles must be tackled in order to perform precise control signal representation. The Mobile Wheeled Inverted Pendulum (MWIP) is an easily controllable system for a human with a bit of practice but a challenge in control theory. Although some of these under-actuated systems show controllability under linearization around a certain equilibrium point, the control tasks entitle arbitrary output reference trajectory tracking, taking the system state away from the equilibrium point, thus overcoming a traditional obstacle to linearization-based control of nonlinear systems [16]

The structure of the article is described next. Section 2 illustrates the robotic MWIP plant, a typical control loop strategy (LQR) used for stabilization, and its implementation using a neural modeling approach. Section 3 describes simulation parameters, configurations, and results. Section 4 boards conclusion and future work.

2 Materials and Methods

Up next, the control problem is described. First, the model dynamics and a typical control algorithm for the system are presented. Then, to construct a matching SNN, neuron models, encoding/decoding process from the continuous to the spiking domain, and a framework that allows us to interconnect a group of neurons to represent our closed control loop signals is summarized.

2.1 Dynamic model of the plant

Figure 1 shows the graphical representation of the MWIP (Mobile Wheeled Inverted Pendulum). Here, x_w, y_w are the wheel coordinates, x_b, y_b are the mass center coordinates of the bar, α is the plane's angle inclination, m_b, m_w stand for the bar and the wheel's mass, respectively, I_b, I_w are the moments of inertia from the bar and the wheel, L is the bar's length, r is the radius of the wheel, and θ_w, θ_b are the states of the system, which stand for the rotation's angle of the wheel, and the bar's inclination, respectively.

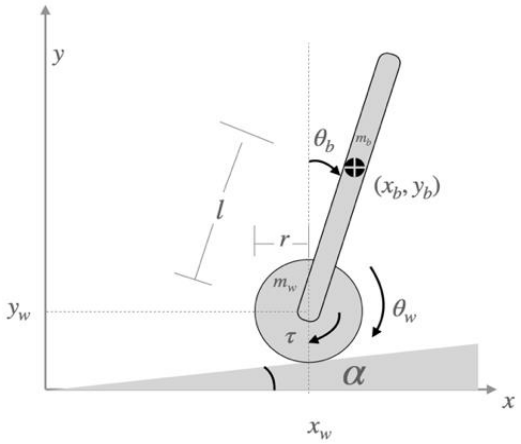

Fig. 1 2D MWIP model, extracted from [17].

The robotic system corresponds to a second-order underactuated system [18]. Starting from the modeling dynamics using Euler - Lagrange technique in [17], lead to a system with the following depiction:

$$\mathrm{M}(\mathrm{q})\ddot{\mathrm{q}} + \mathrm{C}(\mathrm{q}, \dot{\mathrm{q}})\dot{\mathrm{q}} + \mathrm{g}(\mathrm{q}) = \mathrm{Bu} \tag{1}$$

where $\mathrm{M}(\cdot)$ is the inertia matrix, $\mathrm{C}(\cdot)$ groups the Coriolis terms, $\mathrm{g}(\cdot)$ gravity forces and control terms, and B illustrates how a control signal u is fed to the system. For a surface without inclination, $\alpha = 0$, Equation (1) is extended as:

$$\begin{pmatrix} (\mathrm{m_b} + \mathrm{m_w})\mathrm{r}^2 + \mathrm{I_w} & \mathrm{m_b Lr}\cos(\theta_\mathrm{b}) \\ \mathrm{m_b Lr}\cos(\theta_\mathrm{b}) & \mathrm{m_b L}^2 + \mathrm{I_b} \end{pmatrix} \begin{pmatrix} \ddot{\theta}_\mathrm{w} \\ \ddot{\theta}_\mathrm{b} \end{pmatrix} + \begin{pmatrix} 0 & -\mathrm{m_b Lr}\dot{\theta}_\mathrm{b}\sin(\theta_\mathrm{b}) \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \dot{\theta}_\mathrm{w} \\ \dot{\theta}_\mathrm{b} \end{pmatrix} + \begin{pmatrix} 0 \\ -\mathrm{m_b gL}\sin(\theta_\mathrm{b}) \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix} \mathrm{u} \tag{2}$$

At Equation (2), $\mathrm{B} = [1, -1]^\mathrm{T}$ due to the control torque u is equivalent but in opposite direction, as the motor is mounted at the shaft of the wheel, also connected to the bar. In order to obtain the accelerations of the system, we rewrite equation (2) as:

$$\ddot{\mathrm{q}} = \mathrm{M}(\mathrm{q})^{-1}[(\mathrm{Bu} - \mathrm{C}(\mathrm{q}, \dot{\mathrm{q}})\mathrm{q} - \mathrm{g}(\mathrm{q})] \tag{3}$$

Both second order differential equations can be represented in four first-order equations, rewriting the system in a space state manner, by setting $\mathrm{x} = [\mathrm{x_1}, \mathrm{x_2}, \mathrm{x_3}, \mathrm{x_4}] = [\theta_\mathrm{w}, \dot{\theta}_\mathrm{w}, \theta_\mathrm{b}, \dot{\theta}_\mathrm{b}]$. The system can be linearized in its equilibrium states $\theta_\mathrm{b} = \dot{\theta}_\mathrm{w} = \dot{\theta}_\mathrm{b} = 0$, which is equivalent to a pendulum in an upright position. Therefore, the linealized system results in the form $\dot{\mathrm{x}} = \mathrm{Ax} + \mathrm{Bu}$:

$$\mathrm{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{-\mathrm{gL}^2\mathrm{m_b}^2\mathrm{r}}{\mathrm{z}} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\dfrac{\mathrm{gLm_b}^2\mathrm{r}^2 + \mathrm{gLm_w m_b r}^2 + \mathrm{I_w gLm_b}}{\mathrm{z}} & 0 \end{bmatrix} \mathrm{B} = \begin{bmatrix} 0 \\ \dfrac{\mathrm{m_b L}^2 - \mathrm{m_b rL} + \mathrm{I_b}}{\mathrm{z}} \\ 0 \\ -\dfrac{\mathrm{I_w} + \mathrm{m_b r}^2 + \mathrm{m_w r}^2 - \mathrm{Lm_b r}}{\mathrm{z}} \end{bmatrix}$$

With:

$$\mathrm{z} = \mathrm{I_b I_w} + \mathrm{I_w L}^2\mathrm{m_b} + \mathrm{I_b m_b r}^2 + \mathrm{I_b m_w r}^2 + \mathrm{L}^2\mathrm{m_b m_w r}^2$$

It can be easily shown that the system is fully controllable and observable.

2.2 Linear Quadratic Regulator

Given a desired state vector x_d, a control law $u = -K\_r\ (x - x\_d)$ is proposed to achieve a stabilization task. Here, K_r can be chosen to move the closed-loop eigenvalues of the system as far as desired on the left half of the complex plane. The Linear Quadratic Regulator [19] (LQR hereinafter) consists in to deliver full state feedback control law method that minimizes the following cost function:

$$\mathrm{J(t)} = \left(\int_0^\infty (\mathrm{x} - \mathrm{x_d})^{\mathrm{T}}\, \mathrm{Q}(\mathrm{x} - \mathrm{x_d}) + \mathrm{u^T R u}\right)\mathrm{dt} \tag{4}$$

Equation (4) balances the energetic cost of an effective state regulation, which is intended to be low, and a quicker control response, which is intended to be fast. These objectives are regulated by $\mathrm{Q} = \mathrm{Q^T} \geq 0$ and $\mathrm{R} = \mathrm{R^T} \geq 0$ respectively, and they can be selected as wished to prioritize control objectives. As bigger is Q, it will move the system to the desired vector state as soon as possible, this is, closed loop poles would be as far as desired in the left plane, and vice versa. As big as R might be, lower control signals will be priority, while $\mathrm{J} = \lim_{t\to\infty} \mathrm{x(t)} = 0$. As $\mathrm{J}(\cdot)$ is a quadratic function, there exists an analytic solution for control weights in $\mathrm{K_r}$ given by:

$$\mathrm{K_r} = \mathrm{R^{-1}BP} \tag{5}$$

Where P is the Ricatti's algebraic equation solution:

$$\mathrm{PA} + \mathrm{AP^T} - \mathrm{PBR^{-1}B^TP} + \mathrm{Q} = 0 \tag{6}$$

In order to solve equation (6) there are several software implemented methods [20,21], which start from a known A, B for a $\ddot{\mathrm{x}} = \mathrm{f}(\mathrm{q}, \dot{\mathrm{q}}, \mathrm{u})$ system dynamics.

2.3 Neural network Modelling

Diverse models of neuron dynamics are available in the literature, from Hodgin & Huxley model [25], which is biologically plausible but has a higher computational cost of implementation, to simpler yet useful models such as Leaky Integrate and Fire (LIF), based on a simple equivalence between an RC circuit and membrane's voltage [25]. For the instant the switch closes, an input voltage is converted to an input current I_syn by the membrane's resistance $R_m$and fed into a membrane's capacitance C_m, which will integrate its voltage V_m as:

$$\mathrm{R_m C_m}\frac{\mathrm{dV_m}}{\mathrm{dt}} = \mathrm{E_L} - \mathrm{V_m} + \mathrm{R_m I_{syn}} \tag{7}$$

The membrane's voltage accumulates up to a certain threshold value v_{th}. Then, the neuron quickly depolarizes, $V_m = 0$, and a spike is produced. For a given I_{syn} value, the neuron will spike at a different frequency. An input signal x(t), bounded from a minimum $x_{\min}$ to a maximum value $x_{\max}$, defines the amount of current to be provided. This process is called Encoding. Rate-based encoding algorithms interpolate $[x_{\min,} x_{max}]$ with a minimium and maximum spike frequency $[f_{\min,} f_{max}]$ of one neuron [25]. However, other encoding methods, such as population encoding [27], consider *n* neurons in a group, called ensemble, to encode x(t) into the spike domain. Each neuron from the ensemble will be fed in such a way that the input signal domain is equally divided between the n neurons. This is called spike-based sparse coding, and it can be seen in Figure 2. More neurons imply better signal processing, as the signal domain is divided between more neurons. The resulting network structure approximates the original input signal according to neural heterogeneity, stochasticity, and connectivity, which affect its performance [26].

In order to design and implement the neural network, the principles developed in Neural Engineering Framework (NEF) [27,28] are used. NEF can be seen as a 'neural compiler' that guarantees an optimal approximation of the defined dynamic equations by the user-defined ensembles. For a given set of signals $\mathrm{x(t)} \in \mathbb{R}^{\mathrm{q}}$, representing the plant's space state vector of dimension q, an ensemble of n neurons will encode these signals into the spike domain, and then these spikes are used to reconstruct the original signal. This can be seen

in Figure 3. The synaptic weights $W \in \mathbb{R}^{nxn}$ matrix are composed as $W = ED^f$, where $E = [e_I, \dots, e_n]^T \in \mathbb{R}^{n\times q}$ is an encoding matrix and $D^f = [d_1, \dots, d_n]^T \in \mathbb{R}^{q\times n}$ a decoder matrix.

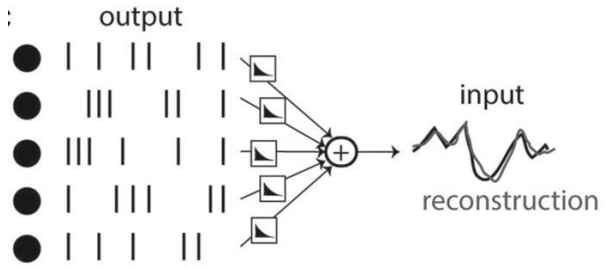

Fig. 2 Spike-based sparse coding. A reconstruction of the signal is obtained from combining filtered spike trains together, and spikes are timed to make the reconstruction accurate. Extracted from [26]

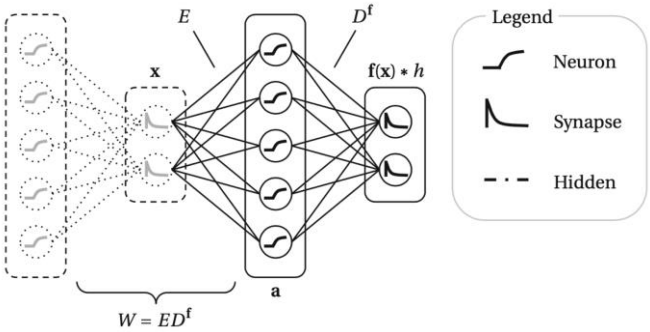

Fig. 3 Neural Engineering Framework: n neurons represent q signals. Synapses then are tuned to properly represent an input signal. Extracted from [7]

The neurons will produce spikes which will modify the weights. The spikes are produced by feeding a current input I_{syn} to each neuron defined by:

$$I_{syn}(t) = \alpha_i < e_i, x(t) > +\beta_i$$
$$a_i(t) = G[I_{syn}] \tag{8}$$

Here $\alpha_i > 0$ are gains, e_I can be seen as the vector value for which the neuron responds with the highest frequency of spikes [30], and is set to be $||e_i||_2 = 1$, $\beta_i (i = 1,2, \dots, n)$ are biases in order to distribute the signal's domain. Equation (7) defines a high-dimensional nonlinear projection of x(t), by taking its dot product with an encoding matrix E and injecting the result as current into n neurons. $G[\cdot]$ represents the neuron dynamics, which describe how spikes are produced for a given input current. In this article, we consider the LIF model from Eq. (7), but other neuron models with fixed v_{th} are usable with NEF[7]. Ensembles are connected through synwhich values must be found to represent the desired signals properly. Spiking activity would perform modifications on D^f selected to reconstruct the original signal $\hat{x}(t)$ from the spiking activity. This can be accomplished by the implementation of learning rules, such as the *Bernerstone-Cooper-Monroe* (BCM henceforth) theory [23,24], which describes how synaptic plasticity on cortical neurons is stabilized by the average postsynaptic activity, or Synaptic Time-Dependant Plasticity (STDP), which sets the change of the synapse value as a function of the pre and postsynaptic spikes [31]. The reconstruction of the signal is performed as:

$$\hat{x}(t) = (x * h)(t) \approx \sum_{i=1}^{N} (a_i * h)(t)\, d_i \tag{9}$$

Where * stands for convolution operation of the spikes with $h(t)$ modeling a synapse conductance as a a low-pass function with time constant τ:

$$h(t) = \frac{1}{\tau} e^{\frac{-t}{\tau}} \tag{10}$$

As, ideally $x(t) = \hat{x}(t)$, synaptic weights can be found as as a least squares optimization problem, under the assumption of knowing the corresponding spiking activity:

$$D^f = \text{argmin}_{D \in R^{q \times n}} \int \left\| f(I_{syn}) - \sum_{\{i=1\}}^{n} (r_{i(v)} + \eta) d_i \right\|_2^2 dv \quad (11)$$

Where η is white gaussian noise with a variance σ^2, and $r_i(v)$ replaces neural activity triggered by an input signal as an average firing rate of the $i-th$ neuron for a given input $I_{syn}(x(t))$, modeled as:

$$r_i(v) = \lim_{t \to \infty} \frac{1}{t} \int_0^t a_i t' dt' \quad (12)$$

Eq. (12) demands *a priori* simulation of the spiking activity to obtain the firing rate of each neuron and, subsequently, the average firing rate of the ensemble. However, solving the LIF differential equation (8) with initial condition parameters allows us to get a function relating current with spike frequency, which can be used to obtain the frequency response of a neuron, and then obtain the average of the ensemble for a given input [33]

3 Experimental results

In this section, a description of the proposed network structure is described, and how was implemented in the specialized software tool. Then, the parameters of the MWIP and the control task are detailed.

3.1 Spiking neural network proposal

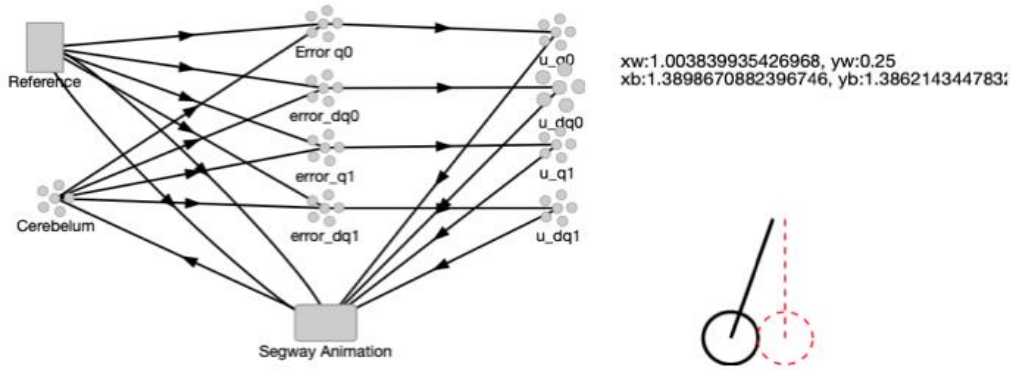

Fig. 4 SNN structure proposed for the control simulation, implemented on Nengo

Figure 4 shows the implemented SNN using the simulation software Nengo [22], which provides libraries for defining and connecting ensembles, learning rules, and synaptic activity. The software allows the designer to modify each ensemble radii to define the represented function domain. Each signal representing a space state variable of our control loop would be encoded/decoded by a neuron ensemble, whose radius would be determined by the signal's minimum and maximum values. In practice, these are limited by the robot's capacity, as its motor has a maximum angular velocity and acceleration. By default, this radius is set between [-1,1]. However, the user can redefine it to fit the actual signal domain from the state space. Next, each of the elements of the proposed structure is described in Table 1. The neurons in this structure are based on the LIF model, with the parameters implemented in the library [9]. Aditionally, two python3 (v3.8) scripts, using Numpy (v1.20) where embebbed on the Nengo (v3.1) simulation:

- Reference: A small function which return the desired vector state x_d
- MWIP Animation: Node used for MWIP simulation, with one control input (u) and four outputs $(\theta_w, \dot{\theta_w}, \theta_b, \dot{\theta_b})$

Table 1. Neuron ensembles of the proposed structure.

Ensemble of neurons	Description	Number of neurons	Radius
Cerebellum	This ensemble will encode the state vector from de MWIP. Called like this in a similar fashion as [10]	1000×4	$\frac{(-10,10)\text{rad}}{\text{s}}$
Error	For encoding the difference between actual and reference vector state	100×4	$(-\pi, \pi)$ rad
Control U	Neuron ensemble which represents the control signal for each state variable.	1000×4	(-350, 350)N/m
Total of neurons		8400	

Algorithm 1. SNN Control Loop simulation process

Algorithm 1: SNN Control Loop

Data: x_initial, x_desired, Kr, timeline, dt
Result: x_historial, u_historial
x ← x_initial
E ← Matrix[q, n](random)
D^f ← Matrix[n, q](Equation 11)
For t **in** timeline:
    Error_spikes = Encode(x(t) – x_desired(t))
    Error = Decode(Error_spikes)
    u_spikes = Encode(-$K_r \times Error$)
    u = Decode(u_spikes)
    $\dot{x} = Ax(t) + Bu$
    $x(t+1) = x(t) + \dot{x})(dt)$
    x_historial.append(x(t))
    u_historial.append(u(t))
end For

In summarizing, the synaptic weights are obtained on the initialization, using the NEF optimization described in section 2. Each space state signal would be encoded into spikes to be processed by the net, returning the pertinent decoded control signal. The implemented execution algorithm is shown in Algorithm 1.

3.2 Simulation Scenario

In order to evaluate the architecture performance, the MWIP starts from an initial state $x_{initial} = [4,0,0.1,0]$, with a desired vector state $x_d = [6,0,0,0]$. Table 2 shows the MWIP model parameters used and $g = 9.81 m/s^2$. For the control loop, $u = -K(x - x_d)$, setting $Q = I$ and $R = 0.001$, the matrix $K_r = [-100, -323.3434, -542.0927, -541.08]$, using the described methods in [20,21]. The simulation period has a duration of $t = 20s$ using euler integration with a timestep of $0.1ms$.

Table 2. MWIP model parameters used for simulation

Parameter	m_b	m_w	L	r	I_w	I_b
Value	1[kg]	2[kg]	1.2[m]	0.25[m]	$10\left[\frac{N}{m}\right]$	$10\left[\frac{N}{m}\right]$

Figure 5a) shows the generated tuning curve for one of the ensembles used in this architecture. 5b) shows the reconstructed signal and 5c) the current neural activity

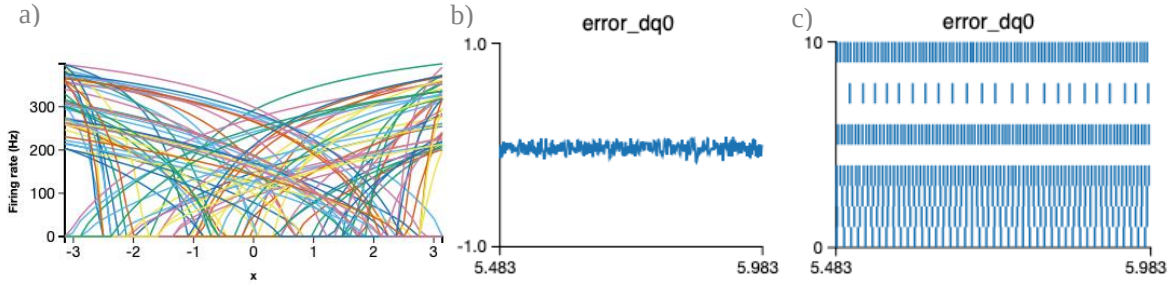

Fig. 5 a) Tuning curve for ensemble error q0, described at Table (1) b) Represented output value c) Spiking activity of 10 neurons from 1000 neurons available in the ensemble, elaborated with Nengo simulated software

Figure 6 shows the MWIP space state evolution, where the initial values successfully evolve towards the desired vector state, with a smooth transition and finishing with a relatively small error. Due to spike-based sparse coding, the control signal is stochastic and noisy (Comparing Fig 6 and Fig.7), But it successfully achieves the control task.

3.3 Varying the desired position

Synaptic weights were optimized to represent the signal's domain based on the ensemble radius described in Table 1, providing the appropriate stimuli for each ensemble and allowing the network to perform correctly for bounded error and control signals. However, as the difference between the MWIP's initial condition x and the desired position x_d increases, the corresponding error and control signals go beyond the encodable domain. In order to test how far the system can go, 100 simulations of 10 seconds each are performed with the same parameters as before, except for $x_d = [\theta_w^d, 0,0,0]$, in which the desired angular position of the wheel value θ_w^d varies from 2 to 7 radians. MWIP initial state remains as $x_{initial} = [4,0,0.1,0]$, represented as a red dotted vertical line in Fig. 8, which shows the Root Mean Squared value of steady-state error at $t = 10s$:

$$RMS = \sqrt{\sum(x_d - x)^2} \tag{13}$$

Here we can observe an inner region where error tends to almost zero values, which means that for this desired state range, all the error and control signals were encodable, and the MWIP could reach the desired state. Outside this region, as the desired state goes further, error and control signals are out of the set radii, and the ensembles cannot encode and decode these signals properly, leading to divergence.

4 Conclusions and Directions for Further Research

This article uses plausible neurological models to assemble a neural structure that can perform control regulation tasks over a robotic underactuated system. Control theory meets neuroscience, as neural architectures are designed by allocating resources to specific requirements in a control loop algorithm, signal representation and state estimation. It is shown that a correct radius specification for each ensemble reflects on the precision of its output control signal. Nonetheless, the produced output control signal for the robotic system has a stochastic noisy nature, similar to output control strategies such as sliding modes [17]. While noise added by the neural dynamics might be problematic, it adds a small value, avoiding singular matrices during the encoding and decoding process used in NEF [7]. This also might be beneficial to avoid an overfitting case, and foster a quick update in synapses. We consider this work leads to possible future research, like a Kalman filter implementation in SNN networks for signal cleaning, or online dynamics estimation of the plant in order to compute the control signal, exploring other control techniques such as ADRC [29] for unknown plant dynamics and possible neuromorphic implementation in FPGA or ASIC devices.

b)

a)

5 Acknowledgements

This work was supported by the Instituto Politécnico Nacional (IPN) and Secretaría de Investigación y Posgrado (SIP-IPN) under the projects: 20210124, 20221780, 20211657, 20220268, 20212044, 20221089, 20210788 and 20220226, the Comisión de Operación y Fomento de Actividades Académicas (COFAA-IPN), also the Consejo Nacional de Ciencia y Tecnología (CONACYT-México) and 6005 (FORDECYT-PRONACES). Juárez-Lora would like to thank to CONACYT for the grant proportioned for his PhD studies.

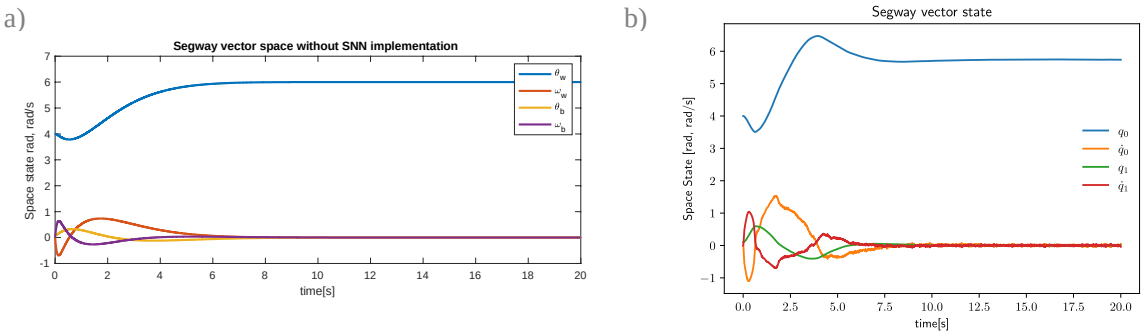

Fig. 6. Evolution of the Vector State a) simple control loop simulation without SNN. b) using the proposed SNN structure

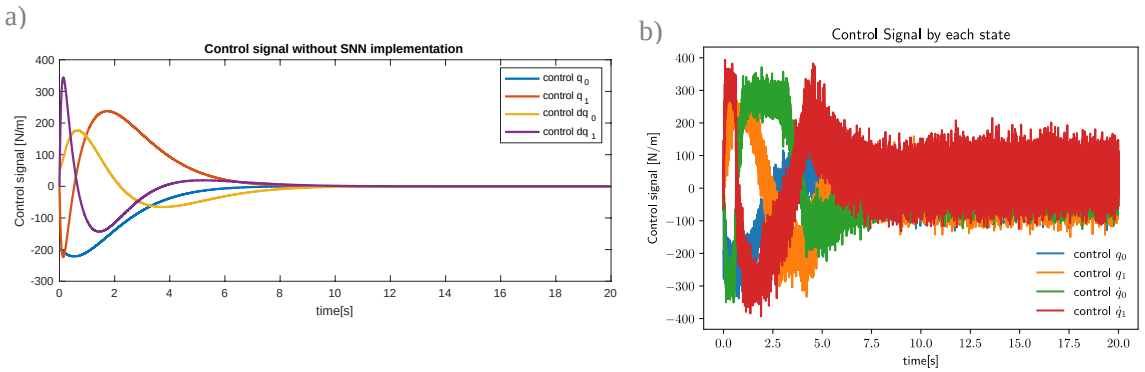

Fig. 7. Evolution of control signal for each state. A) simple control loop simulation without SNN. b) simulating the proposed SNN structure

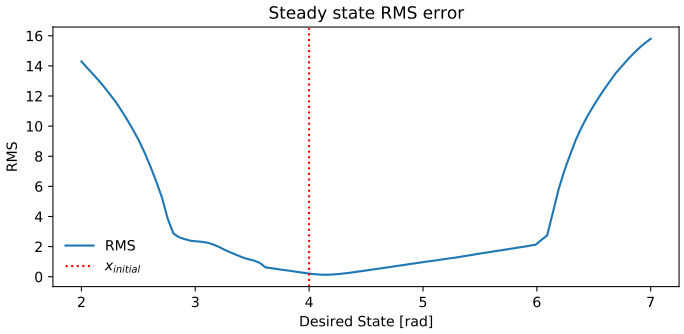

Fig. 8 Resulting RMS value, varying the desired state from a fixed initial state between 100 iterations.

6 Data Avilability

The name of the repository and accession number can be found below: Github https://github.com/AlejandroJuarezLora/IJCOPI.git.

References

1. Yexin Yan, Terrence Stewart, Xuan Choo, Bernhard Vogginger, Johannes Partzsch, Sebastian Hoeppner, Florian Kelber, Chris Eliasmith, Steve Furber, and Christian Mayr. Comparing loihi with a spinnaker 2 prototype on low-latency keyword spotting and adaptive robotic control. Neuromorphic Computing and Engineering, 2021. URL: https://iopscience.iop.org/article/10.1088/2634-4386/abf150, doi:10.1088/2634-4386/abf150.
2. Jesus L. Lobo, Javier Del Ser, Albert Bifet, Nikola Kasabov, Spiking Neural Networks and online learning: An overview and perspectives, Neural Networks, Volume 121, 2020, Pages 88-100, ISSN 0893-6080, https://doi.org/10.1016/j.neunet.2019.09.004
3. Galindo, S., Toharia, P., Robles, O., Ros, E., Pastor, L., & Garrido, J. (2020). Simulation, visualization and analysis tools for pattern recognition assessment with spiking neuronal networks. Neurocomputing.
4. Bogdan, P., Marcinnò, B., Casellato, C., Casali, S., Rowley, A., Hopkins, M., Leporati, F., D'Angelo, E., & Rhodes, O. (2021). Towards a Bio-Inspired Real-Time Neuromorphic Cerebellum. Frontiers in Cellular Neuroscience, 15, 130.
5. Amunts, J. (2019). The Human Brain Project—Synergy between neuroscience, computing, informatics, and brain-inspired technologies. PLOS Biology, 17, 1-7.
6. Falotico, E., Vannucci, L., Ambrosano, A., Albanese, U., Ulbrich, S., Vasquez Tieck, J., Hinkel, G., Kaiser, J., Peric, I., Denninger, O., Cauli, N., Kirtay, M., Roennau, A., Klinker, G., Von Arnim, A., Guyot, L., Peppicelli, D., Martínez-Cañada, P., Ros, E., Maier, P., Weber, S., Huber, M., Plecher, D., Röhrbein, F., Deser, S., Roitberg, A., Smagt, P., Dillman, R., Levi, P., Laschi, C., Knoll, A., & Gewaltig, M.O. (2017). Connecting Artificial Brains to Robots in a Comprehensive Simulation Framework: The Neurorobotics Platform. Frontiers in Neurorobotics, 11, 2.
7. Voelker A.R., Eliasmith C. (2021) Programming Neuromorphics Using the Neural Engineering Framework. In: Thakor N.V. (eds) Handbook of Neuroengineering. Springer, Singapore. https://doi.org/10.1007/978-981-15-2848-4_115-1
8. Daniel Rasmussen. (2019). NengoDL: Combining deep learning and neuromorphic modelling methods.
9. (2019). The Nengo neural simulator — Nengo. Available at https://www.nengo.ai/.
10. DeWolf, T., Stewart, T., Slotine, J.J., & Eliasmith, C. (2016). A spiking neural model of adaptive arm control. Proceedings of the Royal Society B: Biological Sciences, 283(1843), 20162134.
11. C. C. Cheah, C. Liu, & J. J. E. Slotine (2006). Adaptive Tracking Control for Robots with Unknown Kinematic and Dynamic Properties. The International Journal of Robotics Research, 25(3), 283-296.
12. Tieck, J., Steffen, L., Kaiser, J., Roennau, A., & Dillmann, R. (2018). Controlling a Robot Arm for Target Reaching without Planning Using Spiking Neurons. In 2018 IEEE 17th International Conference on Cognitive Informatics Cognitive Computing (ICCI*CC) (pp. 111-116).
13. Tieck, J., Secker, K., Kaiser, J., Roennau, A., & Dillmann, R. (2021). Soft-Grasping With an Anthropomorphic Robotic Hand Using Spiking Neurons. IEEE Robotics and Automation Letters, 6(2), 2894-2901.
14. DeWolf, T., Jaworski, P., & Eliasmith, C. (2020). Nengo and Low-Power AI Hardware for Robust, Embedded Neurorobotics. Frontiers in Neurorobotics, 14, 73.
15. Davies, M., Wild, A., Orchard, G., Sandamirskaya, Y., Guerra, G., Joshi, P., Plank, P., & Risbud, S. (2021). Advancing Neuromorphic Computing With Loihi: A Survey of Results and Outlook. Proceedings of the IEEE, 109(5), 911-934.
16. Sira-Ramírez Hebertt (2017). Active disturbance rejection control of dynamic systems: a flatness-based approach. Oxford: Butterworth-Heinemann.
17. Li, K., Ri, S., Huang, J., Wang, Y., Kim, M., & An, S. (2014). Terminal Sliding Mode Control of Mobile Wheeled Inverted Pendulum System with Nonlinear Disturbance Observer. Mathematical Problems in Engineering, 2014, 284216.
18. Krafes, S., Chalh, Z., & Saka, A. (2018). A Review on the Control of Second Order Underactuated Mechanical Systems. Complexity, 2018, 9573514.
19. Steven L. Brunton, J. (2019). Linear Control Theory. Cambridge University Press.
20. Peter Benner Jing-Rebecca Li, & Thilo Penzl (2008). Numerical solution of large-scale Lyapunov equations, Riccati equations, and linear-quadratic optimal control problems.. Numerical Linear Algebra with Applications, 15(9), 755-777.
21. William F. Arnold, A. (1984). Generalized Eigenproblems Algorithms and software for algebraic Riccati Equations. Proceedings of the IEEE, 72(12), 1746-1754.

22. Rasmussen, D. NengoDL: Combining Deep Learning and Neuromorphic Modelling Methods. Neuroinform **17,** 611–628 (2019). https://doi.org/10.1007/s12021-019-09424-z
23. Izhikevich, E., & Desai, N. (2003). Relating stdp to bcm. Neural computation, 15, 1511-23.
24. Blais B. Shouval H., & Cooper L. N ((1999)). The role of presynaptic activity in monocular deprivation: Comparison of homosynaptic and heterosynaptic mechanisms. Proc. Natl. Acad. Sci, 96, 1083–1087.
25. Wulfram Gerstner, R., & Liam Paninski (2006). Neuronal Dynamics (online book): From single neurons to networks and models of cognition. Cambridge University Press.
26. Brette, R. (2015). Philosophy of the Spike: Rate-Based vs. Spike-Based Theories of the Brain. Frontiers in Systems Neuroscience, 9, 151.
27. Eliasmith C, A. (2004). Neural engineering: computation, representation, and dynamics in neurobiological systems. Cambridge, MA: MIT Press.
28. Eliasmith C (2013). How to Build a Brain: A Neural Architecture for Biological Cognition. Oxford University Press.
29. Carlos Aguilar-Ibanez, Hebertt Sira-Ramirez, José Ángel Acosta, & Miguel S. Suarez-Castanon (2021). An algebraic version of the active disturbance rejection control for second-order flat systems. International Journal of Control, 94(1), 215-222.
30. A. Hazan and E. E. Tsur, "Neuromorphic Spike Timing Dependent Plasticity with adaptive OZ Spiking Neurons," 2021 IEEE Biomedical Circuits and Systems Conference (BioCAS), 2021, pp. 01-04, doi: 10.1109/BioCAS49922.2021.9644944.
31. Zamarreño-Ramos, C., Camuñas-Mesa, L. A., Pérez-Carrasco, J. A., Masquelier, T., Serrano-Gotarredona, T., & Linares-Barranco, B. (2011). On Spike-Timing-Dependent-Plasticity, Memristive Devices, and Building a Self-Learning Visual Cortex. In Frontiers in Neuroscience (Vol. 5). Frontiers Media SA. https://doi.org/10.3389/fnins.2011.00026
32. Kim, H., Mahmoodi, M. R., Nili, H., and Strukov, D. B. (2021). 4k-memristor analog-grade passive crossbar circuit. Nature Communications 12. doi:10.1038/s41467-021-25455-0
33. Eshraghian, Jason K., Max, Ward, Emre, Neftci, Xinxin, Wang, Gregor, Lenz, Girish, Dwivedi, Mohammed, Bennamoun, Doo Seok, Jeong, and Wei D., Lu. "Training Spiking Neural Networks Using Lessons From Deep Learning." (2021).